Logical Foundations of Intermittent Computing

Myra Dotzel

CMU-CS-26-130

August 2026

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Limin Jia, Co-chair
Farzaneh Derakhshan, Co-chair
Jan Hoffmann
Frank Pfenning
Brigitte Pientka (McGill University)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This work was generously funded in part through the Office of Naval Research grant N000142412297 and National Science Foundation Graduate Research Fellowship Program grants DGE1745016 and DGE2140739.

The views and conclusions expressed in this document are those of the author and do not necessarily reflect those of the U.S. government or any sponsoring organization.

Keywords: intermittent computing, modal crash type, adjoint logic, logical relation, shared memory concurrency

"You have to remember that one thing about being a painter is that you're part of this wonderful tradition from Vermeer to Velasquez to Degas to Picasso to Rembrandt, on and on and on – to Chinese painting, Japanese woodblocks, Mayan ceramics.

And the problem suddenly suggests to you, what in the world am I thinking of, picking up a brush when there's a painter like Rembrandt out there?

I mean, why do it? Because it's such fun."

– Wayne Thiebaud

Abstract

Intermittent computing enables applications to thrive in energy-scarce and inaccessible environments, where manual device maintenance is infeasible. Instead of relying on batteries, devices in these applications are instead powered by energy from the environment. They harvest energy from the environment into a rechargeable buffer, compute when energy is available, and power off when energy is depleted, halting program execution until more energy can be retrieved. To enable correct program re-execution despite these potentially frequent and arbitrary power failures, runtime support is needed to save and restore necessary state.

This thesis lays a logical foundation for intermittent computing by building type systems and runtime systems to guarantee correct program execution across diverse features and behaviors, including sequential and concurrent execution, as well as inputs and effectful outputs. First, we explore *sequential*, intermittent computing and model checkpoint, crash, restore, and re-execute operations as computation on crash types. We draw inspiration from adjoint logic and define crash types by introducing two adjoint modality operators to model operations on nonvolatile and volatile memories. We define a crash type system for a core calculus. To prove the correctness of intermittent systems, we define a novel logical relation for crash types. Throughout this thesis, we extend crash types to consider more realistic program and system behaviors, such as more flexible checkpointing policies, common logging styles, I/O and branching.

Lastly, we provide the first provably-correct system for *concurrent*, intermittent program execution, which is needed for supporting hardware-triggered interrupts and accesses to shared memory which are common to many embedded systems applications. We present a co-designed runtime system and type system that together support the provably correct intermittent execution of concurrent programs running on shared memory. This system promotes a more flexible programming model and supports a broader spectrum of task re-execution behaviors than is considered by prior work. We provide the first formal definition and proofs of correctness for concurrent, intermittent execution.

Acknowledgments

I am grateful to the many wonderful people I have interacted with over these past five years who made this journey possible. I would first like to thank my thesis committee: Limin Jia and Farzaneh Derakhshan, my advisors, as well as Jan Hoffmann, Frank Pfenning, and Brigitte Pientka, for their invaluable insights and for taking the time to carefully read and offer feedback on my writing. I thank my advisors for their support and dedication to my growth throughout my PhD. They always provided the most honest feedback on my writing and talks, and paid attention to training me in all other aspects of becoming a researcher. The lessons I learned from them during my PhD I will take with me for the rest of my life. An important lesson is that it doesn't matter where a person starts, as long as they get to where they need to be.

I thank my collaborators, particularly Milijana Surbatovich for being a dedicated and patient mentor from the start. Thank you for your time and guidance from when I was first getting into intermittent computing, and for your friendship. I also thank André Platzer and Stefan Mitsch for introducing me to cyberphysical systems research. Those project meetings have brought me more joy and hope over the course of my PhD than I can convey here in writing.

I thank all of my other mentors from over the course of my PhD, particularly Jan Hoffmann and David Kahn who first introduced me to the field of programming languages in my senior year of college. I am grateful for your time and mentorship from early on that got me on this path. I thank my SIGPLAN mentor Alexandra Silva for her patient listening, grounding advice, and for always making time to meet with me even in the busiest of times. I thank Stephanie Balzer for her support and friendship throughout my PhD. I thank Frank Pfenning for the opportunity to TA and practice giving lectures in his Foundations of Security and Privacy course.

I thank all of my friends, therapist, and family for their support throughout my entire PhD journey. I thank my brother for his adventurous spirit, good humor, and for reminding me that the play is just as important as the work. Thank you Mom and Dad for everything; for investing so much time into developing my interests, for valuing a good education, and for always being there for me in all of the challenging moments of life. Thank you Dad for your wise advice about grad school and for your empathy throughout. Thank you Mom for being not only a parent but a girl's best friend. She taught me to not take opportunities for granted and that happiness is always worth fighting for.

Lastly, I thank my partner Varun for his enduring love and support over the course of my PhD, for standing by me through the thick and the thin, and for taking care of me while writing this thesis and always. Our move to Ann Arbor while preparing for my thesis defense certainly would not have been possible without your support. I am lucky that I get to spend my life with you. And to our sweetest cat Zena. While she spent most of the time I was writing this thesis in time-out, at other times she was provided emotional support and reminded me to take breaks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Intermittent computing is an emerging paradigm that brings the promise of complex and sophisticated computation to remote and inaccessible environments. The applications are diverse, spanning from tiny medical implants [36, 99], to smart wildlife monitors [87], to smart traffic light networks in cities [6], to nano-satellites in space [88, 130]. In each of these settings, manual device maintenance to replace a battery or fix a firmware bug is infeasible; for example, regularly replacing the battery of a medical implant is invasive and financially expensive for patients. In more remote applications, invading the territory of orca pods to replace batteries in wildlife sensors is hazardous to both the maintainer as well as to the surrounding ecosystem. In smart civil infrastructure, tracking down and replacing batteries in all 13,000+ stoplights in New York City is labor-intensive and would cause traffic jams. Furthermore, nano-satellites in space could even be impossible to retrive once deployed.

Therefore, to reduce the cost of maintaining devices in such applications, the devices themselves are designed to be self-sufficient and durable, requiring little to no manual maintenance over prolonged periods of time. Furthermore, to meet the high assurance requirements of each of these domains, programs running on these devices must be guaranteed bug free prior to deployment, therefore demanding *formal static correctness guarantees*.

To eliminate the need for regular battery replacement, intermittent computing relies on batteryless, energy-harvesting devices (EHDs) [7, 35, 49, 58, 63, 78, 87, 110, 111, 131]. Instead of relying on a battery, EHDs operate solely on energy harvested from the environment, e.g., through solar or kinetic sources, which is stored in a *re-chargeable energy buffer*. When the buffer has energy, the device powers on and begins executing a program, consuming the available energy until it is depleted. When the buffer is empty, the device powers off until more energy is retrieved, at which point the device turns on and the program may attempt to re-execute. To make progress despite these frequent and arbitrary *power failures*, these devices typically compute on a combination of volatile and nonvolatile memory [51, 56, 123]. Whereas volatile memory (e.g., registers and stack) is lost on power failure, nonvolatile memoy (e.g. program code and data) persists and can be used by the next re-execution.

1.1 Challenges of Correct Intermittent, Computing

Unfortunately, the very reliance on environmental energy which makes EHDs attractive also impedes these applications' goals for correctness. In particular, the problem stems from a reliance on volatile and nonvolatile state; if left unprovisioned, the device will re-execute on an inconsistent memory state and compute incorrect results. Re-execution of code on unprovisioned systems could spell unintended or disastrous consequences for users of these applications, ranging from a nano-satellite logging an incorrect distance computation, to notifying a forgetful patient to take an extra dose of medicine even after they have already done so.

To enable *correct* behavior in these settings, an EHD requires *intermittent system* [106] support to save necessary copies of values in *checkpointed locations*, while also maintaining memory consistency [77, 119] despite these partial clears. To reason about intermittent program execution, recent work [20, 37, 119, 120, 121] provides formal frameworks that define correctness as follows:

> *"Any intermittent execution of a correct program can be simulated by a continuously-powered execution of it".* [119]

This criterion ensures that any *sequential*, intermittent execution of a "correct" program can generate results that are equivalent to those expected from a continuously-powered execution, regardless of the number of power failures endured. However, proving that a program is correct according to this criterion for sequential intermittent execution is challenging due to the re-execution of code regions, which could cause effectful operations to repeat or computation to rely on results from failed prior executions. Therefore, proving that a program is correct is inherently tied to which variables it checkpoints as well as underlying runtime supports of the system. Moreover, in order to provide guarantees that are useful to real programs, the reasoning must take care to employ reasonable assumptions regarding program behavior as well as hardware constraints. For example, it is expensive to manage a checkpoint for every single variable in a program, leading researchers to investigate smarter checkpointing policies [77, 82, 119]. As another example, embedded systems applications are often concurrent in nature, relying on interrupts to preempt a main application to process inputs from hardware peripherals.

However, proving *concurrent* intermittent programs will behave correctly in the presence of shared memory only becomes more challenging, requiring supports for more sophisticated re-execution behaviors than those considered by sequential intermittent systems. Furthermore, concurrency demands new strategies for checkpointing, as the notion of shared memory breaks checkpoint mechanisms designed for sequential intermittent systems. Prior work has provided a few such systems that tend towards restricted programming models and limited program features [109, 129]. However, no one has developed formal correctness definitions of concurrent, intermittent programs, leaving expected program behavior unknown and applications susceptible to bugs.

1.2 Thesis Statement and Contributions

The overarching goal of this thesis is to uncover the logical foundations of intermittent computing and to build systems that support correct intermittent execution. If the behaviors underlying intermittence can be explained by logic, we can provide richer supports and correctness guarantees for these systems, laying groundwork for examining even more complex program behaviors. If we provide well-defined correctness principles and clear correctness proofs, the resulting systems will be more easily extensible and more dependable. This thesis designs type systems and runtime systems for supporting correct intermittent execution. The contributions and vision of this thesis are summarized by the following statement:

Type-based abstractions rooted in adjoint logic provide a uniform way to statically reason about the safety of intermittent computation that scales to more realistic scenarios.

1.2.1 Contributions and Outline

The contributions of this thesis aid in developing more general and expressive provably correct intermittent systems with well-defined correctness properties. In this section, we explain the progression of this dissertation and highlight contributions from each chapter.

In Chapter 2, we provide more background on sequential intermittent execution, guided by running examples to illustrate key challenges and how correct intermittent systems are designed to overcome them. To set the stage for the following chapters, we present a base syntax and key operational semantics, as well as primers for key concepts that recur throughout the dissertation.

Chapter 3 presents our first key contribution: a logical interpretation of *sequential*, intermittent computing. To provide a general abstraction for reasoning about failures, we present a calculus defined according to the key operations of intermittent execution: *crash*, *restore*, and *re-execute*. To capture the behavior of a power failure in the type of a computation, we introduce *crash types* that are defined in terms of these operations according to the prevailing memory pattern – some values persist across power failures while others do not. In particular, nonvolatile memory state persists across power failures, while volatile memory does not; results from completed computations (or checkpointed values) should persist across power failures, while partially computed results should not. We classify the former as *stable* and the latter as *unstable*, and observe that the interactions between these components bear close resemblance to shifts $\uparrow, \downarrow$ in adjoint logic [18, 107] for switching between different worlds. In particular, the checkpoint and restore operations correspond to these shifts and are internalized in the definition of crash types, depending on the underlying logging style of the intermittent system. For example, in a *redo-logging* style, updates are made to a separate volatile copy of checkpointed locations that is automatically lost on power failure. In this setting, volatile memory is considered unstable whereas nonvolatile memory is stable, and a commit operation moves data from volatile memory (unstable) into nonvolatile memory (stable) while restore performs the inverse. We develop an operational semantics and type system for this language, and a logical relation defined over crash types for reasoning about correctness. We prove type soundness according to progress and preservation, and correctness according the fundamental theorem of logical relation and adequacy. The text in this chapter is based on published paper [40].

To bring crash types closer to reasoning about real intermittent systems, Chapter 4 examines

a more flexible checkpointing policy than is assumed by the core crash types calculus which requires checkpointing fewer variables and is used by real intermittent systems [77]. By extending the type system and redefining policies, we are able to reuse the same crash types and logical relation from Chapter 3 to establish similar correctness guarantees. This chapter is based on the published paper [44].

In Chapter 5, we examine crash types through the lens of a more popular logging style: *undo-logging*. In *undo-logging*, updates are made directly to nonvolatile memory and will be reverted by checkpointed copies on reboot. In this setting, the nonvolatile memory that have checkpointed copies are considered unstable, whereas their copies are stable. To match the execution behavior of resetting all unstable values across volatile and nonvolatile memory, we redefine the crash type with an adjusted ordering of modalities. We define a crash types calculus for undo-logging, and prove progress and preservation. Furthermore, we show that the key operations of intermittent computing for undo-logging continue to have interpretations in adjoint logic.

In Chapter 6, we extend crash types with additional program features that are common to embedded systems applications: sensor inputs, outputs, and more flexible branching behaviors. The resulting system supports more expressive system behaviors than previous provably correct intermittent systems, accepting more correct programs and allowing fewer variables to be checkpointed. Furthermore, outputs are a novel extension for which there currently exist no provably correct intermittent system supports. We define a crash types calculus that supports all of these features, and prove progress and preservation.

Chapter 7 motivates the last main contribution of this thesis, which is a co-designed runtime system and type system for concurrent, intermittent program execution, where programs are specified as tasks that access shared data. To our knowledge, this system provides the first provably correct supports for concurrent, intermittent computing. We build upon the notions of stable and unstable data for undo-logging and information flow type systems to support nestings of more complex re-execution behaviors that do not arise in the sequential setting. Furthermore, we define and prove the correctness of the resulting system. The resulting system is capable of supporting more flexible program behaviors and memory access patterns than do the few existing concurrent, intermittent systems, whose correctness is not guaranteed.

Chapter 8 presentes related work, Chapter 9 presents directions for future work, lastly and Chapter 10 concludes the document with a final discussion of key contributions and insights of this thesis.

The contributions of this thesis are partially based on the published papers [40, 44].

Chapter 2

Background

This thesis aims to provide language and runtime system supports for correct intermittent computing on EHDs. Such supports ensure that every intermittent execution of a program will compute the same results that would otherwise be computed on continuous power. Supporting correct intermittent execution is especially challenging due to a combination of non-deterministic power failures and memory accesses across volatile and nonvolatile memory, which if handled naively, could leave programs susceptible to memory consistency bugs.

This chapter details these challenges and presents runtime mechanisms commonly used by intermittent systems to uphold correct program behavior. We first present a basic syntax of a simple imperative language used for programming embedded devices with re-executable code regions annotated. Then, we define an operational semantics for this language and present an example program. This operational semantics will serve as a building block for future chapters, whose systems formalize intermittent execution in these terms. We conclude with a primer on adjoint logic, whose concepts recur throughout the contributions of this thesis.

2.1 Intermittent Execution Models

To allow long-running programs to make progress and compute correct results despite frequent and arbitrary power failures, the on-board intermittent system must save a snapshot of execution state, such as by creating a *checkpoint* to be restored upon reboot. When, where, and how such checkpoints are updated is determined by the *intermittent execution model* under which programs execute. There are two prevailing execution models. Under a *just-in-time (JIT)* execution model, a program would resume execution from the line of failure, requiring all of volatile state to be saved just before power failure [13, 14, 65, 127]. Alternatively, under an *atomic region* model, a program would re-execute from the beginning, relying on programmer-defined *atomic regions* to checkpoint nonvolatile state and the region command prior to its execution [77, 82, 109, 125] to remember from where to re-execute and with what state.

These execution models have their benefits and tradeoffs, leading programmers to sometimes favor one over the other depending on application demands and available hardware. For example, JIT regions are advantageous in being easy and efficient to implement, requiring relatively low checkpoint management and relying on the hardware to handle power failure and reboot be-

havior, instead of requiring special programmer instrumentation like atomic regions. Though, to execute properly, the JIT execution model requires special hardware, e.g., a voltage comparator, in order to issue the necessary low power signal and checkpoint state before power fails. Even though most EHDs are equipped with these hardware features, a programmer may additionally favor atomic regions over JIT checkpointing to run code that should not be interrupted by power failure, e.g., timing-sensitive reactive tasks [59, 71, 120] and peripheral interactions [19, 59, 80], but at the expense of requiring programmer instrumentation to decide where to place atomic regions and which variables to checkpoint. To allow applications to benefit from both models of execution, state-of-the-art intermittent systems often rely on a hybrid "*JIT + atomics*" model, defaulting to JIT checkpointing outside of atomic regions. This is the model assumed throughout this thesis.

2.2 Syntax

We summarize a basic initial syntax for sequential, intermittent program execution in Fig. 3.2.

values	v	::=	$\mathsf{n} \mid \mathsf{tt} \mid \mathsf{ff} \mid x$
expressions	e	::=	$v \mid e \odot e \mid \oslash e$
commands	c	::=	$\mathsf{skip} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \mid \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \mid x := e \mid c_1; c_2$
programs	p	::=	$\mathsf{skip} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}} \mid c; p$
nonvolatile memory	N	::=	$\cdot \mid \ell \hookrightarrow v, \mathsf{N}$
volatile memory	V	::=	$\cdot \mid \ell \hookrightarrow v, \mathsf{V}$
memory location mapping	γ	::=	$\cdot \mid x \mapsto \ell, \gamma$
execution mode	$\mathtt{Md}$	::=	$\mathsf{aID} \mid \mathsf{jit}$

Figure 2.1: A basic syntax for intermittent programs and execution

Values and expressions include integers, booleans, variables, and binary ($\odot$) and unary ($\oslash$) operations. Commands include skip instructions, mutable let bindings, conditionals, assignments, and sequences of commands. A program consists of sequenced JIT blocks c; p and programmer-defined atomic regions, denoted $\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}$ with a unique region identifier aID, an enclosed command c, and a variable policy *aPol* indicating which variables to checkpoint and how other variables should be accessed within the region (to be detailed later). This policy can be obtained by direct programmer annotation or automatic inference using established techniques [121]. The execution mode $\mathtt{Md}$ of the currently-running command indicates whether it is an atomic (aID) or JIT (jit) region, which will determine how checkpointing is handled during execution. Nonvolatile and volatile memories are denoted as N and V, respectively, which map memory locations to values. The mapping γ associates program variables with their memory locations, which will be needed to separate high-level reasoning about a program's source code from its execution.

For example, consider the simple intermittent program in Fig. 2.2. The program has four variables that a programmer declares as being stored in nonvolatile memory: x, y, and z of type `int` and u of type `bool`. It consists of two code blocks: a regular code block executed in JIT mode and an atomic region. For now, the atomic region policy is left blank (_).

```
let w=not u in
    if w then
        x:=x+y;
        w:=ff
    else
        skip;
skip;
```

```
start_atom(aID, _)(
    y:=y+z;
    let w= x-y in
        if w>0 then
            u:=tt
        else
            u:=ff);
end_atom
```

Figure 2.2: An example program with a JIT block (left) and an atomic region (right)

2.3 Sequential, Intermittent Execution by Example

To ensure correct behavior despite frequent and nondeterministic power failure, an intermittent runtime system must checkpoint portions of volatile and nonvolatile state. To illustrate why, this section examines the execution of the simple program in Fig. 2.2. Since each region has a different (re-)execution behavior, we discuss them separately.

2.3.1 JIT Region Execution

A program executing with JIT checkpointing resumes execution from the point where power failed. To enable correct resumption, the intermittent system must save all of volatile state and the continuation command right before power fails, which would otherwise be wiped. Formally, the intermittent execution of JIT regions relies on a nonvolatile *checkpoint context* denoted for now as $\mathsf{K} = (\mathsf{V}_k, c_k)$, containing checkpointed copies of volatile memory and a continuation command to be restored on reboot.

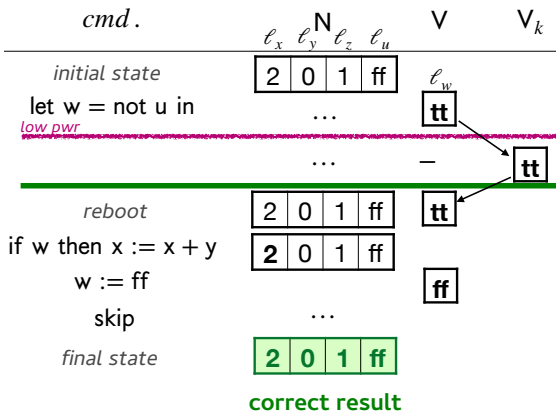

Figure 2.4: Example JIT region execution.

We provide a selection of operational semantic rules for intermittently executing JIT regions in Fig. 2.3. To model relevant aspects of the execution state, each rule steps *configurations* of the form $\gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \mathsf{V} \mid c$, containing a variable-location mapping, the current execution mode, a checkpoint context, nonvolatile and volatile memories, and the next command to execute. The command evaluation rules are defined with a small-step operational semantics ($\Rightarrow$) that step configurations to configurations. The expression evaluation rules use a big-step operational semantics where the judgment $\gamma, \mathsf{N}, \mathsf{V} \vdash e \Downarrow v$ states that an expression e evaluates to a value in one big step, with respect to the current memories N and V and the variable-location mapping γ which will be used to look up values from locations of variables when they are read in a program. For example, to read a variable x whose location is in volatile memory, the rule D-V-READ applies to first deter-

Command evaluation rules.

$$\frac{\mathsf{K}' = (\mathsf{V}, c)}{\gamma \mid \mathsf{jit} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid c \Rightarrow \gamma \mid \mathsf{jit} \mid \mathsf{K}' \mid \mathsf{N} \mid \cdot \mid \mathsf{reboot}} \text{ (D-JIT-LOW-PWR)}$$

$$\frac{\mathsf{K} = (\mathsf{V}, c)}{\gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \cdot \mid \mathsf{reboot} \;\Rightarrow\; \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid c} \text{ (D-REBOOT)}$$

$$\frac{\ell_x \; \textit{fresh} \qquad \gamma, \mathsf{N}, \mathsf{V} \vdash e \Downarrow v}{\gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \;\Rightarrow\; \gamma, x \hookrightarrow \ell_x \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \mathsf{V}, \ell_x \hookrightarrow v \mid c} \text{ (D-LET)}$$

$$\frac{\gamma, \mathsf{N}, \mathsf{V} \vdash e \Downarrow v \qquad \ell_x \in \mathsf{dom}(\mathsf{V})}{\gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \mathsf{V} \mid x := e \Rightarrow \gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \mathsf{V}[\ell_x \hookrightarrow v] \mid \mathsf{skip}} \text{ (D-ASSIGN-V)}$$

$$\frac{\gamma, \mathsf{N}, \mathsf{V} \vdash e \Downarrow v \qquad \ell_x \in \mathsf{dom}(\mathsf{N})}{\gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N} \mid \mathsf{V} \mid x := e \Rightarrow \gamma \mid \mathtt{Md} \mid \mathsf{K} \mid \mathsf{N}[\ell_x \hookrightarrow v] \mid \mathsf{V} \mid \mathsf{skip}} \text{ (D-ASSIGN-N)}$$

Expression evaluation rules.

$$\frac{\gamma(x) = \ell_x \qquad \mathsf{V}(\ell_x) = v}{\gamma, \mathsf{N}, \mathsf{V} \vdash x \Downarrow v} \text{ (D-V-READ)} \qquad \frac{\gamma(x) = \ell_x \qquad \mathsf{N}(\ell_x) = v}{\gamma, \mathsf{N}, \mathsf{V} \vdash x \Downarrow v} \text{ (D-N-READ)}$$

$$\frac{\gamma, \mathsf{N}, \mathsf{V} \vdash e_1 \Downarrow v_1 \qquad \gamma, \mathsf{N}, \mathsf{V} \vdash e_2 \Downarrow v_2 \qquad v = [\![v_1 \odot v_2]\!]}{\gamma, \mathsf{N}, \mathsf{V} \vdash e_1 \odot e_2 \Downarrow v} \text{ (D-BINARY)}$$

$$\frac{\gamma, \mathsf{N}, \mathsf{V} \vdash e \Downarrow v_0 \qquad v = [\![\oslash v_0]\!]}{\gamma, \mathsf{N}, \mathsf{V} \vdash \oslash e \Downarrow v} \text{ (D-UNARY)}$$

Figure 2.3: Selected operational semantic rules for JIT regions, commands and expressions

mine the location of x according to γ, which is ℓ_x, which it then look up the corresponding value in volatile memory: $\mathsf{V}(\ell_x) = v$. The rule for reading values in nonvolatile memory (D-N-READ) is similar. The rules for evaluating binary and unary expressions D-BINARY and D-UNARY are standard.

We highlight key execution steps of the example JIT region in Fig. 2.4 explained together with the operational semantic rules for commands from Fig. 2.3. Each row in Fig. 2.4 indicates how the state changes as a result of running the command on the left, with key changes **bolded**. Let us associate each variable with a memory location $\gamma = (x \hookrightarrow \ell_x, y \hookrightarrow \ell_y, z \hookrightarrow \ell_z, u \hookrightarrow \ell_u)$ where γ will uphold the invariant $\mathsf{range}(\gamma) = \mathsf{dom}(\mathsf{N}) \cup \mathsf{dom}(\mathsf{V})$ throughout the program's execution, and suppose the initial memory state is $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 0, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow \mathsf{ff}$.

The first command to run is a let construct, which binds the results of $\mathsf{not}\,\mathsf{u}$ to a new volatile location ℓ_w corresponding to the program variable w. To evaluate a let construct, the D-LET rule applies to first evaluate the bound expression (e.g., $\mathsf{not}\,\mathsf{u}$) to a value (e.g., tt), and then stores the result in a new volatile location (e.g., ℓ_w). Execution continues with the resulting configuration.

Suppose that the hardware issues a low power signal immediately after, indicating that power failure is iminent. On this signal, an intermittent system saves all necessary state to resume execution from the point where power had failed, as modeled by the rule D-JIT-LOW-PWR. The runtime command reboot indicates that the system will soon encounter a power failure, signaling the system to stash the current volatile memory (e.g., $\ell_w \hookrightarrow$ tt) and continuation command (e.g., if w then...) to the checkpoint context.

When the energy buffer replenishes, the device reboots and prepares the state for re-execution by the rule D-REBOOT, restoring the volatile state V_k and the continuation command c_k from the checkpoint context. Execution resumes with the conditional, first evaluating w to tt, and then running the first branch x := x + y; w := ff, which writes 2 to ℓ_x (via D-ASSIGN-N) and ff to ℓ_w (via D-ASSIGN-V). At the end of the region, the variable w falls out of scope, resulting in the final state $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 0, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow$ ff. These results are *correct*, as they match those expected of a single continuously-powered execution.

2.3.2 Atomic Region Execution

Alternatively, atomic regions are needed to execute code that should not be interrupted by power failure, at the expense of a more complex system design with an increased surface for errors. Without care, re-execution of atomic regions can violate memory consistency. Because updates to nonvolatile memory survive power failure, some code patterns cause an intermittent execution to compute different results than a single, continuously-powered execution.

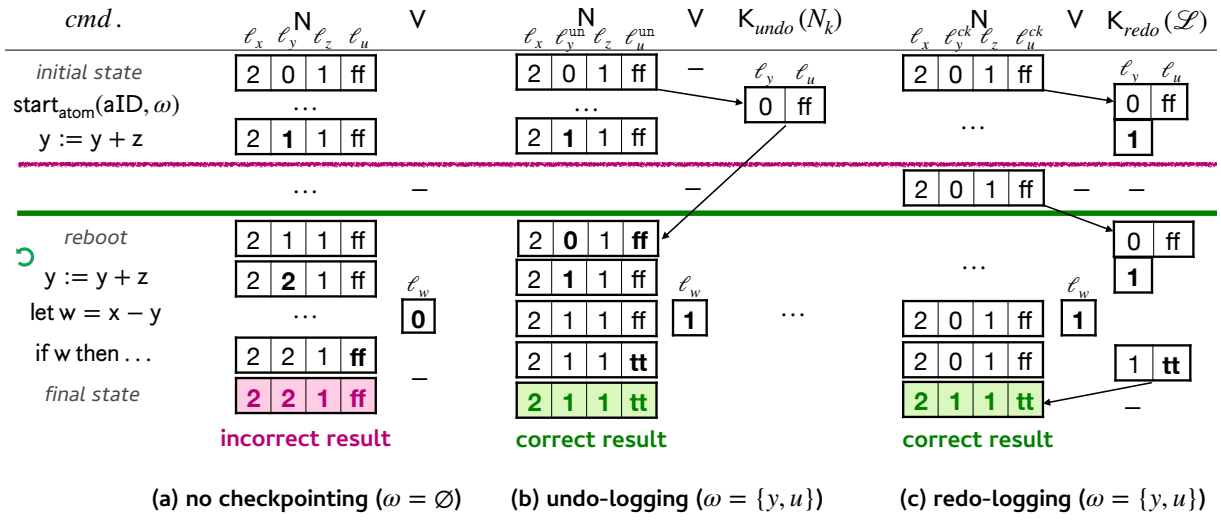

Figure 2.5: Possible atomic region executions: an incorrect execution (a) and correct executions using undo-logging (b) and redo-logging (c)

We illustrate a potential memory consistency bug for the example atomic region in Fig. 2.5 (a). Suppose that the initial nonvolatile memory state is $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 0, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow$ ff and no variables are checkpointed. The assignment writes the sum of y and z, which is 1, to location ℓ_y. If power fails immediately after (and before the end of the atomic region), the program will re-execute from the beginning with the state $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 1, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow$ ff.

On re-execution, the assignment to y would read its own value from the failed prior execution, storing 2 in ℓ_y. This causes 0 to be stored in ℓ_w and the else branch of the conditional to execute, which writes ff to ℓ_u. Execution ends in an *incorrect* final state, i.e., one that does not match the expected final state of any continuously-powered execution of the same program, which is $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 1, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow \mathsf{tt}$. The results are incorrect because they depend on unstable values computed by prior failed executions.

To execute an atomic region correctly, the intermittent system must checkpoint a portion of nonvolatile state prior to its execution. Checkpointing sufficient state will allow the intermittent system to revert unstable values after each failed execution of an atomic region so that the program can re-execute as expected. Which variables to checkpoint in a given atomic region is denoted by a special variable set ω in the atomic region policy. As EHDs are highly resource constrained, the system should save state judiciously. Not only is checkpointing all of nonvolatile memory expensive – it is unnecessary. For example, variables that are read-only (e.g., x, z) in an atomic region do not change value, and therefore need not be checkpointed [70, 81, 109]. When to create and update checkpoints for these variables is decided by the underlying logging style of the intermittent system, of which there are numerous [13, 14, 77, 82, 119, 125].

Undo-logging

A popular logging style used among state-of-the-art intermittent systems [77, 121] is *undo-logging*. With undo-logging, an atomic region saves a snapshot of nonvolatile memory, the entirety of volatile memory, and the continuation command to a special checkpoint context for undo-logging prior to its first execution, denoted K_{undo}. For hybrid programs consisting of both JIT and atomic regions, K_{undo} extends the checkpoint context for JIT regions with checkpointed nonvolatile memory N_k to hold *stable* values that will be used to restore a portion of nonvolatile state in the event of a power failure. For systems that use global volatile memory, V_k is also populated with checkpoints at the beginning of the atomic region execution.

$$\mathsf{K}_{undo} := (\mathsf{N}_k, \mathsf{V}_k, c_k)$$

This context will remain until the program finishes executing, and will be used to revert nonvolatile updates and recover volatile values on reboot from each failure. We show the key rules for undo-logging in Fig. 2.6, and explain them together with the atomic region execution in Fig. 2.5 (b).

According to the policy described above, suppose the programmer checkpoints only variables y and u which are not read-only. To allow the atomic region to re-execute as expected, the intermittent system stashes a copy of variables declared in ω in N_k (e.g., ℓ_y and ℓ_u) and the atomic region command in c_k, as performed by the rule D-A-START-U. The locations in N that have checkpointed counterparts in K_{undo} are annotated with a tag un indicating that these locations hold in-progress, or *unstable*, values.

As before, the atomic region command executes, this time storing the value 1 in a fresh location ℓ_y^{un}. Power fails immediately after, wiping the volatile memory and the current command, as modeled by the rule D-A-PWR-FAIL-U.

When energy is restored, the device turns on and attempts to re-execute the atomic region. This time, the values of nonvolatile locations (e.g., $\ell_y^{\mathsf{un}} \hookrightarrow 1$ and $\ell_u^{\mathsf{un}} \hookrightarrow \mathsf{ff}$) are rolled back to

$$\frac{\mathsf{dom}(\mathsf{N}^c) = \omega \qquad \mathsf{K}_{undo} = (\mathsf{N}^c, \mathsf{V}, c; \mathsf{end}_{\mathsf{atom}}) \qquad \mathsf{N}' = \mathsf{N}^c_{\mathsf{un}}, \mathsf{N}^0 \qquad \mathsf{N}^c_{\mathsf{un}}\ fresh \qquad \gamma' = \gamma \cup \{x \hookrightarrow \ell_{\mathsf{un}} \mid x \in \omega\}}{\gamma \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}^c, \mathsf{N}^0 \mid \mathsf{V} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega); c; \mathsf{end}_{\mathsf{atom}} \Rightarrow \gamma' \mid \mathsf{aID} \mid \mathsf{K}_{undo} \mid \mathsf{N}' \mid \mathsf{V} \mid c; \mathsf{end}_{\mathsf{atom}}} \text{ (D-A-START-U)}$$

$$\frac{\mathsf{range}(\gamma_v) = \mathsf{dom}(\mathsf{V})}{\gamma_n, \gamma_v \mid \mathsf{aID} \mid \mathsf{K}_{undo} \mid \mathsf{N} \mid \mathsf{V} \mid c \Rightarrow \gamma_n \mid \mathsf{aID} \mid \mathsf{K}_{undo} \mid \mathsf{N} \mid \cdot \mid \mathsf{reboot}} \text{ (D-A-PWR-FAIL-U)}$$

$$\frac{\mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, c_k) \qquad \mathsf{N} = \mathsf{N}^1, \mathsf{N}^2_{\mathsf{un}}}{\gamma \mid \mathtt{Md} \mid \mathsf{K}_{undo} \mid \mathsf{N} \mid \cdot \mid \mathsf{reboot} \ \Rightarrow\ \gamma \mid \mathtt{Md} \mid \mathsf{K}_{undo} \mid \mathsf{N}^1, \mathsf{N}_k \mid \mathsf{V}_k \mid c_k} \text{ (D-A-REBOOT-U)}$$

$$\frac{\mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, c) \qquad \gamma_{\mathsf{un}} = \mathsf{dom}(\mathsf{N}_k) \cup \mathsf{dom}(\mathsf{V}_k)}{\gamma, \gamma_{\mathsf{un}} \mid \mathsf{aID} \mid \mathsf{K}_{undo} \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{end}_{\mathsf{atom}} \Rightarrow \gamma \mid \mathsf{jit} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip}} \text{ (D-A-END-U)}$$

Figure 2.6: Selected operational semantic rules for atomic regions with undo-logging

their checkpointed counterparts (e.g., $\ell_y \hookrightarrow 0$ and $\ell_u \hookrightarrow \mathsf{ff}$), thereby "undoing" the effects from the failed partial execution, and the continuation command c_0 is restored, as modeled by the rule D-A-REBOOT-U. As c_0 was the atomic region command and the nonvolatile checkpoints were created (and last updated) prior to its first execution, this operation has the effect of re-executing the atomic region from the beginning on its initial nonvolatile memory state.

When the atomic region finishes executing, the rule D-A-END-U applies to finalize the state. As the region will endure no more power failures, the rule resets the checkpoint context to empty, and the execution mode to jit to prepare for executing the next region (if any). As expected, the final state in the example is *correct*, as it matches that of a single, continuously-powered execution of the atomic region.

Redo-logging

Another implementation of atomic regions is *redo-logging*, which makes use of a checkpoint context K_{redo} consisting of a volatile log $\mathcal{L}$ that contains an unstable copy of checkpointed locations to be operated on throughout the region's execution.

$$\mathsf{K}_{redo} := (\mathcal{L}, \mathsf{V}_k, c_k)$$

Since volatile state is wiped on power failure, updates made by failed executions would be automatically forgotten, thus preventing partial updates by failed executions from affecting future re-executions. The volatile log is (re-)created on every (re-)execution of the program until the execution concludes successfully, at which point its results are committed back to the corresponding locations in nonvolatile memory.

We show key rules for redo-logging in Fig. 2.7. Using these rules, we sketch the key steps involved in executing the example atomic region with redo-logging in Fig. 2.5 (c). Suppose the programmer checkpoints the same variables as before, y and u. When an atomic region starts executing, a redo-logging system would initialize the checkpoint context with a volatile

log and the atomic region command $c_0; \mathsf{end}_{\mathsf{atom}}$, as in the rule D-A-START-R. To remember which locations are checkpointed, which will be needed to recreate the log on each re-execution and commit results back to nonvolatile memory after the final execution, the rule additionally tags checkpointed nonvolatile locations with an annotation ck. Then, the atomic region begins to execute, writing 0 to the volatile log ℓ_y^{ck}. Power fails immediately after, and volatile memory, volatile log, and the current command are all wiped, as modeled by the rule D-A-PWR-FAIL-R, allowing the updates made by this failed execution to be forgotten by the next.

When power is restored, as in the rule D-REBOOT-R, the runtime system reboots and recreates the volatile logs by copying values from the checkpointed locations tagged ck into the same log locations. When the program finishes executing successfully, values computed in $\mathcal{L}$ (e.g., $\ell_y^{\mathsf{ck}} \hookrightarrow 1$ and $\ell_u^{\mathsf{ck}} \hookrightarrow \mathsf{tt}$) are committed to their respective nonvolatile locations (e.g., ℓ_y and ℓ_u), as in D-A-END-R. As before, the results of the example execution are correct.

$$\frac{\begin{array}{c}\mathsf{N} = \mathsf{N}^1, \mathsf{N}^2 \qquad \mathsf{dom}(\mathsf{N}^1) = \omega \\ \mathsf{N}' = \mathsf{N}^1_{\mathsf{ck}}, \mathsf{N}^2 \qquad \mathsf{K}_{redo} = (\{\ell'_x \hookrightarrow \mathsf{N}^1(\ell_x) \mid x \in \mathsf{dom}(\mathsf{N}^1),\ \ell'_x\ \mathit{fresh}\}; \mathsf{V}; c; \mathsf{end}_{\mathsf{atom}}) \\ \gamma' = \gamma \cup \{x \mapsto \ell'_x \mid x \in \omega\}\end{array}}{\gamma \mid \mathsf{jit} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega); c; \mathsf{end}_{\mathsf{atom}} \Rightarrow \gamma' \mid \mathsf{aID} \mid \mathsf{K}_{redo} \mid \mathsf{N}' \mid \mathsf{V} \mid c; \mathsf{end}_{\mathsf{atom}}} \text{ (D-A-START-R)}$$

$$\frac{\mathsf{K}_{redo} = (\mathcal{L}, \mathsf{V}_k, c) \qquad \mathsf{K}'_{redo} = (\cdot, \mathsf{V}_k, c)}{\gamma \mid \mathsf{aID} \mid \mathsf{K}_{redo} \mid \mathsf{N} \mid \mathsf{V} \mid c \Rightarrow \gamma \mid \mathsf{aID} \mid \mathsf{K}'_{redo} \mid \mathsf{N} \mid \cdot \mid \mathsf{reboot}} \text{ (D-A-PWR-FAIL-R)}$$

$$\frac{\mathsf{N} = \mathsf{N}^1, \mathsf{N}^2_{\mathsf{ck}} \qquad \mathsf{K}_{redo} = (\cdot, \mathsf{V}_k, c) \qquad \mathsf{K}'_{redo} = (\mathsf{N}^2, \mathsf{V}_k, c)}{\gamma \mid \mathsf{aID} \mid \mathsf{K}_{redo} \mid \mathsf{N} \mid \cdot \mid \mathsf{reboot} \ \Rightarrow\ \gamma \mid \mathsf{aID} \mid \mathsf{K}'_{redo} \mid \mathsf{N} \mid \mathsf{V}_k \mid c} \text{ (D-REBOOT-R)}$$

$$\frac{\mathsf{K}_{redo} = (\mathcal{L}, \mathsf{V}, c) \qquad \mathsf{dom}(\mathcal{L}) = \mathsf{range}(\gamma^2)}{\gamma^1, \gamma^2 \mid \mathsf{aID} \mid \mathsf{K}_{redo} \mid \mathsf{N}^1, \mathsf{N}^2_{\mathsf{ck}} \mid \mathsf{V} \mid \mathsf{end}_{\mathsf{atom}} \Rightarrow \gamma^1 \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}^1, \mathcal{L} \mid \mathsf{V} \mid \mathsf{skip}} \text{ (D-A-END-R)}$$

Figure 2.7: Selected operational semantic rules for atomic regions with redo-logging

In general, supporting hybrid programs with JIT and atomic regions requires a checkpoint context that includes the command c_k, in addition to both V_k and either N_k or $\mathcal{L}$ depending on which logging style for atomic regions is assumed. All of these components are present in some form in provably correct runtime systems for intermittent computing, whether explicit [119] or implicit to other parts of the formulation [40, 44].

Conceptually, the operational semantics for programs running on continuous power is similar to those running intermittently, except that they will never encounter power failure. Therefore, in a continuously-powered execution, atomic regions do not checkpoint any variables, instead simply running the enclosed command, and JIT regions will never encounter a low power signal where checkpointing is otherwise triggered.

2.4 A Crash Course in Adjoint Logic

In this section, we review the key concepts behind adjoint logic that motivate the contributions of this thesis.

Adjoint logic [18, 64, 95, 100, 101, 107] provides a general framework for combining logics and languages based on simple principles. In adjoint logic, every proposition has an assigned *mode of truth* that represents a single base logic. These modes are arranged according to a preorder, and are combined uniformly into one instance of adjoint logic. For example, in LNL [18] linear and nonlinear propositions correspond to different modes L and N. For general modes m and k that form a preorder $k \geq m$, the key principle of adjoint logic states that a proof of A_m can only depend on a hypothesis B_k [101]. This property is called the *independence principle.* In LNL, since nonlinear propositions admit weakening and contraction while linear propositions do not, these modes form a preorder $\mathsf{N} > \mathsf{L}$. Hence, a nonlinear sequent should not rely on linear assumptions. This property is enforced by the independence principle.

Shifts and adjoint logic sequent calculus rules. To illustrate the sequent calculus rules of adjoint logic, we inspect a simpler example, truth and validity, whose relationship can also be modeled in these terms. To first build some intuition, we begin with a simple motivating example: I wish to prove that every time one of my hair ties goes missing, the culprit is always my cat Zena. I suspect this because last week, she indeed stole one from me. Though, just because Zena ran off with one of my hair ties previously, it is not the case that she is always the culprit.

In general, *truth* in one instance does not imply truth in all instances, or *validity*, though they can be used in proofs of each other under the right conditions. The relationship between truth and validity embodies the independence principle of adjoint logic. In the instance of truth and validity, a proof of validity may only depend on valid assumptions whereas a proof of truth can depend on combinations of true and valid assumptions. For example, in the case of the perpetually disappearing hair tie, the independence principle for adjoint logic finds Zena the cat not guilty: a proof that Zena the cat is always the culprit (validity) cannot rely solely on knowing that she was once the culprit (truth).

To embed propositions of truth into proofs of validity, and vice versa, adjoint logic defines shifts $\downarrow^{valid}_{true}$ (downshift) and $\uparrow^{valid}_{true}$ (upshift), where the former demotes a proposition of validity to truth, and the latter promotes a proposition of truth to validity. The adjoint modalities allow going back and forth between valid and true judgments.

We show a selection of sequent calculus rules for adjoint logic below, of which the first four show the back-and-forth behavior of these shifts. Like [95], we introduce two kinds of contexts for hypotheses: one for truth (denoted Γ) and one for validity (denoted Δ). We introduce two judgment categories. The first category is for deriving validity and corresponds to judgments of the form $\Delta \vdash A_{valid}$, meaning that a proof of A_{valid} can rely only on valid assumptions. The second category is for deriving truth and corresponds to judgments of the form $\Delta; \Gamma \vdash A_{true}$, meaning that a proof of A_{true} can rely on both true and valid assumptions.

$$\frac{\Delta; \cdot \vdash A_{true}}{\Delta \vdash \uparrow^{valid}_{true} A_{true}} \uparrow R \qquad \frac{\Delta; \Gamma, A_{true} \vdash B_{true}}{\Delta, \uparrow^{valid}_{true} A_{true}; \Gamma \vdash B_{true}} \uparrow L$$

$$\frac{\Delta \vdash A_{valid}}{\Delta; \Gamma \vdash \downarrow^{valid}_{true} A_{valid}} \downarrow R \qquad \frac{\Delta, A_{valid}; \Gamma \vdash B_{true}}{\Delta; \Gamma, \downarrow^{valid}_{true} A_{valid} \vdash B_{true}} \downarrow L$$

The sequent calculus rules for upshift and downshift formalize the intuitions above regarding how truth and validity should interact in a proof, and are read bottom-up. In particular, to prove that A is valid, the rule $\uparrow R$ says it suffices to prove A is true under a true judgment with an empty truth context. In $\downarrow R$, the rule says that to establish a proposition A is true, it suffices to prove that A is valid under a strictly valid context. The left rules allow moving assumptions between the truth and validity contexts Γ and Δ: the rule $\downarrow L$ moves an assumption from Γ to Δ by use of the modality $\downarrow^{valid}_{true}$, and vice versa for $\uparrow L$ by way of $\uparrow^{valid}_{true}$.

Additional sequent calculus rules. We also take the opportunity to introduce a particular selection of sequent calculus rules that will arise in discussions of key derived rules in this section and in later chapters.

$$\frac{\Delta; \Gamma \vdash B_{true}}{\Delta; \Gamma, A_{true} \vdash B_{true}}\ WL_1 \qquad \frac{\Delta; \Gamma \vdash B_{true}}{\Delta, A_{valid}; \Gamma \vdash B_{true}}\ WL_2 \qquad \frac{\Delta, A_{valid}, A_{valid}; \Gamma \vdash B_{true}}{\Delta, A_{valid}; \Gamma \vdash B_{true}}\ CL_1$$

$$\frac{\Delta; \Gamma \vdash A_{true} \qquad \Delta; \Gamma, A_{true} \vdash B_{true}}{\Delta; \Gamma \vdash B_{true}}\ cut_1 \qquad \frac{\Delta; \Gamma \vdash A_{true}}{\Delta; \Gamma \vdash (A \vee B)_{true}}\ \vee R_1 \qquad \frac{}{A_{valid} \vdash A_{valid}}\ id_1$$

Assuming that both contexts are structural, we have weakening and contraction rules that allow for discarding and duplicating assumptions. Weakening and contraction will be important for our reasoning about intermittent computation later on, which is nonlinear: variables may be used any number of times in a computation. The rules for left weakening, WL_1 and WL_2, respectively allow us to dispose of an assumption from the true and valid contexts that is not needed to prove B_{true}. The rule CL_1 for left contraction allows us to copy valid assumptions.

Lastly, we include additional sequent calculus rules: cut_1 for introducing an intermediate formula A_{true} in a proof of B_{true}, $\vee R_1$ for proving $(A \vee B)_{true}$ given a proof of A_{true}, and id_1 for assuming A_{valid} to prove itself.

Towards independence for intermittent computing. Truth and validity are specific examples of *modes of truth*, of which all propositions in adjoint logic have inherent. For truth and validity, the independence principle enforces that proofs of validity cannot rely on true assumptions.

In the following chapters, we will see other examples of adjoint logic, where stable and unstable computation in intermittent computing resemble validity and truth. Moreover, the key correctness criterion of intermittent computing bears close resemblance to the independence principle of adjoint logic [18, 101, 107]:

> *"Stable computation can rely only on stable data, while unstable computation can rely on both stable and unstable data."*

Chapters 3-6 develop a type system based on sequent calculus rules for adjoint logic for shifting between modes stable (s) and unstable (u), where $s \geq u$. Like the example, we make use of two categories of judgments corresponding to these modes. In our setting, *true* is interpreted as unstable (u), *valid* is interpreted as stable (s), and the truth and validity contexts are interpreted as unstable (Σ) and stable store typing contexts (Ω). Below, we show typing rule skeletons that are derived from the adjoint logic sequent calculus rules for truth and validity. The explanations

for rules $\uparrow R$, $\downarrow R$, $\uparrow L$, and $\downarrow L$ are similar to before. Combining a cut_1 rule with $\downarrow R$ and id_1, we introduce a derived rule $\uparrow L^*$ that allows us to copy an assumption from the stable context into the unstable context. This rule will be used to model the checkpoint and restore operations for redo-logging atomic regions, which copy stable values to unstable volatile logs.

$$\frac{\Omega; \cdot \vdash \_ : \tau^u}{\Omega \vdash \_ : \uparrow^s_u \tau^u} \uparrow R \qquad \frac{\Omega; \Sigma, \_ : A^u \vdash \_ : \tau^u}{\Omega, \_ : \uparrow^s_u A^u; \Sigma \vdash \_ : \tau^u} \uparrow L \qquad \frac{\Omega, \_ : \uparrow^s_u A^u; \Sigma, \_ : \downarrow^s_u \uparrow^s_u A^u \vdash \_ : \tau^u}{\Omega, \_ : \uparrow^s_u A^u; \Sigma \vdash \_ : \tau^u} \uparrow L^*$$

$$\frac{\Omega \vdash \_ : \tau^s}{\Omega; \Sigma \vdash \_ : \downarrow^s_u \tau^s} \downarrow R \qquad \frac{\Omega, \_ : \uparrow^s_u A^u; \Sigma \vdash \_ : \tau^u}{\Omega; \Sigma, \_ : \downarrow^s_u \uparrow^s_u A^u \vdash \_ : \tau^u} \downarrow L$$

Chapter 3

Modal Crash Types for Intermittent Computing

This chapter introduces the crash types calculus that uses adjoint logic to reason about intermittent computing in terms of its key operations: power off, restore, and commit. As we observed in Chapter 2, the correctness principle for intermittent computing can be represented by the independence principle for adjoint logic: stable computation must not rely on unstable values. This observation is most clearly explained with redo-logging. As sketched in the previous chapter, in redo-logging, locations in nonvolatile memory (e.g., ℓ_x, ℓ_y^{ck}, ℓ_z, and ℓ_u^{ck}) are all stable, meaning that their values are safe to impact future re-executions of the program. On the other hand, volatile logs (e.g., ℓ_y, ℓ_u) which are updated intermittently and disappear on power failure are unstable, as their values should not influence future re-executions of a program. In order to ensure the correct intermittent execution of a program, unstable values should not be written to stable locations until the program successfully finishes executing. This pattern is enforced by redo-logging, where volatile memory contains all unstable values for a given atomic region execution, which are automatically erased when a device shuts off due to power failure, and whose logs are restored with copies of stable values when the device reboots. Otherwise, if stable locations could depend on unstable results, an unstable value could persist across power failures and affect later re-executions of the program, causing it to compute incorrect results, thus violating the independence principle.

We begin by presenting a calculus for atomic regions, whose re-execution causes the most interesting behavior with respect to execution and correctness of computation. Section 3.1 formally motivates the key ideas behind the core crash type system. Sections 3.2 and 3.3 formally define our operational semantics and type system. Section 3.4 defines key correctness theorems and a novel logical relation to help us prove that correctness is upheld by all programs in the defined language. Section 3.5 defines our calculus for JIT regions, which uphold similar correctness guarantees by means of the same logical relation. Lastly, Section 3.6 presents key theorems of our calculus, including progress, preservation, the fundamental theorem of logical relation, and adequacy to establish correctness, and sketches the proofs of the latter two. Throughout the chapter, we refer to the program executions from the previous chapter as an example.

3.1 Key Ideas of Crash Types

This section presents the intuition behind crash types, beginning with stable and unstable memory types (Section 3.1.1), then how to construct crash types for an intermittent computation (Section 3.1.2), and lastly how to build typing rules from these types based on the independence principle of adjoint logic (Section 3.1.3).

3.1.1 Modal Store Types

An *unstable value* is an intermediate result of a partial execution towards a *stable value*. Unstable values are lost upon power failure, like the update to the volatile copy of y on the first failed execution. On the other hand, stable values persist, like those stored in nonvolatile memory (e.g., ℓ_x, ℓ_y^{ck}, ℓ_z, and ℓ_u^{ck}). Unstable values that are committed to nonvolatile memory become stable and thus persist, like the values of ℓ_y and ℓ_u committed from their volatile locations to their checkpointed locations in nonvolatile memory at the end of the atomic region. To reflect the behavior of a memory location in its type, we encode the adjoint modalities $\uparrow_u^s$ and $\downarrow_u^s$ into our types, such that $\uparrow_u^s \tau^u$ types a location storing a stable value of type τ and $\downarrow_u^s \tau^s$ types a location that stores an unstable value of type τ. These types are further annotated with a qualifier describing the access mode of a variable's location, *read-only* RD and *checkpointed* CK. Including this information at the type level will help the type system check for appropriate memory accesses.

For example, the read-only variables x and z are stored in nonvolatile memory, so they have types $x :\uparrow_u^s \mathsf{int@RD}$ and $z :\uparrow_u^s \mathsf{int@RD}$. The checkpointed variables y and u have type $y^{\mathsf{ck}} :\uparrow_u^s \mathsf{int@CK}$ and $u^{\mathsf{ck}} :\uparrow_u^s \mathsf{int@CK}$ in nonvolatile memory, while y's volatile copy has type $y :\downarrow_u^s \uparrow_u^s \mathsf{int@CK}$. The variable w in volatile memory also has unstable type $w :\downarrow_u^s \uparrow_u^s \mathsf{int@CK}$. We sketch the correct program execution from the previous chapter in Fig. 3.1 with this typing information, where *aPol* indicates that variables y and u are checkpointed, the typing contexts Ω and Σ type nonvolatile (N) and volatile (V) memory, respectively, and we write i for int and b for bool. In the figure and following formulation, the volatile log $\mathcal{L}$ from the previous chapter is lumped into volatile memory.

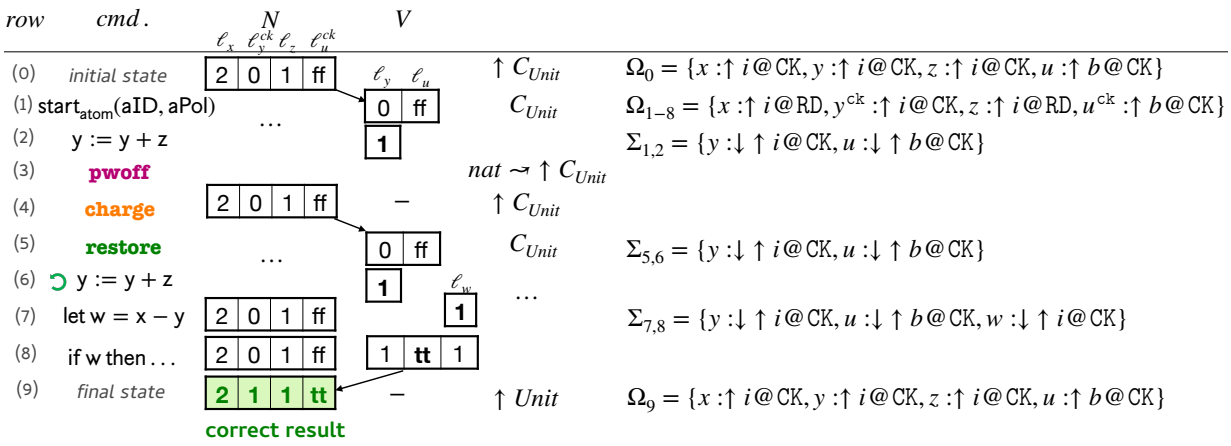

Figure 3.1: Atomic region redo-log execution with crash type annotations.

3.1.2 Crash Types

To capture the effects of intermittent execution in the type of expressions and commands, we introduce *crash types*, as the notion of stable and unstable values alone is insufficient. One might expect the expression $x - y$ to have the type $\downarrow_u^s\uparrow_u^s\texttt{int}$ as it is a (partial) execution ($\downarrow_u^s$) towards computing a stable ($\uparrow_u^s$) integer value. However, this type does not account for steps due to power failure: the crash itself, waiting for the device to charge, restoration, and re-execution. To reflect these runtime system steps at the type level, we assign the expression a type in the form of a disjunction $\boxed{?} \vee \downarrow_u^s\uparrow_u^s\texttt{int}$, where $\boxed{?}$ is a type for computations that handle power failures. This type means that the expression either fails its execution, or succeeds and evaluates to `int`. Next, we fill in $\boxed{?}$ for commands and expressions. $\boxed{?}$ is a recursive type because it handles re-execution.

Commands. The crash type for commands is $\mathsf{C}_{\texttt{unit}} = \downarrow_u^s(\texttt{nat} \rightsquigarrow \uparrow_u^s \mathsf{C}_{\texttt{unit}}) \vee \downarrow_u^s\uparrow_u^s\texttt{unit}$. The definition of the crash type represents each possible outcome of an intermittent execution: success or failure. The right disjunct states that if executing the command succeeds with no power failure, then it computes a stable value of type `unit`. This case also represents the last exeuction of an intermittent computation, at which point the computation succeeds and returns a stable value: after unstable state is lost due to failure ($\downarrow_u^s$), stable state is restored ($\uparrow_u^s$) and the command successfuly completes (`unit`).

Alternatively, the left disjunct states that on power failure, the computation continues as a function: on power failure the unstable state is lost ($\downarrow_u^s$), at which point the system must receive a (logical) energy input from the environment (`nat`) to charge the energy buffer before unstable state is restored and becomes a computation that yields a stable value ($\uparrow_u^s\mathsf{C}_{\texttt{unit}}$). This computation will execute after the restore, which differs for atomic and JIT modes. In an atomic region, the system re-executes the region from the beginning, and in a JIT region, the system continues with the same command that was interrupted by the failure.

Expressions. The definition of the crash type for expressions depends on the execution mode, which dictates whether a program would resume execution from the failed expression or re-execute from the beginning as a command. In an atomic region, the system restores an interrupted run of the expression to the original command enclosed in the region, so the type of an atomic mode expression is $\mathsf{C}_A^{\mathsf{atom}} = \downarrow_u^s(\texttt{nat} \rightsquigarrow \uparrow_u^s \mathsf{C}_{\texttt{unit}}) \vee \downarrow_u^s\uparrow_u^s A$, where the left disjunct is the same as that of a command. On the other hand, an interrupted run of an expression in JIT mode will be restored to the expression itself. Hence, the type of a JIT mode expression is $\mathsf{C}_A^{\mathsf{jit}} = \downarrow_u^s(\texttt{nat} \rightsquigarrow \uparrow_u^s\mathsf{C}_A^{\mathsf{jit}}) \vee \downarrow_u^s\uparrow_u^s A$, where the left disjunct states that after power failure and reception of the energy input, the computation again yields a stable value of a JIT mode expression type.

To type a program, we develop a type system for crash types. In the next section, we show the structure of these typing rules and explain their relation to adjoint logic.

3.1.3 Typing Intermittent Execution Based on Adjoint Logic

We explain each operation's interpretation in adjoint logic according to the rule skeletons from Chapter 2. We explain the switching between stable and unstable modes using the execution steps in Fig. 3.1, in terms of the key operations of intermittent computing: restore, power off, and commit.

Start and Restore ($\uparrow R$ and $\uparrow L^*$). A combination of the rules $\uparrow R$ and $\uparrow L^*$ corresponds to creating a volatile log from checkpointed nonvolatile locations when starting an atomic region, as in the step from row (0) to row (1) to create volatile locations for y and u. The row (0) type $\uparrow^s_u \mathsf{C}_{\mathtt{unit}}$ and typing context Ω corresponds to the conclusion of a $\uparrow R$ rule: $\Omega \vdash \_ : \uparrow^s_u \mathsf{C}_{\mathtt{unit}}$. An application of $\uparrow R$ from bottom to top drops the $\uparrow^s_u$ modality from the type of the program and starts an empty typing context for the volatile memory region: $\Omega; \cdot \vdash \_ : \mathsf{C}_{\mathtt{unit}}$. Next, one application of $\uparrow L^*$ copies the variable y of type $\uparrow^s_u \mathtt{int}$ to the volatile memory typing context Σ with the type $\downarrow^s_u \uparrow^s_u \mathtt{int}$. The same combination corresponds to creating a volatile log from a nonvolatile location when restarting the atomic region: the step from row (5) to row (6), where the variable y is copied to the volatile memory.

Power Off ($\downarrow R$). The $\downarrow R$ rule corresponds to a power failure, which erases the volatile memory typing context Σ. From row (2) to row (3) in Fig. 3.1, the system loses the volatile locations of y and u and closes off the volatile context. Row (2) corresponds to the conclusion of the rule, and row (3) corresponds to its premise. The type of the command in row (2) changes from $\mathsf{C}_{\mathtt{unit}}$ to $\downarrow^s_u(\mathtt{nat} \rightsquigarrow \uparrow^s_u \mathsf{C}_{\mathtt{unit}})$ (by a logical $\vee$-R rule as a crash is detected), and then to the type $(\mathtt{nat} \rightsquigarrow \uparrow^s_u \mathsf{C}_{\mathtt{unit}})$ in row (3).

Commit ($\downarrow L$, $\downarrow R$, WL_2). Finally, a $\downarrow L$ rule combined with weakening WL_2 and a $\downarrow R$ rule corresponds to the final commit of the volatile context: stepping from row (8) to row (9). There, the nonvolatile context drops the location of y with type $\uparrow^s_u \mathtt{int}$ by an application of WL_2 since y in the nonvolatile context maps to a location with an outdated value. Next, the up-to-date value stored in the volatile location of y is committed to the nonvolatile location of y and thus y in the Σ context is committed to the Ω context by a $\downarrow L$ rule. Then, a $\downarrow R$ rule drops the remaining volatile context, which contains w of type $\downarrow^s_u \uparrow^s_u \mathtt{int}$. The type of the command in row (9) becomes $\uparrow^s_u \mathtt{unit}$, signifying that the system detects a successful execution.

3.2 Syntax and Operational Semantics for Atomic Regions

In this section, we introduce the syntax and operational semantics of our base calculus for intermittent computing, focusing on atomic region execution.

Syntax. The syntactic constructs in our language are summarized in Fig. 3.2. Values, expressions, and commands are standard from the previous chapter. A program, for now, is a sequence of atomic regions. Since the assumed checkpointing policy checkpoints all variables that are not read-only, the atomic region policies *aPol* consist of sets of read-only and checkpointed variables, ρ and ω.

The variable-location mapping γ is defined as before, and nonvolatile memory (N) and volatile memory (V) map locations ℓ to values v that are each annotated with an access qualifier ($\mathtt{RD}$ or $\mathtt{CK}$). These access qualifiers are set at the beginning of an atomic region depending on whether the correponding variable is declared in ρ or ω. Nonvolatile memory locations that have a volatile log are tagged as $\ell_{\mathtt{ck}}$, while their volatile counterpart is identified as ℓ.

The runtime command $c_1 ;_W c_2$ is used for evaluating c_1 under the execution context W which will assist with enforcing proper variable scoping. The execution context W (written concretely as $\gamma \mid \mathsf{V}$) consists of the volatile memory state of in-scope variables for the command c_1 and a mapping from these variables to their memory locations (γ).

Commands and expressions

values	v	::=	$\mathsf{n} \mid \mathsf{tt} \mid \mathsf{ff} \mid x$
exprs	e	::=	$v \mid e \odot e \mid \oslash e$
cmds	c	::=	$\mathsf{skip} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \mid \mathsf{if}\, e\, \mathsf{then}\, c\, \mathsf{else}\, c \mid x := e \mid c; c \mid c;_W c$
atom. reg. pols.	$aPol$	::=	(ρ, ω)
progs	p	::=	$\mathsf{skip} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}; p$

Access qualifiers and memories

access qualifier	q	::=	$\mathtt{CK} \mid \mathtt{RD}$
nonvolatile mem	N	::=	$\cdot \mid \ell \mathbin{@} q \hookrightarrow v, \mathsf{N} \mid \ell_{\mathsf{ck}} \mathbin{@} \mathtt{CK} \hookrightarrow v, \mathsf{N}$
volatile mem	V	::=	$\cdot \mid \ell \mathbin{@} \mathtt{CK} \hookrightarrow v, \mathsf{V}$
var loc map	γ	::=	$\cdot \mid \gamma, x \mapsto \ell$

Instructions, statements, and configurations

continuations	κ	::=	$c \mid e$
statements	s	::=	$\kappa \mid i \mid p$
crash instrs	i	::=	$\downarrow\varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa) \mid \varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa) \mid \uparrow \kappa$
energy level	g	::=	$\cdot \mid n$
charge stream	χ	::=	$n :: \chi$
open config	C_o	::=	$(\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{V} \mid s) \mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid s)$
closed config	C_c	::=	$[\chi \rhd \varepsilon] \otimes C_o$
exec. mode	$\mathtt{Md}$	::=	$\mathsf{aID}(c)$

Figure 3.2: Summary of syntax

To model energy harvesting from the environment, we assume a unique external energy channel, ε, from which the system receives energy. Three crash instructions control the system in the event of a power failure. For these, we use shifts $\downarrow$ to denote power failure and $\uparrow$ to denote re-execution. The instruction $\downarrow\varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa)$ models the system that faces a power failure, where κ is the interrupted command or expression, and $b > 0$ is a guard to ensure that the bound incoming energy variable b is positive. The instruction $\varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa)$ models the system awaiting an energy input to be bound to b. The instruction $\uparrow\kappa$ models the system ready to restore memory and re-execute.

We write C_o to denote an *open* system configuration, consisting of the mapping γ, the mode of execution $\mathtt{Md}$ (for now, just aID for atomic regions), energy available for this execution g, memories, and the statement s to be executed. In the mode $\mathsf{aID}(c)$, the inclusion of c indicates from where an atomic region will re-execute. The energy level $(\cdot)$ models the state right after power failure. We assume that there are no global volatile memory locations, and hence V is neglected from some configurations (e.g., top-level program execution and power failure steps). We *close* an open configuration with $[\chi \rhd \varepsilon]$; we connect it via an external energy channel ε to an infinite charging stream χ of natural numbers, which models available energy the configuration harvests from the environment at each power failure for re-execution.

$$\frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip})}\ (\text{V-SKIP}) \qquad \frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{n})}\ (\text{V-NAT})$$

$$\frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{tt})}\ (\text{V-BOOL-T}) \qquad \frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{ff})}\ (\text{V-BOOL-F})$$

$$\frac{}{Val(\gamma \mid \texttt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid \kappa)}\ (\text{V-CRASH}) \qquad \frac{}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \texttt{Md} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa))}\ (\text{V-}\downarrow)$$

$$\frac{}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \texttt{Md} \mid \cdot \mid \mathsf{N} \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa))}\ (\text{V-}\#\ \mathsf{in}) \qquad \frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \uparrow\kappa)}\ (\text{V-}\uparrow)$$

$$\frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{skip})}\ (\text{V-P-DONE}) \qquad \frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip})}\ (\text{V-C-DONE})$$

Figure 3.3: Value configurations

3.2.1 Operational Semantics

In this section, we present the operational semantic rules for our base calculus with atomic regions. The top-level operational semantic rule for evaluating a sequence of atomic regions is of the form $C_c \Rightarrow_p C_c'$; the operational semantic rules for stepping commands and handling power failures inside an atomic region are of the form $C_c \Rightarrow C_c'$ and rules for evaluating commands and expressions under an open configuration where no power failure occurs are of the form $C_o \rightarrow C_o'$. We first define several auxiliary definitions before explaining each set of the operational semantic rules.

Value Configurations. We call a configuration that cannot take a step a *value configuration* (*value* for short), summarized in Fig. 3.3. An open configuration is a value if the statement s under evaluation is a constant or skip (the first four rules), the configuration of s has depleted all energy for this execution (rule V-CRASH), or s is a crash instruction (rules V-$\downarrow$,V-# in, and V-$\uparrow$). The latter two cases are values because they cannot take a step without interacting with the environment or perform operations on the volatile and novolatile memory specific to handling power failures. A closed configuration is a value only if the statement s is skip with some energy left ($n > 0$) (rule V-P-DONE for programs and rule V-C-DONE for commands).

Top-Level Atomic Region Execution (Closed Config). The D-P-CKPT rule in Fig. 3.4 executes the next atomic region in a program according to its policies $\mathsf{InitWorld}_d$ and $\mathsf{FinWorld}_d$. The nonvolatile locations N_0 and checkpointed context V_0 are initialized using the $\mathsf{InitWorld}_d$ function, which takes as inputs: the current N, declared read-only, must-first-write, and checkpointed variables ρ and ω, and their mapping to locations γ. The $\mathsf{InitWorld}_d$ function (a) changes the qualifier of locations in N that are declared as read-only in ρ from CK to RD, (b) checks that the rest of the locations in N are declared as checkpointed in ω and maintains the qualifier CK for these locations, (c) creates volatile logs by copying the locations of N that have qualifier CK, (d)

extends γ with variable-location mappings for the logs, and (e) marks the original version of the locations ℓ in N that still have qualifier $\mathtt{CK}$ as checkpointed (ℓ_{ck}) to indicate that they now store checkpointed values. This part corresponds to the step from row (0) to row (1) in Fig. 3.1. The next premise evaluates the closed configuration of c_0 until completion, using the rules in Fig. 3.5. This execution may undergo several power failures and corresponds to the steps from row (1) to row (7) in Fig. 3.1. Finally, the $\mathsf{FinWorld}_d$ function closes off atomic regions, finalizing the nonvolatile locations according to the context. $\mathsf{FinWorld}_d$ (a) copies the values from the volatile logs into N' to commit the changes to these variables to nonvolatile memory, (b) removes the subscript $\mathtt{ck}$ from the locations in N' to indicate that they now store up-to-date values and not checkpointed values, and (c) replaces the $\mathtt{RD}$ qualifiers of the locations in N' with $\mathtt{CK}$ to reset the access mode since they are out of the atomic region. This corresponds to the step from row (7) to row (8) in Fig. 3.1.

$\boxed{C_c \Rightarrow_p C_c'}$

$$\frac{\begin{array}{c} n > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{V}_0 \mid \gamma_0 \\ [\chi \rhd \varepsilon] \otimes \gamma_0 \mid \mathsf{aID} \mid n \mid \mathsf{N}_0 \mid \cdot \mid c_0 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p \;\Rightarrow_p\; [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p} \text{ (D-P-CKPT)}$$

Figure 3.4: Operational semantic rule for atomic regions

The policies are formally defined below.

Definition 1 $\mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{V}_0 \mid \gamma_0$ *where* $aPol = (\rho, \omega)$ *iff*

- $\mathsf{N} = \mathsf{N}_1, \mathsf{N}_2$,
- $\mathsf{dom}(\mathsf{N}_1) = \mathsf{range}(\gamma \restriction \rho)$,
- $\mathsf{dom}(\mathsf{N}_2) = \mathsf{range}(\gamma \restriction \omega)$,
- $\mathsf{V}_0, \gamma_v = \mathtt{unzip}(\{(\ell^*@\mathtt{CK} \hookrightarrow v,\ x \mapsto \ell^*) \mid \ell@\mathtt{CK} \hookrightarrow v \in \mathsf{N}_2,\ \ell^* \textit{ fresh}, x \in \omega \textit{ s.t. } \gamma(x) = \ell\})$,
- $\mathsf{N}_0 = \{\ell@\mathtt{RD} \hookrightarrow v \mid \mathsf{N}_1(\ell) = v\} \cup \{\ell_{\mathsf{ck}}@\mathtt{CK} \hookrightarrow v \mid \mathsf{N}_2(\ell) = v\}$,
- $\gamma_0 = \{x_{\mathtt{ck}} \mapsto \ell \mid \ell = \gamma(x)_{\mathtt{ck}},\ x \in \omega\} \cup \{x \mapsto \ell \mid \gamma(x)_{\mathtt{ck}},\ x \notin \omega\} \cup \gamma_v$

Definition 2 $\mathsf{FinWorld}_d(\mathsf{N}; \mathsf{V}) = \mathsf{N}_f$ *iff*

- $\mathsf{N} = \mathsf{N}_{1\,\mathtt{ck}}, \mathsf{N}_2$,
- $\mathsf{dom}(\mathsf{N}_1) = \mathsf{dom}(\mathsf{V})$,
- $\mathsf{N}_f = \mathsf{V} \cup \{\ell@\mathtt{CK} \hookrightarrow v \mid \ell@\mathtt{RD} \hookrightarrow v \in \mathsf{N}_2\}$

In Definition 1, we write $\gamma \restriction t$ to denote the subset of γ whose domain includes all the variables in the set t. When constructing the volatile logs V_0, we build the corresponding variable-location mappings γ_v at the same time which will be split into their own structures via a helper function $\mathtt{unzip}$. The mappings of checkpointed variables are tagged with a subscript $\mathtt{ck}$. This will enforce that the variable-location mapping associates checkpointed variables with their new logs in V_0, and that these logs will be accessed by the execution when their untagged variables are mentioned.

Command Execution (Closed Config). We summarize rules for a closed configuration in Fig. 3.5. Rule D-STEP steps the closed command configuration using the Fig. 3.6 rules. When the

energy for this execution is depleted (i.e., $n = 0$), the D-Crash rule applies, stepping the current expression or command (denoted κ) to the crash instruction $\downarrow\varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa)$. Upon a crash, D-S-aID applies and drops all volatile memory locations. The D-charge rule steps the execution when a natural number $n > 0$ is received from the energy channel. The number n represents the energy available for the re-execution. Finally, the program is restored via D-restore-aID which copies checkpointed locations into volatile memory to prepare for re-execution. D-restore-aID maintains the checkpointed locations $\mathsf{N}''_{\mathsf{CK}}$ and starts re-executing the original command c_0 in the atomic region.

$\boxed{C_c \Rightarrow C'_c}$

$$\frac{\gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, c \rightarrow \ \gamma \,|\, \mathtt{Md} \,|\, n' \,|\, \mathsf{N}' \,|\, \mathsf{V}' \,|\, c'}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, c \Rightarrow \ [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n' \,|\, \mathsf{N}' \,|\, \mathsf{V}' \,|\, c'} \ \text{(D-step)}$$

$$\frac{}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, 0 \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \downarrow \varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa)} \ \text{(D-Crash)}$$

$$\frac{\gamma' \subseteq \gamma \qquad \mathit{range}(\gamma') = \mathit{dom}(\mathsf{NV})}{\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0) \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \downarrow\varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa) \\ \Rightarrow \ [\chi \rhd \varepsilon] \otimes \gamma' \,|\, \mathsf{aID}(c_0) \,|\, \cdot \,|\, \mathsf{N} \,|\, \varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa)\end{array}} \ \text{(D-S-aID)}$$

$$\frac{}{[n :: \chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, \cdot \,|\, \mathsf{N} \,|\, \varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa) \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{N} \,|\, \uparrow\kappa} \ \text{(D-charge)}$$

$$\frac{\mathsf{N} = \mathsf{N}'_{\mathsf{RD}}, \mathsf{N}''_{\mathsf{ck}}}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N} \mid \uparrow\kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N} \mid \mathsf{N}'' \mid c_0} \ \text{(D-restore-aID)}$$

Figure 3.5: Operational semantics for commands and crash instructions under a closed configuration

Command/Expression Execution (Open Config). The rules for executing commands and expressions in an open configuration are standard and used in combination with the rule D-step to step a closed configuration command. We present them in Figs. 3.6 and 3.7. Both sets of rules use a small-step operational semantics, where each step decrements the energy level by one.

The rule D-NV-Read looks up the value v stored in a location ℓ corresponding to variable x. The rule D-Assign-NV applies when a nonvolatile memory location ℓ is written to via assignment statements. If the qualifier q is not read-only, the system replaces the value v' already in the location ℓ with the assigned value e.

The sequencing rules D-seq, D-seq-step, and D-seq-v use a runtime construct W that handles the scoping of volatile locations. The rule D-seq steps $c_1; c_2$ to the scoped command $c_1;_{\gamma|\mathsf{V}} c_2$ which initializes the world with a mapping γ and volatile memory state V. To ensure proper scoping, the idea is to remember the original volatile memory V before evaluating the first command c_1 of a sequence $c_1; c_2$, and to revert the volatile state back to V when the execution of the first command completes successfully. This removes any volatile locations allocated with let during the execution of command c_1. The rule D-seq-step preserves this world while evaluating the first command c_1. To simplify the proofs, we assume that D-seq-step always applies to the

$\boxed{C_o \to C_o'}$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e \,\mathsf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e' \,\mathsf{in}\, c} \text{ (D-Let-step)}$$

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\ \mathit{fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e_1 \,\mathsf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{CK} \hookrightarrow e_1 \mid c} \text{ (D-Let-v)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid p := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid p := e'} \text{ (D-Assign-step)}$$

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \gamma = \gamma', [x \to \ell] \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q \hookrightarrow e \mid \mathsf{skip}} \text{ (D-Assign-V)}$$

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \qquad q \neq \mathtt{RD} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip}} \text{ (D-Assign-N)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, e \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, e' \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2} \text{ (D-If)}$$

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, \mathsf{tt} \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \text{ (D-If-tt)}$$

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, \mathsf{ff} \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2} \text{ (D-If-ff)}$$

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma|\mathsf{V}} c_2} \text{ (D-seq)}$$

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{ (D-seq-step)}$$

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \upharpoonright \mathsf{dom}(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{skip};_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2} \text{ (D-seq-v)}$$

Figure 3.6: Operational semantics for command under an open configuration

$\boxed{C_o \to C'_o}$

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \ell \in \mathsf{dom}(\mathsf{V}) \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-V-READ)}$$

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \ell \in \mathsf{dom}(\mathsf{N}) \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-N-READ)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'_1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'_1 \odot e_2} \text{ (D-BINARY-1)}$$

$$\frac{n > 0 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'_2}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e'_2} \text{ (D-BINARY-2)}$$

$$\frac{n > 0 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2) \qquad v = e_1 \odot e_2}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-BINARY-V)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \oslash e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \oslash e'} \text{ (D-UNARY)}$$

$$\frac{n > 0 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad v = \oslash e}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \oslash e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-UNARY-V)}$$

Figure 3.7: Operational semantics for expression under an open configuration

leftmost sequence, meaning that c_1 is not of the form $c';_W c''$. Finally, when c_1 completely executes to skip, the D-SEQ-V steps to c_2 and only keeps those volatile locations that are declared in the original V', and their corresponding mapping γ'. The rule D-LET-V does not remove let-allocated location bindings as this is handled by other parts of the system. If there is a command that follows, this scoping is handled by the rules D-SEQ, D-SEQ-STEP, and D-SEQ-V as described above. If not, these locations will be dropped at the end of the atomic block by the FinWorld function in the D-SEQ-STEP rule as they are freshly introduced and do not have a checkpointed counterpart in nonvolatile memory.

Finally, we define the well-formedness conditions for configurations to constrain proper scoping of memory locations with respect to command sequencing as follows.

Definition 3 (Well-formedness for configurations) *A configuration* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed iff when* $c = c_1;_{W_1} \cdots ;_{W_{n-1}} c_n;_{W_n} c_{n+1}$ *where* $n > 1$*, all of the following hold*

- $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}_j) \subseteq dom(\mathsf{V}_i) \subseteq dom(\mathsf{V})$
- $\gamma_j \subseteq \gamma_i \subseteq \gamma$

where $1 \leq i < j \leq n$ *and* $W_k = \gamma_k \mid \mathsf{V}_k$ *for all* $k \in [1, n]$.

Definition 26 constrains configurations whose command is of the form $c_1;_{W_1} \cdots ;_{W_{n-1}} c_n;_{W_n} c_{n+1}$, where W_i records volatile memory V_i and mappings γ_i that are in scope for c_i. During execution, volatile memory V grows monotonically except when stepping with D-SEQ-V which discards out-of-scope variables, and the mapping γ grows and shrinks accordingly. For this reason, the locations of V_i (which was stashed in W_i prior to stepping to this configuration) should always be a subset of the locations of V which contain in scope memory locations for the whole command, and hence $dom(\mathsf{V}_i) \subseteq dom(\mathsf{V})$. Similarly, $dom(\mathsf{V}_j) \subseteq dom(\mathsf{V}_i)$ for $i < j$ because D-SEQ-STEP always applies to the leftmost sequence in a command and first executes the left part of the sequence. As a result, the locations in V_i include newly allocated locations while executing the left part of the sequence and the domain of V_j is a subset of the domain of V_i. The checkpointed locations in nonvolatile memory N_{ck} always have corresponding locations in volatile memory. The condition $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}_j)$ asserts that the checkpointed memory locations are never scoped out of volatile memory. Lastly, the condition $\gamma_j \subseteq \gamma_i \subseteq \gamma$ follows because the mappings γ, γ_i, and γ_j grow and shrink with their respective memories V, V_i, and V_j. This well-formedness definition is used to establish type preservation in Section 3.6.

3.3 Crash Type System for Atomic Regions

We define the types and typing rules of the crash type system for atomic regions. Our types are summarized in Fig. 3.8. The two modalities stratify types into the varieties stable (τ^s) and unstable (τ^u). The base store types `int` and `bool` are considered unstable, whereas $\uparrow^s_u\mathtt{int}$ and $\uparrow^s_u\mathtt{bool}$ are considered stable. A type variable $\mathsf{C}^{\mathtt{Md}}_T$ denotes a type in the set $\{\mathsf{C}_{\mathtt{unit}}, \mathsf{C}^{\mathsf{atom}}_A\}$, and implements the recursive nature of crash types. For now, we consider `Md` to be atomic mode (atom). Below, we repeat the crash types that were introduced in Section 3.1.

We include the connectives $\vee$ and $\rightsquigarrow$ solely for the purpose of defining crash types; they are not used elsewhere. Defining crash types using these connectives will allow us to define a logical relation based on the intended meaning of its index type (to be discussed). Some well-

formed types, e.g., $\texttt{nat} \rightsquigarrow \texttt{nat} \rightsquigarrow \uparrow^s_u \texttt{unit}$, have no inhabitants in our system, i.e., no well-typed configuration is of these types.

Store Typing. We define stable and unstable typing contexts for typing variables also in Fig. 3.8. A stable store typing context Ω assigns stable types to variables with locations in nonvolatile memory, i.e. all variables in Ω have a type of the form $\uparrow^s_u A$. The unstable store typing context Σ assigns unstable types to variables with locations in volatile memory, i.e., variables in Σ are of the type $\downarrow^s_u \uparrow^s_u A$. We write $x_{\texttt{ck}}$ to refer to a location in nonvolatile memory that has been checkpointed. In atomic mode with redo-logging, $x_{\texttt{ck}}$ has an active volatile log that is typed by Σ. Each variable's type is annotated with an access qualifier representing its prescribed access mode, RD or CK. The access qualifer will be used when type checking commands to enforce that variable accesses are obeyed (e.g., that read-only variables are not written).

To enforce the runtime stores N and V are well-typed with respect to typing contexts Ω and Σ, we present the well-formedness definitions for stores in Fig. 3.9. The rule V-Loc checks a volatile location's type with respect to Σ and γ. The rule NV-Loc-aID-1 checks non-checkpointed locations in nonvolatile memory by checking that the value stored in the location ℓ allocated for x has the same type as x. The rule NV-Loc-aID-2 applies when x has a checkpointed value stored in N and an up-to-date value stored in the log in V. In this case, the rule ensures that these two values have the same type. Allowing for volatile memory V and the volatile typing context Σ to be $\cdot$, these rules also define well-formedness of N with respect to Ω which arises when configurations only contain nonvolatile memory.

Typing Judgments. Table 3.1 summarizes all the typing judgments, which are also stratified into two varieties, those that derive a stable type (J_s) and those that derive an unstable type (J_u).

Since programs begin their execution from a stable state, they are typed under a stable typing judgment with only a stable typing context (e.g., the state in row (0), before the atomic region begins executing). On the other hand, commands, whose unfinished executions are susceptible to power failure, are typed under an unstable typing judgment, which depend on both nonvolatile and volatile store typing contexts. Unstable judgments J_u have both nonvolatile and volatile store typing contexts which type the state during an unstable computation. Alternatively, J_s judgments include only the stable typing context because these type stable command state. The structure

store types	A	::=	$\texttt{int} \mid \texttt{bool}$
basic types	T	::=	$\texttt{unit} \mid A$
unstable types	τ^u	::=	$T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid \mathsf{C}^{\texttt{Md}}_T$
stable types	τ^s	::=	$\texttt{nat} \rightsquigarrow \tau^s \mid \uparrow \tau^u$
type variables	$\mathsf{C}^{\texttt{Md}}_T$	::=	$\mathsf{C}_{\texttt{unit}} \mid \mathsf{C}^{\texttt{Md}}_A$
unstable store typing context	Σ	::=	$\cdot \mid x : \downarrow^s_u \uparrow^s_u A @ \texttt{CK}, \Sigma$
stable store typing context	Ω	:=	$\cdot \mid x : \uparrow^s_u A @ q,\ \Omega \mid x_{\texttt{ck}} : \uparrow^s_u A @ \texttt{CK}, \Omega$
command crash type	$\mathsf{C}_{\texttt{unit}}$	=	$\downarrow(\texttt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\texttt{unit}}) \vee \downarrow\uparrow\texttt{unit}$
expression crash type	$\mathsf{C}^{\texttt{Md}}_A$	=	$\downarrow(\texttt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\texttt{unit}}) \vee \downarrow\uparrow A$

Figure 3.8: Types and typing contexts

$\boxed{\vdash_\gamma^{\mathsf{aID}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma}$

$$\frac{}{\vdash_\gamma^{\mathsf{aID}} \cdot \mid \cdot : \cdot \mid \cdot} \text{(Empty)}$$

$$\frac{\mathsf{V} = \mathsf{V}', \ell \mathbin{@} q \hookrightarrow v \qquad q = \mathsf{Ck} \qquad \vdash_{\gamma'}^{\mathsf{aID}} \mathsf{N} \mid \mathsf{V}' : \Omega \mid \Sigma \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{aID} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} v : \uparrow A}{\vdash_\gamma^{\mathsf{aID}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma, (x{:}\downarrow\uparrow A @ q)} \text{(V-loc)}$$

$$\frac{\mathsf{N} = \mathsf{N}', \ell \mathbin{@} q \hookrightarrow v \qquad q \neq \mathsf{Ck} \qquad \vdash_{\gamma'}^{\mathsf{aID}} \mathsf{N}' \mid \mathsf{V} : \Omega \mid \Sigma \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} v : \uparrow A}{\vdash_\gamma^{\mathsf{aID}} \mathsf{N} \mid \mathsf{V} : \Omega, (x{:}\uparrow A @ q) \mid \Sigma} \text{(N-loc-aID-1)}$$

$$\frac{\vdash_{\gamma'}^{\mathsf{aID}} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma \qquad \mathsf{N} = \mathsf{N}', \ell_{\mathsf{ck}} \mathbin{@} q \hookrightarrow v \qquad \mathsf{V} = \mathsf{V}', \ell \mathbin{@} q \hookrightarrow v' \qquad q = \mathsf{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{aID} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} v : \uparrow A \qquad \mathsf{aID} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} v' : \uparrow A}{\vdash_\gamma^{\mathsf{aID}} \mathsf{N} \mid \mathsf{V} : \Omega, (x_{\mathsf{ck}}{:}\uparrow A @ q) \mid \Sigma, (x{:}\downarrow\uparrow A @ q)} \text{(N-loc-aID-2)}$$

Figure 3.9: Well-formedness of $\mathsf{N} \mid \mathsf{V}$ w.r.t. $\Omega \mid \Sigma$

of our unstable and stable typing judgments help guarantee that the independence principle for intermittent execution is upheld: values in volatile memory should not influence those in the nonvolatile memory unless a stable state is reached.

The unstable and stable typing judgments are parameterized over the execution mode `Md` of the expression or command to be typed. The judgment also tracks a variable b corresponding to the current energy level of this execution. b ranges over natural numbers (`nat`) and is constrained by a relation $\mathcal{R} \in \{\geq, >\}$ or is set to zero. When the constraint is $b \geq 0$, b is effectively unconstrained. The constraint on b determines whether or not a command can evaluate a value without power failure.

The command judgments use a signature context `Sig`, which is populated at different points of the type checking depending on the mode. For atomic regions, `Sig` stores the typing judgments for the original command of the atomic region such that when typing commands restored from a power failure, the signature is used to check that the restored command typing matches the one stored in the signature without needing to derive it again. This is similar to typing recursive functions, and the signature makes the typing derivations finitary and inductive. There are three judgments for command typing. The first judgment is used when the command has not yet successfully finished executing; its next step, depending on its constraint $\mathcal{R}$, may or may not crash. The second judgment types commands with type $\downarrow\uparrow\mathtt{unit}$. In this judgment, b no longer needs to be constrained because this judgment only types commands that have succeeded completing the execution. This judgment invokes the third judgment and uses it as a sub-derivation-tree to type the configuration after the volatile log is committed.

The expressions that are being written to are only of the simple form x. As no execution is required to evaluate a variable x, we consider the operation atomic so its judgment is crash-free, and hence no constraint is required on b. The labels `RD` and `WT` indicate read and write accesses

cmd.	(J_u)	$\texttt{Md} \mid b \mathcal{R} 0 : \texttt{nat} \mid \Omega; \Sigma$	$\vdash_{\texttt{Sig}} c$	$: \mathsf{C}_{\texttt{unit}}$	c could crash
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma$	$\vdash_{\texttt{Sig}} \mathsf{skip}$	$: \downarrow\uparrow\texttt{unit}$	c will not crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega$	$\vdash_{\texttt{Sig}} \mathsf{skip}$	$: \uparrow\texttt{unit}$	after commit
expr.	(J_u)	$\texttt{Md} \mid b \mathcal{R} 0 : \texttt{nat} \mid \Omega; \Sigma$	$\vdash_{\texttt{RD;Sig}} e$	$: \mathsf{C}_A^{\texttt{Md}}$	e read, could crash
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma$	$\vdash_{\texttt{RD;Sig}} v$	$: \downarrow\uparrow A$	e read no crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega$	$\vdash_{\texttt{RD}} v$	$: \uparrow A$	e read, commit
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma$	$\vdash_{\texttt{WT}} x$	$: \downarrow\uparrow A$	write on x, no crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega$	$\vdash_{\texttt{WT}} x$	$: \uparrow A$	write on x, commit
prog.	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega$	$\vdash p$	$: \uparrow \mathsf{C}_{\texttt{unit}}$	before execution
crash	(J_u)	$\texttt{Md} \mid b = 0 : \texttt{nat} \mid \Omega; \Sigma$	$\vdash_{\texttt{Sig}} \kappa$	$: \mathsf{C}_T^{\texttt{Md}}$	about to crash
	(J_u)	$\texttt{Md} \mid \cdot \mid \Omega; \Sigma$	$\vdash_{\texttt{Sig}} \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa)$	$: \downarrow(\texttt{nat} \rightsquigarrow \uparrow \mathsf{C}_T^{\texttt{Md}})$	crash state
	(J_s)	$\texttt{Md} \mid \cdot \mid \Omega$	$\vdash_{\texttt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa)$	$: \texttt{nat} \rightsquigarrow \uparrow \mathsf{C}_T^{\texttt{Md}}$	waiting for energy
	(J_s)	$\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega$	$\vdash_{\texttt{Sig}} \uparrow\kappa$	$: \uparrow \mathsf{C}_T^{\texttt{Md}}$	before re-execution

Table 3.1: Summary of typing judgments

in a program. The static type checking compares these labels to variable access qualifiers in the typing contexts to enforce proper memory accesses of variables.

For program typing, we only have one judgment that refers to the type of the program before the execution of its next block starts.

The rest of the judgments type states after a crash. The first crash typing judgment has the constraint $b = 0$, which corresponds to the power failure condition. It invokes the second judgment, which types a state right after a crash. The third judgment types the state awaiting energy to continue re-execution, and the final judgment types the state that is ready for restoration and re-execution.

3.3.1 Typing Rules

The structure of our unstable and stable typing judgments help guarantee that the independence principle for intermittent execution is upheld: values in volatile memory should not influence those in nonvolatile memory unless a stable state is reached.

Program Typing. In particular, the rules for typing atomic regions encode the behavior of the $\uparrow^s_u R$ rule sketched in the previous section into the crash type system. We show a typing rule below.

$$\frac{\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; aPol) \qquad \texttt{Sig} = \{\mathsf{aID}(c_0) \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma_0 \vdash c_0 : \mathsf{C}_{\texttt{unit}}\} \quad \mathsf{aID}(c_0) \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma_0 \vdash_{\texttt{Sig}} c_0 : \mathsf{C}_{\texttt{unit}} \qquad b : \texttt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\texttt{unit}}}{b : \texttt{nat} \mid \Omega \vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\texttt{unit}}} \text{ (T-P-Ckpt)}$$

To type an atomic region, the rule T-P-Ckpt applies. The rule types the enclosed command c_0 under the mode $\mathsf{aID}(c_0)$ (e.g., $y := y + z$; *let* $w = x - y$ *in* ...) and then types the rest of the program p. The first premise sets up the initial typing contexts for nonvolatile and volatile memories, as illustrated between rows (0) and (1) in Fig. 3.1. The partial function $\mathsf{InitWorld}_t$ initializes the volatile memory typing context by creating a log of variables in Ω that are checkpointed (e.g., $y{:}\downarrow^s_u\uparrow^s_u i@\texttt{CK}$ and $u{:}\downarrow^s_u\uparrow^s_u b@\texttt{CK}$, from their counterparts $y^{ck}{:}\uparrow^s_u i@\texttt{CK}$ and $u^{ck}{:}\uparrow^s_u b@\texttt{CK}$ in the sta-

ble typing context). In particular, for all variables defined in ω the function assigns an access qualifier CK, and RD otherwise (e.g., $y^{ck}{:}\uparrow^{s}_{u} i@\texttt{CK}$ versus $x^{ck}{:}\uparrow^{s}_{u} i@\texttt{RD}$).

We write $\Omega \restriction t$ to denote the subset of Ω whose domain includes all the locations in the set t. We define $\Omega_{\texttt{ck}}$ to be the same mapping as Ω except that a subscript ck is added to all the variables in the domain of $\Omega_{\texttt{ck}}$. This is used to generate a typing context for checkpointed locations. We define $\downarrow\Omega$ to be the context resulting from adding a $\downarrow$ in front of each type that variables in the domain of Ω are mapped to. This definition is used to generate a typing context for the volatile log from their checkpointed locations. They are formally defined below:

$$\Omega_{\texttt{ck}} = \{x_{\texttt{ck}} : \uparrow A@q \mid x : \uparrow A@q \in \Omega\} \qquad \downarrow\Omega = \{x : \downarrow\uparrow A@q \mid x : \uparrow A@q \in \Omega\}.$$

Next, we define the $\mathsf{InitWorld}_t$ function as follows.

Definition 4 $\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; aPol)$ *where* $aPol = (\rho, \omega)$ *iff*

- $\mathsf{dom}(\Omega) = \rho \cup \omega$,
- $\rho \cap \omega = \emptyset$,
- $\Omega_0 = \{x : \uparrow A@\texttt{RD} \mid x : \uparrow A@q \in \Omega^r\} \cup \Omega^c_{\texttt{ck}}$ *and*
- $\Sigma_0 = \downarrow\Omega^c$

where $\Omega = \Omega^r, \Omega^c$ *and* $\Omega^r = \Omega\restriction\rho$, *and* $\Omega^c = \Omega\restriction\omega$.

If the sets of read-only and checkpointed variables, ρ and ω, do not comprise the domain of Ω or are not disjoint, then the function $\mathsf{InitWorld}_t$ is not defined and type-checking fails.

After constructing the new typing contexts using $\mathsf{InitWorld}_t$, the rule T-P-Ckpt uses the signature Sig to remember that the atomic region's original command c_0 is typed under the initial static memory context $\Omega_0; \Sigma_0$. The region is then typed relative to the signature.

Command Typing. The typing rules for commands are presented in Fig. 3.10. The T-Skip rule types the command skip at the stable type $\uparrow\texttt{unit}$. At this point, the command execution is complete and the initial nonvolatile context Ω should match the final nonvolatile context. Rule T-$\vee$-Succ applies when the command successfully completes its execution and still has at least one unit of energy available ($b > 0$) to conclude the execution by committing. In this case, we close off the constraint on the energy level variable and continue typing the command against the type $\downarrow\uparrow$ unit. Rule T-C-Shift is invoked by T-$\vee$-Succ and updates the memory typing contexts by removing checkpointed locations in Ω as now they are not needed, and making locations in Σ stable as now they are committed. We write $\Omega = \Omega', \Omega''_{\texttt{ck}}$ to uniquely partition the stable typing context into typings for checkpointed ($\Omega''_{\texttt{ck}}$) and non-checkpointed variables (Ω'), and $\Sigma = \downarrow\Sigma'$ to produce stable typings for all variables in Σ'. Therefore, to type skip under a stable type, we type it under the stable context Ω', Σ', whose typings correspond to non-checkpointed nonvolatile variables and volatile logs whose values now stabilize. This corresponds to the last step of Fig. 3.1.

The rules T-Let and T-Assign, are mostly standard except that we consider crashes. For example, in typing the command $x := e$, the first premise of T-Assign considers the type of expression e to be the crash type $\mathsf{C}^{\texttt{Md}}_A$, since the evaluation of e could crash. The constraint on the energy levels for this premise is $b \geq 0$, as one energy unit is consumed to deconstruct the assignment command. The second premise types the location x to be of type $\downarrow\uparrow A$ because the assignment only occurs when e completes evaluation.

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow \mathtt{unit}} \text{ (T-Skip)}$$

$$\frac{\Sigma = \downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathtt{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow \mathtt{unit}}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow \uparrow \mathtt{unit}} \text{ (T-C-Shift)}$$

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow \uparrow \mathtt{unit}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \vee \downarrow \uparrow \mathtt{unit}} \text{ (T-}\vee\text{-Succ)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_A \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}\downarrow\uparrow A @ \mathtt{CK} \vdash_{\mathtt{Sig}} c : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{let}\, x = e_1 \,\mathsf{in}\, c : \tau} \text{ (T-Let)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}}} \text{ (T-Assign)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{if}\, e \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2 : \tau} \text{ (T-If)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau} \text{ (T-seq)}$$

$$\frac{\begin{array}{c}W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \\ \Sigma' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau} \text{ (T-seq-d)}$$

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau} \text{ (T-enough?)}$$

Figure 3.10: Command typing

In the rule T-If, the first premise checks that expression e has type $\mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}}$, and the second and third premises type the commands c_1 and c_2.

The rule T-seq types the command $c_1; c_2$ by first typing c_1 and then c_2. The rule T-seq-d applies to the runtime sequencing command where an extra context $W = \gamma \mid \mathsf{V}$ keeping track of the scoping of volatile locations is used. The premise $\Sigma' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma)$ trims the volatile typing context to remove let variables scoped within c_1 and match the scoped volatile memory V and mapping γ. The trimmed context Σ' is the original context for typing the second command c_2.

The rule T-enough? cases on $b \geq 0$ and considers two cases: $b = 0$ (the first premise) where the execution crashes and $b > 0$ (the second premise) where there is at least one unit of energy available to decompose a command construct, e.g. T-Let or T-Assign.

Expression Typing. Expression typing rules are very similar to those of the commands and they

$$\frac{\Omega(x) = \uparrow A @ q \qquad q \neq \mathtt{RD}}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{WT}} x : \uparrow A} \text{ (T-Loc-Write)}$$

$$\frac{\Sigma =\downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A}{\mathtt{Md} \mid \ b : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A} \text{ (T-W-Shift)}$$

$$\frac{\Omega(x) = \uparrow A @ q}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} x : \uparrow A} \text{ (T-Loc-Read)} \qquad \frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathsf{tt} :\uparrow \mathtt{bool}} \text{ (T-Bool-t)}$$

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathsf{ff} : \uparrow\mathtt{bool}} \text{ (T-Bool-f)} \qquad \frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathsf{n} : \uparrow\mathtt{int}} \text{ (T-Int)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \downarrow\uparrow A} \text{ (T-R-Shift)}$$

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A}{\mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau_1 \vee \downarrow\uparrow A} \text{ (T-}\vee\text{-Succ)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_{T}^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}} \qquad \odot : \uparrow T \times \uparrow T' \to \uparrow T''\end{array}}{\mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}_{T''}^{\mathtt{Md}}} \text{ (T-Binary)}$$

$$\frac{\mathtt{Md} \mid b \ = 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{Sig}} e : \tau \qquad \mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau} \text{ (T-enough?)}$$

Figure 3.11: Expression typing

are shown in Fig. 3.11. The T-Loc-Write and T-Loc-Read rules match the location variable x with an existing variable inside the context. T-Loc-Write checks a variable to be written to x by checking that it is not read-only. The T-W-Shift rule is similar to T-C-Shift but considers the variable x at type $\downarrow\uparrow A$ under an unstable typing judgment in the conclusion, and at $\uparrow A$ under a stable typing judgment in the premise.

Statement Typing. The typing rules for crash instructions are presented in Fig. 3.12. A crash is detected by the depleted energy level $b = 0$ in the T-$\vee$-crash rule. In the premise, the crash instruction $\downarrow\varepsilon \ \# \ \mathsf{in}(b > 0, \uparrow\kappa')$ is typed. The T-aID-stop rule simply drops the volatile locations in Σ. The T-Charge rule inputs a new energy level from the energy channel ε. The first premise shows that the energy channel is needed to provide a natural number greater than zero. Finally, the T-aID-Restore rule prepares for re-execution; volatile memory is restored from the checkpointed locations in Ω. The checkpointed location typings remain in Ω as we may need them if there is another power failure. Execution continues with the original command c_0 enclosed in the atomic region. Instead of retyping the restored judgments, we check if there are already typing derivations by matching them up with the saved judgment in the signature.

$$\frac{\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T'})}{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \kappa' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-}\vee\text{-Crash)}$$

$$\frac{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}})}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}})} \text{ (T-aID-stop)}$$

$$\frac{\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow \kappa' : \uparrow \mathsf{C}^{\mathtt{Md}}_{T}}{\mathtt{Md} \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow \kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})} \text{ (T-Charge)}$$

$$\frac{\Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow\kappa' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-aID-Restore)}$$

Figure 3.12: Crash, restore, and checkpoint typing

3.4 A Logical Relation for Crash Types

To prove the correctness of a sequential, intermittent program over a finite but abritrary number of crashes, we define a logical relation over crash types. The key idea is that in order to prove that an intermittent execution is correct, its final results must match those of a single, continuously-powered execution, which is obtained by matching certain steps from the intermittent execution. The logical relation provides the vehicle for this matching, maintaining a relation between configurations of the intermittent and continuously-powered executions at each step, which is eventually used to establish that the final memory states between the two executions are the same.

In this section, we define a logical relation for relating intermittent and continuously-powered executions of a single atomic region. We use the logical relation to introduce a semantic typing for programs consisting of atomic regions. Later, in Section 3.6, we show that statically well-typed programs are semantically well-typed and that our semantic typing ensures idempotency.

3.4.1 Semantic Typing via a Logical Relation

We define a binary logical relation for crash types that relates an intermittent execution with a continuously-powered execution for a single atomic region. Since crash types are recursive, we rely on step-indexing to ensure the well-foundedness of the definitions. The step index is the maximum number of executions that an observer would allow for the intermittent execution. Similar to prior work [8, 45, 122], our definition consists of a term relation $\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$ and a value relation $\mathcal{V}[\![\tau]\!]^m$, which relate an open configuration with at most m attempts for executing the region to an open configuration with continuous power. The value relation requires the intermittently executed configuration to be a value configuration.

The logical relation is constructed in such a way that two related configurations will result in the same nonvolatile memories upon completing evaluation of the command, which captures the idempotency requirements. The term and value logical relations are inductively defined over

$$\frac{\mathsf{N}_2 = \mathsf{N}'_{1\,\mathtt{ck}}, \mathsf{V}'' \qquad \mathsf{V}_1 = \mathsf{V}'_1, \mathsf{V}'' \qquad \mathsf{N}_1 = \mathsf{N}'_{1\,\mathtt{RD}}, \mathsf{N}''_{\mathtt{ck}} \qquad \mathsf{dom}(\mathsf{V}'') = \mathsf{dom}(\mathsf{N}'') \qquad \gamma' \subseteq \gamma \; s.t. \; range(\gamma') = \mathsf{dom}(\mathsf{N}_1)}{\mathtt{Commit}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{N}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{\mathtt{RD}}, \mathsf{N}''_{\mathtt{ck}} \qquad \mathsf{V}_2 = \mathsf{N}''}{\mathtt{Restore}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \kappa) = \mathsf{N}_2 \mid \mathsf{V}_2 \mid c} \qquad \frac{\gamma' \subseteq \gamma \; s.t. \; range(\gamma') = \mathsf{dom}(\mathsf{N}_1) \qquad \mathsf{V}_2 = \cdot}{\mathtt{PwOff}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2}$$

Figure 3.13: `Commit`, `Restore`, and `PwOff` policy definitions for atomic regions

a lexicographic induction on the index m and the structure of the types. Our logical relation is presented in Fig. 3.14.

Auxiliary Definitions. Before explaining details of our logical relation, we define notations and policies used in its definition.

We write C_o^{∞} to denote an open configuration with an infinite energy level: they are of the form $(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c)$. Such configuration is used to model a continuously-powered execution. We write $C_o \rightarrow^*_{irred} C'_o$ to mean that C_o evaluates to an irreducible configuration C'_o which cannot take any more steps. Since our semantics for commands is deterministic, for each configuration C_o there is exactly one such irreducible configuration. An irreducible configuration might not be a value but rather an ill-typed configuration that cannot take any more steps. Syntactic typing, by enabling a proof of progress and preservation, ensures that an irreducible configuration can only be a value configuration as defined in Fig. 3.3. However, our logical relation does not assume syntactic typing of configurations. Instead, by defining a value relation for each type, the logical relation ensures that an irreducible configuration reachable from a semantically well-typed program is indeed a value configuration.

To account for the power failure, recovery, and finalizing atomic regions' effects on memory, the logical relation relies on `PwOff`, `Restore`, and `Commit` policies, referred to as power failure, restore, and commit policies, respectively. We formally define these policies in Fig. 3.13. The definitions match the memory operations in the dynamic rules that deal with crash, restore, and re-execution (D-S-aID, D-restore-aID, and D-P-Ckpt) for atomic regions, though they can be generalized as we will discuss in Sections 3.5.

In Fig. 3.13, the commit policy is defined to capture the effects of finalizing the nonvolatile memory after a successful atomic region execution. The commit policy copies the values of the volatile logs into the nonvolatile memory and changes the qualifiers of all other locations in the nonvolatile memory to `CK`. Additionally, the location variable mapping γ is scoped down to γ' to cover only the remaining locations in N_1, i.e., $range(\gamma') = dom(\mathsf{N}_1)$.

The restore policy describes the effect of setting up the memories for re-executing an atomic region, by recreating the volatile logs with the checkpointed nonvolatile locations N'' and restoring the atomic region c_0, causing the program to re-execute from the beginning.

Finally, we define a power off policy to state the effect of the system in the event of a power failure. Because atomic regions already checkpoint the necessary locations before they begin executing, the `PwOff` policy here is defined to lose the volatile memory state and logs upon power failure. Like in commit, γ is scoped down to γ' to cover the remaining locations, i.e.,

Binary relation for commands

$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega \mid \Sigma \Vdash c_1 \leq c_2 : \mathsf{C}_{\mathtt{unit}}$
iff $\forall n, m \geq 0.\ \forall \gamma, \mathsf{N}, \mathsf{V}.s.t.\ \vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma.$
$(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1, \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$

Term relation

$$\begin{aligned}
\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1} &= \{(C_{o1}, C_{o2}^{\infty}) \mid \exists C'_{o1}\ s.t. C_{o1} \rightarrow^*_{irred} C'_{o1} \wedge \exists C_{o2}^{\infty'}\ s.t. C_{o2}^{\infty} \rightarrow^* C_{o2}^{\infty'} \wedge \\
&\qquad (C'_{o1}, C_{o2}^{\infty'}) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}\} \\
\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{0} &= \{(C_{o1}, C_{o2}^{\infty})\}
\end{aligned}$$

Value relation

$$\begin{aligned}
\mathcal{V}[\![\uparrow\mathtt{unit}]\!] &= \{(C_{o1}, C_{o2}^{\infty}) \mid C_{o1} = (\gamma \mid \mathtt{Md} \mid n_1 \mid \mathsf{N} \mid \mathsf{skip}) \wedge C_{o2}^{\infty} = (\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{skip})\} \\
\mathcal{V}[\![\downarrow\uparrow\mathtt{unit}]\!] &= \{(C_{o1}, C_{o2}^{\infty}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \mathsf{skip}) \wedge \\
&\qquad C_{o2}^{\infty} = (\gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid \mathsf{skip}) \wedge \\
&\qquad \mathtt{Commit}(\gamma_i, \mathtt{Md}, \mathsf{N}_i, \mathsf{V}_i) = \gamma'_i \mid \mathsf{N}'_i \wedge \\
&\qquad (\gamma'_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}'_1 \mid \mathsf{skip}, \gamma'_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow\mathtt{unit}]\!]\} \\
\mathcal{V}[\![\uparrow\mathsf{C}_{\mathtt{unit}}]\!]^m &= \{(C_{o1}, C_{o2}^{\infty}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \uparrow\kappa) \wedge \\
&\qquad C_{o2}^{\infty} = (\gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \wedge \\
&\qquad \mathtt{Restore}(\gamma_1, \mathtt{Md}, \mathsf{N}_1, \kappa) = \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \wedge \\
&\qquad (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m\} \\
\mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}]\!]^m &= \{(C_{o1}, C_{o2}^{\infty}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa)) \wedge \\
&\qquad \forall n{>}0. (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \uparrow\kappa, C_{o2}^{\infty}) \in \mathcal{V}[\![\uparrow\ \mathsf{C}_{\mathtt{unit}}]\!]^m\} \\
\mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})]\!]^m &= \{(C_{o1}, C_{o2}^{\infty}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa)) \wedge \\
&\qquad \mathtt{PwOff}(\gamma_1, \mathtt{Md}, \mathsf{N}_1, \mathsf{V}_1) = \gamma'_1 \mid \mathsf{V}' \wedge \\
&\qquad (\gamma'_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{V}'_{\mathsf{ck}}, \mathsf{N}_1 \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa), C_{o2}^{\infty}) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}]\!]^m\} \\
\mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1} &= \{(C_{o1}, C_{o2}^{\infty}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1) \wedge \\
&\qquad \textit{either}\ n_1 = 0 \wedge (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow c_1), C_{o2}^{\infty}) \\
&\qquad\qquad\qquad \in \mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\ \mathsf{C}_{\mathtt{unit}})]\!]^m, \\
&\qquad \textit{or} \quad n_1 > 0 \wedge (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, C_{o2}^{\infty}) \in \mathcal{V}[\![\downarrow\uparrow\ \mathtt{unit}]\!]\}
\end{aligned}$$

Figure 3.14: Step-indexed logical relation for crash types

$range(\gamma') = \mathsf{dom}(\mathsf{N}_1)$.

Now we are ready to explain our term and value relations.

Term Relation. A pair of open command configurations of type $\mathsf{C}_{\mathtt{unit}}$ are in the term relation of index m if any intermittent execution of the first configuration after m observed executions is indistinguishable from a continuously-powered execution of the second configuration. In the base case where the index is $m = 0$, no execution is observed, so any two configurations are in the term relation. In the inductive case where the index is $m + 1$, the term relation relates two configurations at type $\mathsf{C}_{\mathtt{unit}}$ if the first configuration eventually steps to a value configuration and the second configuration can take zero or more steps such that the pair continue to be in the value relation of $\mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$.

Value Relation. The value relation is defined based on the intended meaning of the type, and relates two value configurations that will have the same effect on the stores.

The last two rules omit the index m because the types $\downarrow\uparrow\mathtt{unit}$ and $\uparrow\mathtt{unit}$ are not recursive and thus no index is needed. The value relation at type $\uparrow\mathtt{unit}$ relates two configurations that have finalized their executions and thus requires they have the same nonvolatile memories N. The value relation at type $\downarrow\uparrow\mathtt{unit}$ relates two configurations that have completed their executions and right before they commit their changes to nonvolatile memory and requires that the commands in both configurations be skip and that after committing changes to their nonvolatile memory, the configurations be related at type $\uparrow\mathtt{unit}$. In other words, this requires these two configurations to have the same effect on nonvolatile memory.

The value relations in the last four rows of Fig. 3.14 are defined based on the type of the *first configuration*, as it goes through power failure, charging, and recovery, characteristic of intermittent execution. However, the second configurations in these relations do not encounter power failures at all and continue to be of type $\mathsf{C}_{\mathtt{unit}}$. Only in the relations defined in the first and second rows of the Fig. 3.14 term relation do the types of both configurations match the indexed type of the relation. Hence, the value relation has varying arity: in the first and second rows of Fig. 3.14, the relation is *binary* while in the rest, the relation degenerates to *unary*, with the second configuration as its Kripke world [66].

Each definition relates a configuration with a crash instruction to a continuously-powered execution. It includes conditions on the store that quantify the effects of the partial execution, crash, or recovery, so that these three rules together requires that previous partial execution, crashes, and recovery have no impact to the final nonvolatile memory compared to the continuously-powered execution and at the same time, allow unharmful effects of the partial execution. Because the structure of the crash type enforces that power failure, charging, and recovery happens in sequence, each definition applies effects of its own crash instruction and then hands off the resulting configuration to the previous definition.

The value relation at type $\uparrow\mathsf{C}_{\mathtt{unit}}$ requires that the intermittent configuration runs the crash instruction $\uparrow\kappa$, which restores a continuation of the execution. The restored configuration continues to be related to the continuously-powered configuration in the term interpretation at its original type $\mathsf{C}_{\mathtt{unit}}$.

The value relation at type $(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})$ requires the intermittent configuration to run an instruction for charging the energy. It is defined similarly to a function type in a value relation and requires the configurations to be related at type $(\uparrow\mathsf{C}_{\mathtt{unit}})$ for every energy input level n provided to the first configuration.

The value relation at type $\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})$ relates two configurations if the first one runs the crash instruction $\downarrow\varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa)$ which processes the power failure. The power failure policy creates a checkpoint of volatile locations such that the configurations continue to be in the value relation at type $(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})$.

Finally, the value relation relates two open command configurations at type $\mathsf{C}_{\mathtt{unit}}$ and index $m + 1$ if either (a) the first configuration has faced a power failure, and the two configurations continue to relate by $\mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})]\!]^m$, or (b) the first configuration executed successfully without any power failures, and the two configurations are related by $\mathcal{V}[\![\downarrow\uparrow\mathtt{unit}]\!]$. This definition matches the disjunctive nature of type $\mathsf{C}_{\mathtt{unit}}$, which is recursively defined in the signature as $\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Since we unfold the recursive definition of $\mathsf{C}_{\mathtt{unit}}$, we decrease the index from $m + 1$ to m to ensure the relation's well-foundedness. Note that the value relation is neither defined nor used by other definitions for $\mathsf{C}_{\mathtt{unit}}$ at index 0.

Semantic Typing. The top-level logical relation is written $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \Vdash c_1 \leq c_2 : \mathsf{C}_{\texttt{unit}}$ stating that each intermittent execution of c_1 yields the same nonvolatile memory as a continuously-powered execution of c_2, if it begins execution with well-formed memories w.r.t. Ω and Σ (see Fig. 3.14).

We formalize *semantic typing* as every atomic region of the program being logically-related to itself. The rule P-CKPT-SEMANTIC says that a program is semantically well-typed if every atomic region of it is self-related under our logical relation.

$$\frac{\begin{array}{c}\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; aPol) \\ \mathsf{aID}(c_0) \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma_0 \Vdash c_0 \leq c_0 : \mathsf{C}_{\texttt{unit}} \qquad b : \texttt{nat} \mid \Omega \Vdash p : \uparrow \mathsf{C}_{\texttt{unit}}\end{array}}{b : \texttt{nat} \mid \Omega \Vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\texttt{unit}}} \text{ (P-CKPT-SEMANTIC)}$$

Self-relatedness of a block helps us to build a continuously-powered execution for each intermittent execution of it as required for idempotency. We give the theorem and proof that a program comprised of only self-related blocks is idempotent in Section 3.6.

The P-CKPT-SEMANTIC rule allows us to reason about programs by considering each code block independently. We can extend this aproach to accomodate other code blocks in a program. In the next section, we extend the system to accomodate just-in-time (JIT) blocks and explain how to compose them with atomic regions in programs.

3.5 JIT and Hybrid Program Execution

Up until this point, we have focused our discussion entirely on atomic regions. In this section, we extend the system we have developed thus far to also include just-in-time (JIT) regions, written $c; p$, where c is the JIT region and p is the rest of the program. We add another mode for JIT regions, denoted jit, which is used to index our operational semantics and typing rules. Below, we give the full syntax of programs and execution modes extended to accommodate JIT programs:

$$\begin{array}{llll} \textit{programs} & p & ::= & \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}; p \mid c; p \mid \mathsf{skip} \\ \textit{execution mode} & \texttt{Md} & ::= & \mathsf{aID}(c) \mid \mathsf{jit} \end{array}$$

In Section 3.5.1, we explain the example JIT region from Chapter 2. In Section 3.5.2, we extend the calculus from above with operational semantic rules for executing JIT regions. Sections 3.5.4 and 3.5.3 introduce types and typing rules specific to JIT mode. In Section 3.5.5, we revisit the semantic typing and logical relation considering JIT mode.

3.5.1 Example of JIT Region Execution and Typing

Unlike atomic regions, JIT regions are not susceptible to memory consistency bugs because they do not re-execute. Instead, all volatile state is checkpointed in nonvolatile memory just before a power failure so that upon reboot, program execution resumes from the point where power had failed. Due to this, reasoning about memory accesses in JIT mode is not needed. As a result, all variables in a JIT region are annotated with the qualifier CK. Fig. 3.15 shows the details of executing the JIT region from Chapter 2 Fig. 2.2.

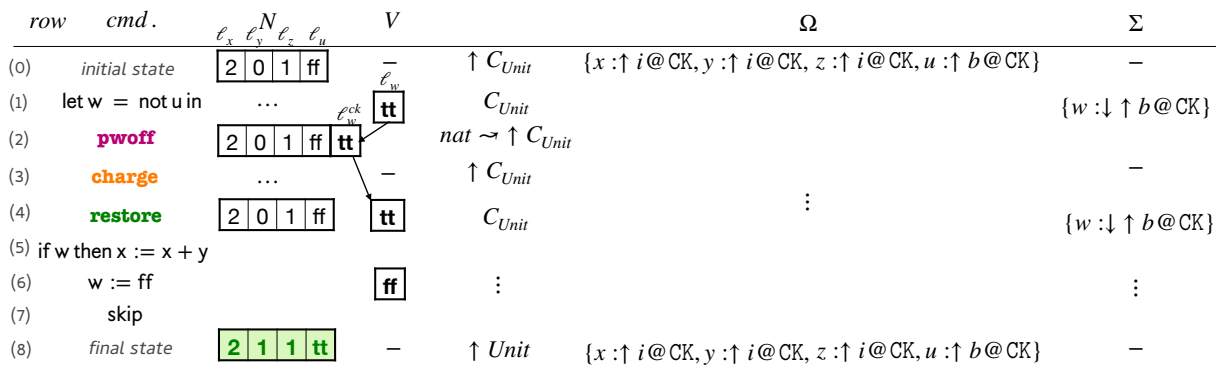

Figure 3.15: Intermittent execution of a JIT region. We write i for `int` and b for `bool`.

Row (0) shows the initial nonvolatile locations, their values, and the mapping from variables to locations. The system starts executing the JIT region by creating an empty context to be populated by volatile locations (Row (1)). The `let` construct allocates a fresh location ℓ_w in volatile memory corresponding to the variable w. On a power failure in JIT mode, the system creates a nonvolatile copy of the volatile location ℓ_w just before it loses the location (Row (2)). It marks the nonvolatile copy with the superscript `ck`. When resuming program execution, the system restores these copies to volatile memory which it accesses during execution, dismissing their nonvolatile backups (Row (3)). Execution then continues with the `if` clause, finally dropping the volatile location ℓ_w, as it is out of scope (Row (7)).

Shifts in JIT Mode (Fig. 3.15): We explain how to build typing contexts for JIT programs, again using the rules from Chapter 2 and the execution in Fig. 3.15. The process of constructing typing contexts for JIT regions is the same as for atomic regions. However, all variables in JIT regions are checkpointed, and thus, the resulting typing contexts differ from those for atomic regions even when typing the same commands. In Fig. 3.15, we create an empty volatile context Σ when starting the JIT region (the step from Row (0) to Row (1)) by an application of the $\uparrow R$ rule. A combination of the rules $\downarrow L$ and $\downarrow R$ corresponds to a power failure, i.e., the stepping from Row (1) to Row (2) in Fig. 3.15. An application of the $\downarrow L$ rule copies the variable w of type $\downarrow^s_u \uparrow^s_u b$ in Fig. 3.15 from volatile memory context Σ into nonvolatile memory context Ω. A $\downarrow R$ rule application closes off the (empty) nonvolatile memory context. As in atomic mode, a combination of $\uparrow R$ and $\uparrow L^*$ rules corresponds to restoring a nonvolatile location to volatile memory (e.g., w) when restarting the command after the failure, i.e., the step from Row (2) to Row (3). The $\uparrow R$ rule prepares for re-execution by setting up an empty volatile context into which the $\uparrow L^*$ rule copies variable w from the nonvolatile context. We assume an extra weakening rule to eliminate the remaining variable w in nonvolatile memory context. The dropping of the volatile memory context at the end of execution (Row (6)) is not a modal step, but rather follows from a standard rule for the let clause. Lastly, in Row (7), an application of $\downarrow R$ drops the volatile typing context and produces the stable type $\uparrow^s_u \texttt{unit}$, detecting a successful execution.

3.5.2 Operational Semantics for JIT Regions

The operational semantics for JIT regions are of the same forms as those for atomic regions: $C_c \Rightarrow_p C_c'$ (program level), $C_c \Rightarrow C_c'$ (command level), and $C_o \to C_o'$ (expression level).

We present selected operational semantic rules for JIT programs in Fig. 3.16.

$$\frac{n' > 0 \qquad \begin{array}{c} n > 0 \\ [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \cdot \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid c; p \Rightarrow_p [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p} \text{ (D-P-SEQ)}$$

$$\frac{}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \downarrow\varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa) \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \mathsf{N}, \mathsf{V}_{\mathsf{ck}} \mid \varepsilon \# \mathsf{in}(b > 0; \uparrow\kappa)} \text{ (D-S-JIT)}$$

$$\frac{}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N}, \mathsf{V}_{\mathsf{ck}} \mid \uparrow \kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \kappa} \text{ (D-RESTORE-JIT)}$$

Figure 3.16: Selected operational semantic rules for JIT regions

Top-Level Program Execution. The D-P-SEQ rule applies when the next code block is a regular command c. The closed configuration of c with an empty initial set of volatile locations is fully evaluated in JIT mode. In Fig. 3.15, we use the rule D-P-SEQ to set up an initially empty volatile context to begin executing the JIT region (the step from Row (0) to Row (1)), and to step the JIT region (from Row (1) to Row (6)) according to the third premise. Then the resulting volatile locations V' scoped in c are dropped because in JIT mode the nonvolatile memory always stores up-to-date values and V' only contains out-of-scope let variables. This corresponds to the step from Row (6) to Row (7) in Fig. 3.15. The constraint $n > 0$ checks that there is energy available for the first code block to take a step, and $n' > 0$ checks that the first code block finishes executing successfully and is not in the midst of a power failure.

Together, the rules D-P-SEQ and D-P-CKPT allow running programs composed of both JIT and atomic regions, and the combination of rules T-P-SEQ and T-P-CKPT allows us to statically check them.

Command Execution (Closed Config). The rule D-S-JIT corresponds to checkpointing, and stores all volatile memory in nonvolatile locations. The rule D-RESTORE-JIT moves the checkpointed locations into volatile memory, thereby dropping the checkpointed locations from the scope of the nonvolatile memory. D-RESTORE-JIT recovers the interrupted command κ for the program to resume execution.

3.5.3 Store Typing

We present the store typing rules for JIT mode in Fig. 3.17: EMPTY for checking empty stores, V-LOC-JIT for checking volatile locations, and NV-LOC-JIT for checking nonvolatile locations. The rule V-LOC-JIT is similar to the volatile store typing rule for atomic mode. The rule checks that the volatile location ℓ has the qualifier CK, and that the type of the value v stored in ℓ matches the type of the corresponding variable x in the volatile typing context Σ. The first premise recursively

checks the rest of the volatile store. The rule NV-LOC-JIT checks a nonvolatile location ℓ by ensuring that the variable x, for which ℓ is allocated, has the same type in Ω as the value v stored in ℓ and that the qualifier $q = \mathtt{CK}$. The rule then recursively checks the rest of the store.

$\boxed{\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma}$

$$\frac{}{\vdash^{\mathsf{jit}}_{\gamma} \cdot \mid \cdot : \cdot \mid \cdot} \text{(EMPTY)}$$

$$\frac{\mathsf{V} = \mathsf{V}', \ell \mathbin{@} q \hookrightarrow v \qquad q = \mathtt{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} v : \uparrow A \qquad \vdash^{\mathsf{jit}}_{\gamma'} \mathsf{N} \mid \mathsf{V}' : \Omega \mid \Sigma}{\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma, (x{:}\downarrow\uparrow A @ q)} \text{(V-LOC-JIT)}$$

$$\frac{\mathsf{N} = \mathsf{N}', \ell \mathbin{@} q \hookrightarrow v \qquad q = \mathtt{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} v : \uparrow A \qquad \vdash^{\mathsf{jit}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} : \Omega \mid \Sigma}{\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, (x{:}\uparrow A @ q) \mid \Sigma} \text{(N-LOC-JIT)}$$

Figure 3.17: Well-formedness of $\mathsf{N} \mid \mathsf{V}$ w.r.t. $\Omega \mid \Sigma$ in JIT mode

3.5.4 Static Typing

We extend the type system to accommodate JIT regions by introducing crash types and typing rules for JIT mode, summarized in Figs. 3.18 and 3.19.

Crash Types. We begin by extending the definition of type variables $\mathsf{C}^{\mathtt{Md}}_{T}$. Because the mode has been extended to accommodate both atomic regions and JIT regions, we replace the single type variable $\mathsf{C}^{\mathtt{Md}}_{A}$ with a type variable for each mode: $\mathsf{C}^{\mathsf{atom}}_{A}$ and $\mathsf{C}^{\mathsf{jit}}_{A}$. The definition of crash type for commands in JIT mode ($\mathsf{C}_{\mathtt{unit}}$) is the same as for atomic mode, where the right disjunct represents successful execution and the left disjunct represents a power failure. To capture the JIT policy, the type $\mathsf{C}_{\mathtt{unit}}$ in the left disjunct represents the same command that was interrupted by the power failure. The definition of expression crash type for JIT mode ($\mathsf{C}^{\mathsf{jit}}_{A}$) is similar to the one for atomic mode ($\mathsf{C}^{\mathsf{atom}}_{A}$), where the right disjunct types a successful execution and the left disjunct represents power failure. However, the type $\uparrow\mathsf{C}^{\mathsf{jit}}_{A}$ in the left disjunct captures the JIT recovery policy, that an interrupted run of an expression in JIT mode will be restored to the expression itself.

Program Typing. To type JIT programs, we extend our type system with the rule T-P-SEQ. The T-P-SEQ rule types program $c; p$ by first typing c under JIT mode with $b \geq 0$, which allows for the possibility of power failure. The command c is typed under an empty volatile memory typing context, which will be populated when let commands in c allocate new volatile locations. The signature `Sig` for typing c is also empty and will be populated later at individual points of failure (rule T-ENOUGH?). The second premise types the rest of the program p under the nonvolatile typing context Ω.

Command, Expression, and Statement Typing. The command and expression typing rules are extended to accommodate JIT mode which just entails updating the T-ENOUGH? rules for

$$
\begin{array}{lllll}
\textit{type variables} & \mathsf{C}_T^{\mathtt{Md}} & ::= & \mathsf{C}_{\mathtt{unit}} \mid \mathsf{C}_A^{\mathsf{atom}} \mid \mathsf{C}_A^{\mathsf{jit}} \\
\textit{command crash type} & \mathsf{C}_{\mathtt{unit}} & ::= & \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{unit} \\
\textit{atomic crash type} & \mathsf{C}_A^{\mathsf{atom}} & ::= & \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow A \\
\textit{jit crash type} & \mathsf{C}_A^{\mathsf{jit}} & ::= & \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_A^{\mathsf{jit}}) \vee \downarrow\uparrow A
\end{array}
$$

Figure 3.18: Crash types for both JIT and atomic mode

$$
\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash c; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-SEQ)}
$$

$$
\frac{\begin{array}{c}\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\} \qquad \mathtt{Sig}'' = \textit{if } \mathtt{Md} = \mathsf{jit}, \textit{then } \mathtt{Sig}', \textit{else } \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'} \text{ (T-ENOUGH?)}
$$

$$
\frac{\begin{array}{c}\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\} \qquad \mathtt{Sig}'' = \textit{if } \mathtt{Md} = \mathsf{jit}, \textit{ then } \mathtt{Sig}', \textit{else } \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau} \text{ (T-ENOUGH?)}
$$

$$
\frac{\Sigma = \downarrow\uparrow\Sigma' \qquad \mathsf{jit} \mid \cdot \mid \Omega, \uparrow\Sigma'_{\mathtt{ck}} \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_T^{\mathtt{Md}})}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_T^{\mathtt{Md}})} \text{ (T-JIT-STOP)}
$$

$$
\frac{\Omega = \Omega', \Omega''_{\mathtt{ck}} \qquad \mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \downarrow\Omega'' \vdash \kappa' : \mathsf{C}_T^{\mathtt{Md}} \in \mathtt{Sig}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow\kappa' :\uparrow \mathsf{C}_T^{\mathtt{Md}}} \text{ (T-JIT-RESTORE)}
$$

Figure 3.19: Selected typing rules for JIT regions

commands and expressions to populate the signature $\mathtt{Sig}''$ at the appropriate place. In particular, both rules are extended with a premise stating that the signature remains unchanged if the mode is atomic, but is populated by $\mathtt{Sig}'$ (the current command typing) if the mode is JIT. In JIT mode, after a power failure, the command c is restored to itself, and $\mathtt{Sig}'$ remembers that the well-typedness of the command (when the energy level is non-negative) has already been checked.

Typing rules for crash instructions also change to adopt the stop and restore policies for JIT programs. The T-JIT-STOP rule brings a checkpointed version of all the volatile variables in Σ inside Ω since they are checkpointed upon power failure. Once an energy input is received from the environment ($b > 0$), the rule T-JIT-RESTORE applies to type the restored command. Here, volatile memory is restored from the checkpointed locations in Ω. Checkpoints are dropped from Ω and execution resumes with the expression or command κ, which is the code running right before the power failure.

3.5.5 Logical Relation

Our logical relation from Section 3.4 is designed to reason about JIT regions too, but according to different policies. We provide policy definitions for JIT mode in Fig. 3.20. The policies are defined to match the dynamic rules for crash, restore, and re-execution for JIT execution (D-S-JIT, D-RESTORE-JIT, and D-P-SEQ).

$$\frac{\mathsf{N}_1 = \mathsf{N}^1_{\mathsf{RD}}, \mathsf{N}^2_{\mathsf{ck}} \qquad \mathsf{N}_2 = \mathsf{N}^1_{\mathsf{ck}}, \mathsf{N}^2_{\mathsf{ck}} \qquad \mathsf{range}(\gamma') = \mathsf{dom}(\mathsf{N}_1) \qquad \gamma' \subseteq \gamma}{\texttt{Commit}(\gamma, \mathsf{jit}, \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{N}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}', \mathsf{N}''_{\mathsf{ck}} \qquad \mathsf{N}_2 = \mathsf{N}' \qquad \mathsf{V}_2 = \mathsf{N}''}{\texttt{Restore}(\gamma, \mathsf{jit}, \mathsf{N}_1, \kappa) = \mathsf{N}_2 \mid \mathsf{V}_2 \mid \kappa} \qquad \frac{\mathsf{V}_2 = \mathsf{V}_1 \qquad \gamma' = \gamma}{\texttt{PwOff}(\gamma, \mathsf{jit}, \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2}$$

Figure 3.20: `Commit`, `Restore`, and `PwOff` policy definitions for JIT mode

The `Commit` policy for JIT mode simply drops all volatile memory and changes the qualifiers of all locations in nonvolatile memory to `ck`. Additionally, the original map γ covers the committed map γ' ($\gamma' \subseteq \gamma$), and the locations mapped to by γ' comprise the locations of N_1 ($range(\gamma') = dom(\mathsf{N}_1)$).

The `Restore` policy for JIT mode prepares for execution to resume by extracting let-bound variables that were stashed in nonvolatile memory N'' back to their volatile locations and returning the current continuation κ. We write $\mathsf{N}_1 = \mathsf{N}', \mathsf{N}''_{\mathsf{ck}}$ to state that N_1 can be uniquely partitioned into all locations that are checkpointed ($\mathsf{N}''_{\mathsf{ck}}$), which are of the form $\ell_{\mathtt{ck}}$, and regular locations (N') of the form ℓ. N'' is the non-checkpointed version of $\mathsf{N}''_{\mathsf{ck}}$ which could be retrieved by removing the `ck` subscript from every location in $\mathsf{N}''_{\mathsf{ck}}$.

Lastly, the `PwOff` policy for JIT mode is defined to checkpoint all volatile locations in nonvolatile memory.

Semantic Typing. Lastly, we extend our semantic typing definition to accommodate JIT regions too. We do so with the rule P-SEQ-SEMANTIC which says that a program of the form $c; p$ is semantically well-typed if the JIT region c is self-related under the logical relation and if the remaining program p is semantically well-typed.

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega \mid \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash c; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-SEQ-SEMANTIC)}$$

For programs composed of a combination of JIT and atomic regions, the semantic typing rules P-SEQ-SEMANTIC and P-CKPT-SEMANTIC indicate that a program is semantically well-typed if each of its program blocks (JIT *and* atomic regions) is self-related under our logical relation.

3.6 Metatheory

This section establishes the main properties of our system: progress and preservation, adequacy—which states that intermittent executions of semantically well-typed programs are correct—and

the fundamental theorem of logical relation which states that statically well-typed programs are semantically well-typed. Combining the latter two, it follows that statically well-typed programs can be correctly executed intermittently. The detailed proofs of these theorems are provided in Appendix A.

3.6.1 Statically Well-Typed Programs are Type Safe

We prove type safety according to progress and preservation. The progress and preservation theorems assume that memory locations are well-formed, $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. The progress theorem (formally defined below) states that if a command c is well-typed, then for all energy levels n and well-formed memories N and V, either the configuration of c is a value configuration or it can take a step according to the operational semantics.

Theorem 1 (Progress for Commands) *If* $\mathtt{Md} \mid b \; \mathcal{R} \; m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$, *then* $\forall \; n : \mathtt{nat}$ *with* $n \; \mathcal{R} \; m$ *and* $\forall \gamma, \mathsf{N}, \mathsf{V}$ *with* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, *either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ *or*
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ *s.t.* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \; \rightarrow \; \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Here, $m = 0$ and the configuration's concrete energy level n instantiates energy variable b in the typing judgment. The energy level decrements with each command step until the configuration of c is a value where $n > 0$ (indicating a successful execution), or until $n = 0$ (indicating a crash) which our semantics also considers a value. The constraint $n \; \mathcal{R} \; m$ in the theorem ensures that the concrete energy level n preserves the relation of b with m as required by typing. Note that this progress theorem does not ensure that a given program will always make enough progress to complete the execution, because sufficient energy may not be available.

The preservation theorem for commands is formally defined below.

Theorem 2 (Preservation for Commands) *If* $\mathtt{Md} \mid b \; \geq \; 0 : nat \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$ *where*

- $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,
- $n : \mathtt{nat} \geq 0$, *and*
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \; \rightarrow \; \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$

then for some Σ', *we have* $\mathtt{Md} \mid b \; \geq \; 0 : nat \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c' : \tau$ *where*

- $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma'$ *and*
- $n' : \mathtt{nat} \geq 0$.

Theorem 2 states that if c is a syntactically well-typed command that, with well-formed memories N and V and nonnegative energy n, steps to another configuration $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$, then c' is also syntactically well-typed, the memories N' and V' are well-formed, and n' is a nonnegative natural number. Together, Theorems 2 (Progress for commands) and 2 ensure the soundness of execution between power failures, where energy is available. Their full proofs are included in Appendix A.1.

3.6.2 Statically Well-Typed Programs are Semantically Well-Typed

To prove the soundness of intermittent execution, where power failure may occur, we rely on the logical relation defined in Section 3.4. In particular, we prove the soundness of our static

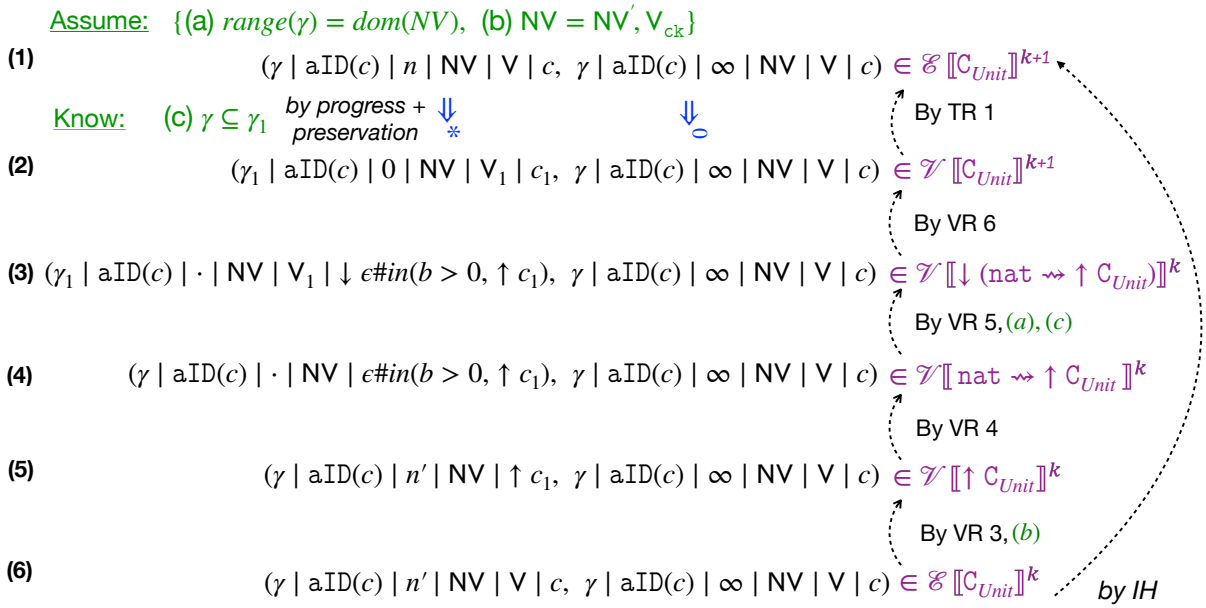

Figure 3.21: Proof of the fundamental theorem for T-P-Ckpt (inductive case)

typing rules with respect to the semantic typing rules (Theorem 3). We sketch the proof below and provide the full proof in Appendix A.2.

Theorem 3 (Fundamental Theorem) *If* $b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}$, *then* $b : \mathtt{nat} \mid \Omega \Vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}$.

The proof of Theorem 3 is by induction on the static typing derivation for p and considers the last step in the derivation.

Typing an atomic region: We sketch the high level idea behind the proof for the case where the last step of the derivation is T-P-Ckpt. By inversion, $p = \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}; p'$ where $aPol = (\omega, \rho)$. Also, c is well-typed for static contexts Ω' and Σ, where $\Omega' = \Omega'', \Sigma_{\mathsf{ck}}$.

The goal is to establish that c is related to itself in the term interpretation for arbitrary n, m, γ, N and V where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$. This means that we need to prove $(\gamma \mid \mathsf{aID}(c) \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c, \gamma \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$ for all indices m given that the condition $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ holds. The last condition enforces that the static contexts match the dynamic context. In particular, $\mathsf{N} = \mathsf{N}', \mathsf{V}_{\mathsf{ck}}$ and $range(\gamma) = dom(\mathsf{N})$.

Base Case. For $m = 0$, the logical relation in Fig. 3.14 relates any two configurations with the index zero. Thus, the base case immediately follows from the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$.

Inductive Case. Now we discuss the inductive case where $m = k + 1$. By the progress and preservation theorems, the first configuration can take multiple steps until it becomes a value $\gamma_1 \mid \mathsf{aID}(c) \mid n' \mid \mathsf{N} \mid \mathsf{V}_1 \mid c_1$ that continues to be well-typed. If $n' > 0$, then the intermittent execution was successful, in which case the second configuration steps similarly to completion such that the two resulting configurations are in the value relation.

Alternatively, if $n' = 0$, then the intermittent execution failed. Fig. 3.22 sketches the proof of this case. The green conditions (a) and (b) are known by assumption, and (c) is learned while conducting the proof. The condition (a) asserts that γ covers the locations in the nonvolatile memory N, and the condition (b) asserts that the initial log V matches the values of checkpointed locations in the intial nonvolatile memory N.

In this case, the second configuration does not step and instead reaches point (2), where the initial variable location mapping is a subset of the stepped variable location mapping: (c) $\gamma \subseteq \gamma_1$. At point (2), the proof must show that the configurations are in the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$.

The dashed line in the figure states that establishing point (2) implies the relation in point (1). In general, the cascade of implications (dashed lines) follows the definition of the value relations at each type. At each step, we invert on the typing rule of the open configuration and show that runtime memories stay well-defined for static contexts. For example, to prove point (1), it suffices to establish point (2) which follows by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$. To prove point (2), it is enough to prove point (3), and so on. The proof continues unfolding the logical relation in this manner until point (6) is reached. Point (6) states that c is self-related by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$ but at the smaller index k. By establishing the appropriate conditions, we can apply the inductive hypothesis to establish the desired result. This is the high-level proof strategy, and we provide more detail in the explanation below.

At point (4), we apply the power failure policy for atomic regions, which drops the volatile memory V' and creates a mapping using the domain of N'. By the prior conditions established, we know the resulting mapping is the original mapping γ which covers nonvolatile memory N.

At point (6), we apply the restore policy for atomic regions, which creates a new volatile log based on N.

By the semantics of the restore function, we know that the volatile log created is the original log V containing only checkpointed locations.

The goal at point (6) is similar to our original goal at point (1), but at a smaller index k. We apply the inductive hypothesis to establish the desired result.

Typing a JIT region: Now, we sketch the proof for the case where the last step of the derivation is T-P-Seq. Since there is no re-execution, the reasoning is simpler than atomic regions. By inversion, $p = c; p'$. Also, c is well-typed for static contexts Ω and Σ. Unlike the atomic region case, initially there are no locations tagged as $\mathtt{ck}$.

The goal is to establish that c is related to itself in the term interpretation for arbitrary n, m, γ, N and V where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. This means that we need to prove $(\gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c, \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$ for all indices m given that the condition $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ holds. The last condition enforces that the static contexts match the dynamic context. In particular, $\mathit{range}(\gamma) = \mathit{dom}(\mathsf{N})$.

Base Case. For $m = 0$, the logical relation in Fig. 3.14 relates any two configurations with the index zero. Thus, the base case immediately follows from the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$.

Inductive Case. Now we discuss the inductive case where $m = k + 1$. By the progress and preservation theorems, the first configuration can take multiple steps until it becomes a value

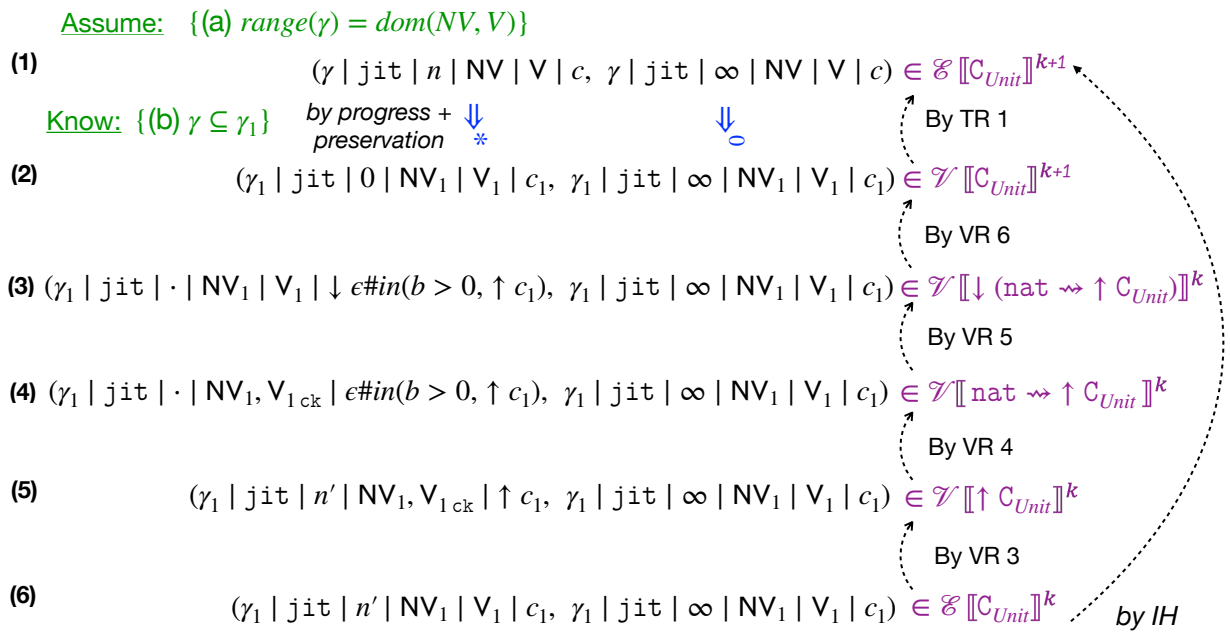

Figure 3.22: Proof of the fundamental theorem for T-P-Seq (inductive case)

$\gamma_1 \mid \mathsf{aID}(c) \mid n' \mid \mathsf{N} \mid \mathsf{V}_1 \mid c_1$ that continues to be well-typed. If $n' > 0$, then the intermittent execution was successful, in which case the second configuration steps similarly to completion such that the two resulting configurations are in the value relation.

Alternatively, if $n' = 0$, then the intermittent execution failed. Fig. 3.22 sketches the proof of this case. The green condition (a) is known by assumption, and (b) is learned while conducting the proof. The condition (a) asserts that γ covers the locations in the nonvolatile memory N. In this case, the second configuration *steps in lockstep with the first configuration* and reaches point (2), where the initial variable location mapping is a subset of the stepped variable location mapping: (b) $\gamma \subseteq \gamma_1$. At point (2), the proof must show that the configurations are in the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$.

At point (4), we apply the power failure policy for JIT regions, which *stashes the volatile memory* V_1 *into nonvolatile memory*. The subscript ck indicates which portion of the resulting memory is actually checkpointed.

At point (6), we apply the restore policy for JIT regions, which repopulates volatile memory with the locations in nonvolatile memory that were marked ck.

The goal at point (6) is similar to our original goal at point (1), but at a smaller index k. We apply the inductive hypothesis to establish the desired result.

3.6.3 Semantically Well-Typed Programs are Idempotent

A program is idempotent if every intermittent execution of it can be simulated by a continuous execution. Definition 5 defines this property formally.

Definition 5 (Idempotency) *A triple of a program p, nonvolatile memory* N*, and a mapping* γ *is idempotent, if every intermittent execution of the program can be simulated by a continuous*

execution of it: for all $n, n', \chi_1, \chi_1', \mathsf{N}', p'$*, if* $[\chi_1 \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid\ p \Rightarrow_p [\chi_1' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'$*, then* $[\chi_2 \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N} \mid\ p \Rightarrow_p [\chi_2 \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N}' \mid p'$.

When showing that a program p with memory N and mapping γ is idempotent, we construct a simulation of the continuous execution assuming that it begins with the same program code p and memory N as the intermittent execution. We use the logical relation from Section 3.4 to enforce that the stepped configurations are related in the same way; they share the same program code p' and memory N'. The charge stream χ_2 remains unchanged between the initial and final configurations as the continuous execution does not experience power failures.

Theorem 4 states that semantically well-typed programs are idempotent. In the theorem, we use the store typing judgment for the jit mode to ensure the well-formedness of N with respect to a typing context Ω. This reflects the fact that the programs start executing in jit mode by default.

Theorem 4 (Adequacy) *Consider* $b : \mathtt{nat} \mid \Omega \Vdash p : \mathsf{C}_{\mathtt{unit}}$*, a nonvolatile memory* N*, and a map* γ *such that* $\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \cdot : \Omega \mid \cdot$*. The triple of p,* N*, and* γ *is idempotent.*

We include the full proof in Appendix A.3, and sketch the reasoning below. The proof builds the simulation required by idempotency using induction on the number of observable attempts during each step of a program's intermittent execution. The construction of this simulation (and proof) is illustrated in Fig. 3.23. The figure presents the trace of the intermittent execution on the left and builds the corresponding continuous execution on the right. The purple text shows the relation between the two configurations at each step according to our logical relation. The orange box highlights the steps taken during a power failure. The green box highlights where the inductive hypothesis is applied. The red box highlights the base case. Lastly, the blue text gives conditions that are known by assumption or are learned during the proof.

Executing an atomic region: Assume that c_1 is self-related and that the configuration $[\chi_1 \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID} \mid n \mid \mathsf{N}_1 \mid\ \mathsf{start}_{\mathsf{atom}}(\overline{\mathsf{aID}}, \mathit{aPol}); c_1; \mathsf{end}_{\mathsf{atom}}; p$ takes a step to $[\chi_k \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid n' \mid \mathsf{N}' \mid p$ via the rule D-P-Ckpt while incurring possibly m-many power failures. We need to show that there exists a continuously-powered execution of the same program that steps the configuration $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID} \mid \infty \mid \mathsf{N}_1 \mid\ \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \mathit{aPol}); c_1; \mathsf{end}_{\mathsf{atom}}; p$ to $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid \infty \mid \mathsf{N}' \mid p$. Our proof constructs such continuously-powered execution.

The proof assumes that program $\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \mathit{aPol}); c_1; \mathsf{end}_{\mathsf{atom}}; p$ is semantically well-typed. Applying P-Ckpt-semantic, we learn that the command c_1 is semantically well-typed and self-related at type $\mathsf{C}_{\mathtt{unit}}$. By the definition of logical relation, it follows that the intermittent configuration of c_1 with energy level n is related to the continuously-powered configuration of c_1 with energy level ∞ at the type $\mathsf{C}_{\mathtt{unit}}$ for every index, including $m + 1$. We use this relation to build the desired simulation. Towards this goal, we prove a more general result: for any two related configurations, if the first configuration takes zero or more steps, then the second configuration takes zero or more steps such that the two stepped configurations remain related. By definition of our relation, both configurations result in the same final nonvolatile memories.

The proof is by induction on the number of failures m. Fig. 3.23 shows the inductive case (lines (1)-(6)) and the base case (lines (7)-(9)). Starting with index $m + 1$, representing the number of remaining attempts (point (1) in Fig. 3.23), we build a continuously-powered execution for c_1 while showing that the intermediate configurations continue to relate to their intermittent counterparts at a smaller index m (point (6) in Fig. 3.23). At that point, we apply the inductive

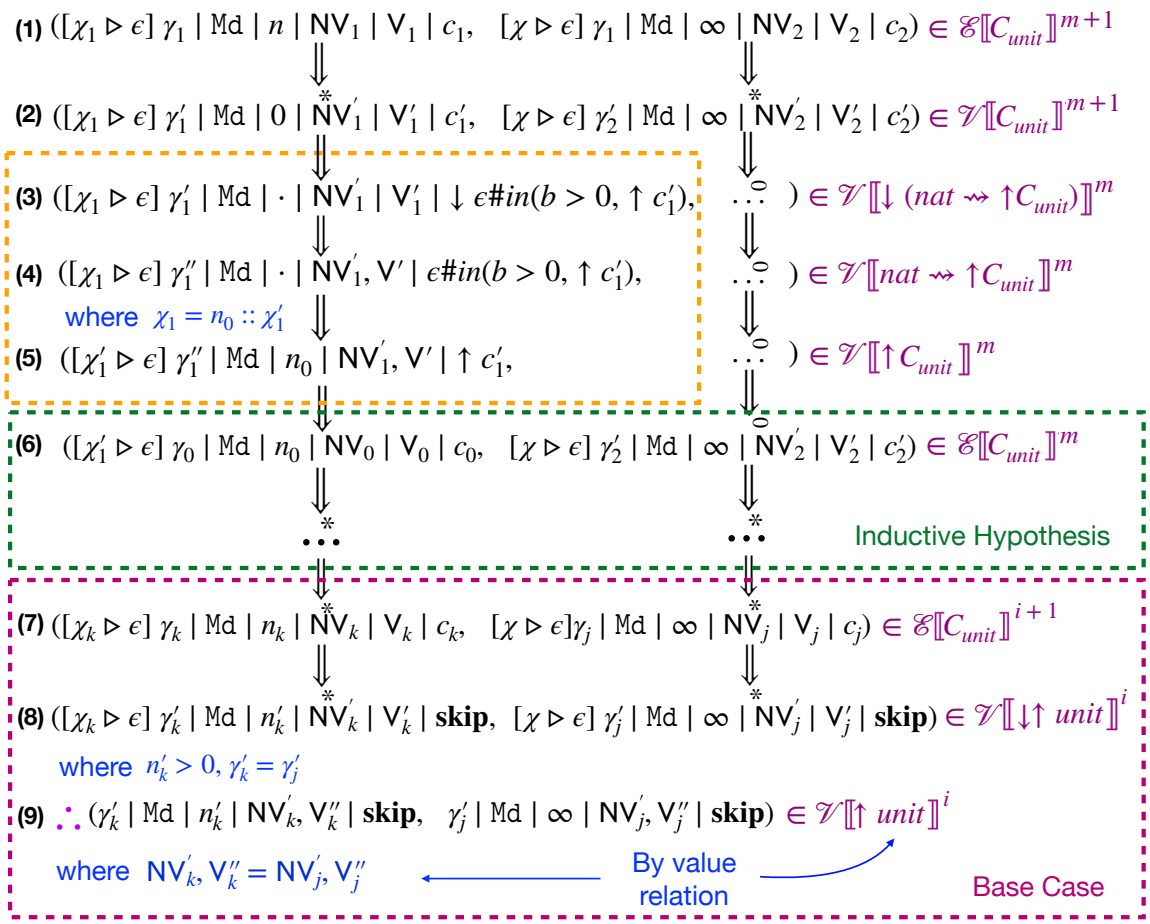

Figure 3.23: Adequacy proof sketch.

hypothesis to obtain the desired result. Finally, the base case establishes that the memories of the final configurations are the same.

Inductive Case (Fig. 3.23, lines (1)-(6)). By definition of the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$, c_1 in the first configuration runs via D-STEP until the next power failure. Moreover, it follows by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$ that the second configuration can also be executed such that the resulting configurations remain related (point (2)). The first configuration takes a step from point (2) to point (3) with the rule D-CRASH. Since the energy level of the first configuration is 0, it follows by the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$ that these two configurations are related by the value interpretation at type $\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})$ (point (3)). Using the rule D-S-AID, the first configuration takes a step from point (3) to point (4). The volatile memory V_1' is dropped, and hence $\mathsf{V}' = \emptyset$. Note that this matches the power off policy for atomic mode. By assumption to D-S-AID, the mapping γ_1'' drops the discarded volatile memory locations from γ_1' such that $\gamma_1'' \subseteq \gamma_1'$. The two configurations remain related by the value interpretation at type $\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}$ (point (4)). The first configuration then takes another step to point (5) by D-CHARGE, inputting the harvested energy n_0, thus updating the energy stream to χ_1' where $\chi_1 = n_0 :: \chi_1'$. By definition

of the value relation, the two configurations remain related at the type $\uparrow \mathsf{C}_{\mathtt{unit}}$ for all n_0 (point (5)).

From point (5) to point (6), the first configuration steps via D-RESTORE-AID. In doing so, it loads the checkpointed data of the nonvolatile memory into a new volatile log V_0. As such, the program counter resets to its original value, and the atomic region re-executes from the beginning such that $c_0 = c_1$. The variable mapping stays the same ($\gamma_0 = \gamma_1''$). Note that this step matches the restore policy for atomic mode. The two configurations are then related by the term interpretation at the type $\mathsf{C}_{\mathtt{unit}}$. This is similar to what we had initially, but with one fewer power failure remaining. At this point, we apply the inductive hypothesis to build the simulation at the smaller index m.

Base Case (Fig. 3.23, lines (7)-(9)). When the first configuration finally steps to completion by the D-STEP rule (point (8) in Fig. 3.23), it follows by the definition of logical relation that the second configuration also steps to skip thus detecting a successfully completed execution. This completes the simulation of the continuous execution of c_1. Here, we can apply the rule D-P-CKPT on this construction to obtain a continuously-powered execution of the program $\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p$. The $\mathsf{FinWorld}_d$ function copies the up-to-date values of volatile memory V_k' that have checkpointed locations back into nonvolatile memory N_k', removes the subscript ck from these locations, and changes the qualifiers of all locations in nonvolatile memory to CK. The volatile memory is dropped and the mapping is reset to γ. This matches the commit policy defined for atomic mode. By the value interpretation at type $\downarrow\uparrow\mathtt{unit}$, the two configurations remain related at the type $\uparrow\mathtt{unit}$ (line (9)). Then, by the value interpretation at type $\uparrow\mathtt{unit}$, the final nonvolatile memories are the same ($\mathsf{N}_k', \mathsf{V}_k'' = \mathsf{N}_j', \mathsf{V}_j''$) which is the last piece we needed.

For the case corresponding to initial execution and re-execution, where $c_2 = c_1$, $\mathsf{N}_2 = \mathsf{N}_1$, and $\mathsf{V}_2 = \mathsf{V}_1$, we can apply the rule D-P-CKPT to the constructed execution to find a continuously-powered execution of $\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p$ as desired: $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID} \mid \infty \mid \mathsf{N}_1 \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p \Rightarrow_p [\chi \rhd \varepsilon] \otimes \ \gamma \mid \mathsf{aID} \mid \infty \mid \mathsf{N}' \mid p$.

Executing a JIT region: Assume that c_1 is self-related, and $[\chi_1 \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid c_1; p$ takes a step to $[\chi_k \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid p$ via the D-P-SEQ rule with possibly m-many power failures along the way. We need to show that there exists a continuously-powered execution stepping the configuration $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1 \mid c_1; p$ to $[\chi \rhd \varepsilon] \otimes \ \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid p$. Our proof constructs such continuously-powered execution.

The overall proof structure is the same as the case for atomic regions. Similarly, the key idea is that the power off and restore policies in the logical relation exactly follow how the rules D-S-JIT and D-RESTORE-JIT handle nonvolatile and volatile memories in the operational semantics.

The proof assumes that $c_1; p$ is semantically well-typed, which by P-SEQ-SEMANTIC, yields that c_1 is semantically well-typed and self-related at type $\mathsf{C}_{\mathtt{unit}}$. By definition of logical relation, if c_1 is semantically well-typed, then its intermittent configuration with energy level n is related with the same configuration but with energy level ∞ for every index, including $m + 1$. We use this condition to obtain the desired result. In particular, we prove a more general goal stating

that for any two related configurations, if the first configuration takes zero or more steps, then the second configuration can take zero or more steps such that the stepped configurations remain related. Moreover, in both configurations, the final nonvolatile memories at the end of the JIT block execution will store the same values.

The proof is by induction on the number of attempts $m + 1$ until the first configuration succeeds. Fig. 3.23 shows the inductive case (lines (1)-(6)) and the base case (lines (7)-(9)). Starting with index $m+1$ (point (1) in Fig. 3.23), we show that the intermediate configurations continue to relate at a smaller index m. Then, we apply the inductive hypothesis to obtain the desired result. Finally, in the base case, we show that the memories of the two final configurations are the same.

Inductive Case (Fig. 3.23, lines (1)-(6)). By definition of the term relation, c_1 in the first configuration is executed via D-STEP until the next power failure occurs, which is detected by a 0 energy level. Moreover, by the term relation at type $\mathsf{C}_{\mathtt{unit}}$, we can execute c_2 in the second configuration to c_2' such that the resulting configurations remain related by the value relation at type $\mathsf{C}_{\mathtt{unit}}$ (point (2) in Fig. 3.23).

The orange box around lines (3)-(5) highlights the steps taken when handling a power failure. Each step corresponds to one of the key operations crash, power off, restore, and re-execute. Since power failures are incurred by intermittent executions, we step the first configuration according to the corresponding rules in the operational semantics, but do not step the second configuration through this sequence. Here, the first configuration takes a step from point (2) to point (3) using the D-CRASH rule. By the definition of the logical relation, the two configurations continue to be related by the value interpretation at type $\downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})$. Then, the first configuration takes a step from point (3) to point (4) by the D-S-JIT rule. In this case, we know (by the assumptions of the rule) that $\mathsf{V}' = \mathsf{V}_1'$ and $\gamma_1'' = \gamma_1'$. This matches the definition of the power-off policy for JIT blocks. By the value interpretation at type $\downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})$, the two configurations remain related by the value relation at type $\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}$. Next, the first configuration takes a step to point (5) by the rule D-CHARGE which inputs a new energy level (n_0) from the environment and updates the energy stream from χ_1 to χ_1' where $\chi_1 = n_0 :: \chi_1'$. By the definition of the value relation, the two configurations remain related by the value interpretation at type $\uparrow \mathsf{C}_{\mathtt{unit}}$.

Finally, the configuration steps to point (6) by D-RESTORE-JIT which copies all checkpointed locations in the nonvolatile memory into volatile memory and continues by running the interrupted command c_1'. That is, $\mathsf{V}_0 = \mathsf{V}'$ and $c_0 = c_1'$, and the nonvolatile memory and variable mapping remain the same ($\mathsf{N}_0 = \mathsf{N}_1'$ and $\gamma_0 = \gamma_1''$). This matches the restore policy defined for JIT regions. Thus, the configurations continue to be related by the *term relation* at type $\mathsf{C}_{\mathtt{unit}}$, similar to what we had earlier at point (1) in Fig. 3.23, but with fewer power failures remaining. At this point, we apply the inductive hypothesis (highlighted by a green box in Fig. 3.23) to build the continuous execution.

Base Case (Fig. 3.23, lines (7)-(9)). The red box in Fig. 3.23 highlights the base case. Since the index is 1, there are 0 remaining power failures. So, $n_k' > 0$ and the first configuration steps to completion via the D-STEP rule. By the definition of the value interpretation, we know that the second configuration also steps to skip indicating a complete execution. This completes the construction of the continuous execution for c_1. The volatile memory V_j' is dropped, and the

mapping is reset to γ. This matches the commit policy defined for JIT blocks.

Note that the value interpretation at type $\downarrow\uparrow\texttt{unit}$ establishes that the two configurations remain related at the type $\uparrow\texttt{unit}$ (line (9)). The commit policy drops the volatile memories V'_k and V'_j such that $\mathsf{V}''_k = \mathsf{V}''_j = \emptyset$. Finally, by the value interpretation at type $\uparrow\texttt{unit}$, we get that the final nonvolatile memories are the same $(\mathsf{N}'_j, \mathsf{V}''_j = \mathsf{N}'_k, \mathsf{V}''_k)$ which completes the proof.

In the case where the initial configurations are the same, we have that $c_2 = c_1$, $\mathsf{N}_2 = \mathsf{N}_1$, and $\mathsf{V}_2 = \mathsf{V}_1$. Here, we can apply the D-P-SEQ rule to the construction to obtain a continuous execution for the program $c_1; p$ as desired: $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1 \mid c_1; p \Rightarrow_p [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid p$.

Chapter 4

More Efficient Checkpointing Policies

The crash types formulation, by nature of being parameterized by policies defining the beahviors of key operations, is also capable of reasoning about systems that make use of more flexible checkpointing policies. This chapter investigates an extension to a more efficient checkpointing policy that is used by real intermittent systems [77, 82, 119]. We highlight key changes from Chapter 3 in yellow.

4.1 Write-After-Read Patterns

While checkpointing all written variables, as done in the previous system, is simple and relatively straightforward to reason about, this strategy results in unnecessary memory and runtime overheads to track and restore state, rendering it unsuitable for the extremely resource constrained nature of the domain. To reduce overhead from checkpointing, we must determine which variables in a program must be checkpointed while still allowing the program to compute correct results on all executions. For this we recall the example atomic region from Chapter 2. If no variables are checkpointed and power fails after the assignment $y := y + z$, the value of y will not be restored, allowing the next execution to read the unstable value of y. Therefore, y must be checkpointed. On the other hand, other writable variables, e.g., u, are first written before they are read, thereby overwriting any unstable values from failed prior executions before they can be used on the next one. Therefore, variables like u need not be checkpointed, only y.

It happens that an entire class of memory consistency bug in intermittent computing is attributed to *write-after-read (WAR)* dependencies such as $y := y + z$ if a *WAR variable*, e.g., y, is not checkpointed [77, 125]. Prior work shows that checkpointing only variables involved in problematic code patterns, such as WAR, is sufficient to guarantee that any intermittent execution of a sequential program corresponds to a continuously-powered execution of that program [119][1].

For the example atomic region, it therefore suffices to checkpoint only the WAR variable, e.g., y, to produce the same correct result as an execution that checkpoints all written variables. We sketch the execution of the atomic region that checkpoints only WAR variables in Fig. 4.1, again using redo-logging. Before executing an atomic region, the runtime system checkpoints *only* y, creating a volatile copy of it in ℓ_y (row (1)). Upon reboot from power failure, the checkpointed

[1] at least for programs that do not depend on sensor inputs

value stored in ℓ_y^{ck} recreates the log ℓ_y (row (7)), so that the assignment to y re-executes on its initial value (row (8)), correctly computing 1, and resulting in the same correct final state as before in row (11): $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 1, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow \mathtt{tt}$.

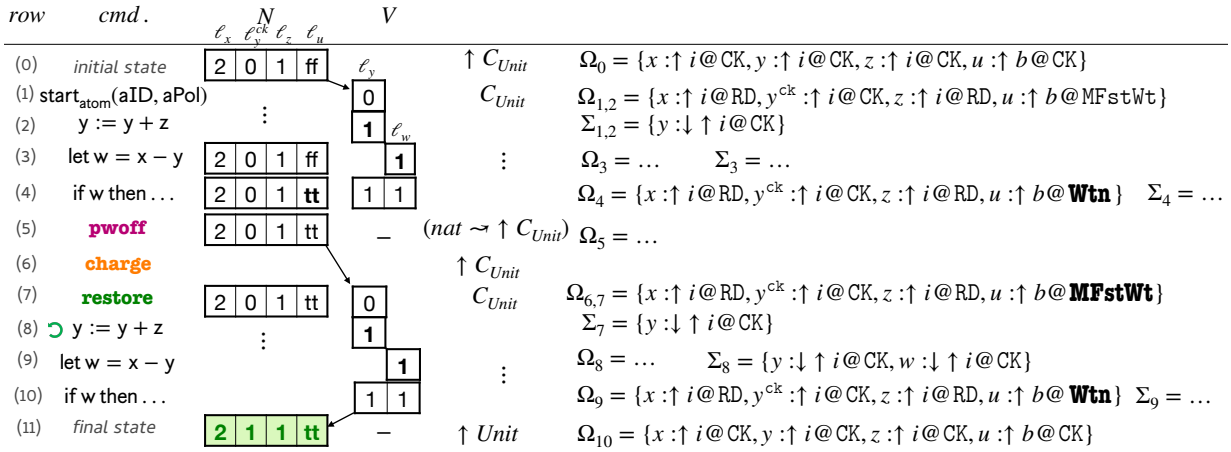

Figure 4.1: Example atomic region execution with only WAR variables checkpointed.

With the intuitions behind this new checkpointing policy in hand, we are now ready to explore an extension of the crash types calculus that supports more flexible checkpointing policies – programs that checkpoint at least their WAR variables. This chapter focuses on formulating atomic regions which are primarily impacted by this shift in policy due to their re-execution. Since JIT regions do not re-execute, they are not susceptible to memory consistency bugs, hence leaving their formulation unchanged from the previous chapter. We relegate the full system definition and proofs for both JIT and atomic regions to Appendix B.

4.2 Syntax and Operational Semantics

We highlight key modifications needed to the language syntax below.

$$\begin{array}{llll} \textit{atom. reg. pols.} & \textit{aPol} & ::= & (\rho, \mu, \omega) \\ \textit{access qualifier} & q & ::= & \mathtt{CK} \mid \mathtt{RD} \mid \mathtt{MFstWt} \mid \mathtt{Wtn} \end{array}$$

According to the new checkpointing policy, since only the write-after-read variables must be checkpointed (i.e., have qualifier CK), the remaining variables accessed within an atomic region are either read-only (i.e., have qualifier RD) or first accessed as a write. To handle the latter, we extend atomic region policies with a set of *must-first-write variables* (μ) that is disjoint from ω and ρ, and the type system with new access qualifiers: MFstWt and Wtn. Variables declared in μ are first annotated with the qualifier MFstWt, to be updated to Wtn on their first write in a given execution. This operation is performed by a modified version of the $\mathsf{InitWorld}_d$ function and illustrated in the step from row (0) to row (1) in Fig. 4.1. The modified function is formally defined in Appendix B.1. To guarantee the correctness of an atomic region with must-first-write variables, all must-first-write variables must be marked Wtn by the end of an atomic region's

final execution, indicating that only results from the last successful execution have survived in the final state and that results from failed prior executions have been overwritten.

Furthermore, these access qualifiers are used to reject programs susceptible to write-after-read bugs, which we discuss next.

Evolving Access Qualifiers. To enforce proper memory accesses in an atomic region, we define a function δ over the access qualifiers and possible memory accesses *write* (*Wt*) and *read* (*Rd*). This function is defined in Fig. 4.2. We write *UN* to represent the result of an undefined or prohibited action that would cause type checking to fail, e.g., writing (*Wt*) to a read-only (`RD`) location or reading (*Rd*) from a must-first-write (`MFstWt`) location that has not yet been written on the current execution. Furthermore, these checks should fail because they could point to uncheckpointed WAR variables. All other memory accessed are considered safe and are allowed by our system. We define transitions on these qualifiers. In particular, writing (*Wt*) to a must-first-write location causes the qualifier `MFstWt` to change to `Wtn`. Since any following write or read accesses on `Wtn` locations are allowed, the type qualifier does not evolve further and remains `Wtn`. Similarly, write or read accesses of checkpointed locations (`CK`) and read accesses of read-only locations (`RD`) do not change the qualifier.

$$\delta(\texttt{CK}, Rd) = \delta(\texttt{CK}, Wt) = \texttt{CK} \qquad \delta(\texttt{MFstWt}, Wt) = \texttt{Wtn} \qquad \delta(\texttt{MFstWt}, Rd) = UN$$
$$\delta(\texttt{RD}, Rd) = \texttt{RD} \qquad \delta(\texttt{RD}, Wt) = UN \qquad \delta(\texttt{Wtn}, Wt) = \delta(\texttt{Wtn}, Rd) = \texttt{Wtn}$$

Figure 4.2: Definition of the memory access transition function δ

We are now ready to define the operational semantics for our calculus. The full operational semantics is defined in Appendix B.1, and we highlight the key rules that change from the previous formulation in Fig. 4.3.

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v \qquad q' = \delta(q, Rd) \neq UN \qquad n = n' + 1}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N}', \ell@\,q' \hookrightarrow v \mid \mathsf{V} \mid v} \text{ (D-N-READ)}$$

$$\frac{\begin{array}{c} Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid v) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v_0 \qquad q' = \delta(q, Wt) \neq UN \qquad n = n' + 1 \end{array}}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\to\; \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N}', \ell@\,q' \hookrightarrow v \mid \mathsf{V} \mid \mathsf{skip}} \text{ (D-N-ASSIGN)}$$

$$\frac{\mathsf{N} = \mathsf{N}'_{\texttt{RD},\texttt{MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}'''_{\texttt{Wtn}} \qquad \mathsf{N}_2 = \mathsf{N}'_{\texttt{RD},\texttt{MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}'''_{\texttt{MFstWt}}}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N} \mid \uparrow\kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_2 \mid \mathsf{N}'' \mid c_0} \text{ (D-RESTORE-AID)}$$

Figure 4.3: Selected operational semantic rules for atomic regions.

Reads and Writes (D-N-READ and D-N-ASSIGN). Now when a variable is accessed, we must check that the access is legal. The operational semantic rules for read and write therefore make use of δ to guard against illegal accesses.

When a location is read, the rule D-N-Read applies to look up the value v stored in a location ℓ corresponding to variable x. The premise $q' = \delta(q, Rd) \neq UN$ first checks that reading the desired location is allowed by δ, effectively checking that the location being read is not marked as `MFstWt`. If the read is allowed, the function computes the resulting qualifier q' from the original q and updates the access qualifier in nonvolatile memory accordingly.

When a location is begin written to, the rule D-Assign-N applies to check that the write is legal under δ, meaning that the location being written is not read-only (`RD`). If the access is allowed, a new qualifier q' is computed to annotate the new value v in nonvolatile memory.

Restore (D-restore-aID). Since a program could crash in a state with a combination of `MFstWt` and `Wtn` qualifiers, the system behavior on restore (D-restore-aID) changes. Like before, the volatile logs are recreated from their checkpointed nonvolatile counterparts. However, since nonvolatile memory could contain uncheckpointed writes from the previous failed execution, which are marked as `Wtn`, the rule reverts these qualifiers to `MFstWt`. The change in qualifier indicates that these variables have not yet been written on the current re-execution, which will help to prevent any illegal reads to these locations on the next execution.

4.3 Type System

At the type level, qualifiers are also extended to include `MFstWt` and `Wtn` to denote the access restrictions on their locations. Since qualifiers evolve over the course of execution, they also evolve throughout type checking which our typing judgments must be equipped to support.

We summarize typing judgments for commands and selected expressions in Table 4.1 that differ from the previous chapter. To support evolving qualifiers, certain typing judgments are extended with a post-context Ω' whose domain always matches the initial context Ω and range reflects qualifier changes from Ω that result from type checking a command. Since updating a qualifier should only occur when checking a write, we must extend `WT` typings judgments for expressions to carry a post-context. Since these changes propagate to assignments, and subsequently later commands, it is carried by the command typing judgments too.

command	(J_u)	$\texttt{Md} \mid b\,\mathcal{R}\,0 : \texttt{nat} \mid \Omega;\Sigma$	$\vdash_{\texttt{Sig}} c$	$: \mathsf{C}_{\texttt{unit}}$	$\dashv \Omega'$	c could crash
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega;\Sigma$	$\vdash_{\texttt{Sig}} \mathsf{skip}$	$: \downarrow\uparrow\texttt{unit}$	$\dashv \Omega'$	c will not crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega$	$\vdash_{\texttt{Sig}} \mathsf{skip}$	$: \uparrow\texttt{unit}$	$\dashv \Omega'$	after commit
expression	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega;\Sigma$	$\vdash_{\texttt{WT}} x$	$: \downarrow\uparrow A$	$\dashv \Omega'$	write on x, no crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega$	$\vdash_{\texttt{WT}} x$	$: \uparrow A$	$\dashv \Omega'$	write on x, commit

Table 4.1: Typing judgments for commands and `WT` expressions

Now we are ready to discuss the typing rules that change with this formulation. The rules for statement and store typing, which remain the same as Chapter 3, are relegated entirely to Appendix B.2.

4.3.1 Program Typing

The typing rule for atomic regions is shown below.

$$\frac{\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0 \qquad \mathtt{Sig} = \{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'\} \quad \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0;\ \Sigma_0 \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \quad \Omega' \restriction \{\mathtt{MFstWt}\} = \emptyset \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-Ckpt)}$$

The first premise sets up the initial typing contexts for nonvolatile and volatile memories. As before, the function $\mathsf{InitWorld}_t$ initializes the unstable typing context Σ_0 by creating a log of variables based on the typings for checkpointed variables from Ω. Definition 6 formally states this policy, which is extended to handle must-first-write variables.

Definition 6 $\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; aPol)$ *where* $aPol = (\rho, \mu, \omega)$ *iff*

- $\mathsf{dom}(\Omega) = \rho \cup \mu \cup \omega$,
- $\rho \cap \mu = \emptyset, \mu \cap \omega = \emptyset, \rho \cap \omega = \emptyset$,
- $\Omega_0 = \{x : \uparrow A\,@\,\mathtt{RD} \mid x : \uparrow A\,@\,q \in \Omega^r\} \cup \{x : \uparrow A\,@\,\mathtt{MFstWt} \mid x : \uparrow A\,@\,q \in \Omega^m\} \cup \Omega^c_{\mathsf{ck}}$ *and*
- $\Sigma_0 = \downarrow \Omega^c$

where $\Omega = \Omega^r, \Omega^m, \Omega^c$ *and* $\Omega^r = \Omega \restriction \rho$, $\Omega^m = \Omega \restriction \mu$, *and* $\Omega^c = \Omega \restriction \omega$.

Like before, if the declared sets of read-only, must-first-write, and checkpointed variables do not cover all nonvolatile variables or are not pairwise disjoint, then the function $\mathsf{InitWorld}_t$ is not defined and type-checking fails.

Using the typing contexts constructed by $\mathsf{InitWorld}_t$, the rule T-P-Ckpt proceeds to check the atomic region command c_0. In this version of the rule, the command typing judgment contains a post-context Ω' with updated qualifiers resulting from checking c_0. The rule uses Ω' to check that the atomic region is correct: that all must-first-write variables (`MFstWt`) are written to on the last successful execution of the region. Provided the atomic region is correct, the last premise proceeds to check the remainder of the program p with the original typing context Ω.

4.3.2 Command and Expression Typing

We present selected typing rules for commands and expressions in Fig. 4.4.

Successful Execution (T-∨-Succ, T-C-Shift, and T-∨-Succ). The T-Skip rule types the command skip at a stable type $\uparrow\mathtt{unit}$. At this point, the command execution is complete and the initial stable typing context Ω should match the final stable typing context. Rule T-∨-Succ applies when the command successfully completes its execution and still has at least one unit of energy available ($b > 0$) to conclude the execution by committing. In this case, we close off the constraint on the energy level variable and continue typing the command against the type $\downarrow\uparrow \mathtt{unit}$. Rule T-C-Shift is invoked by T-∨-Succ and updates the memory typing contexts by removing checkpointed locations in Ω as now they are not needed, and making locations in Σ stable as now they are committed. This corresponds to the last step of Fig. 4.1. The post-context $\Omega_1 \restriction dom(\Omega)$ restricts the post-context to include only variables appearing in Ω.

Reads, Writes, Sequences (T-Loc-Read, T-Loc-Write, T-W-Shift, T-Assign, T-seq, T-seq-d). To check an assignment to a variable, the rule T-Assign applies to check the variable x under

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega} \text{(T-Skip)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \lceil dom(\Omega)} \text{(T-C-Shift)}$$

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \vee \downarrow\uparrow\mathtt{unit} \dashv \Omega'} \text{(T-}\vee\text{-Succ)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{A} \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'} \text{(T-Assign)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \tau \dashv \Omega'} \text{(T-If)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''} \text{(T-seq)}$$

$$\frac{\begin{array}{c}W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \\ \Sigma' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''} \text{(T-seq-d)}$$

$$\frac{\Omega, \Sigma' = x{:}\uparrow A@q, \Omega'_2 \qquad q' = \delta(q, Wt) \neq UN}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A \dashv x{:}\uparrow A@q', \Omega'_2} \text{(T-Loc-Write)}$$

$$\frac{\Sigma = \downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A \dashv \Omega_1}{\mathtt{Md} \mid\ b : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega_1 \lceil dom(\Omega)} \text{(T-W-Shift)}$$

$$\frac{\Omega(x) = \uparrow A@q \qquad \delta(q, Rd) \neq UN}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} x : \uparrow A} \text{(T-Loc-Read)}$$

Figure 4.4: Selected command and expression typing rules

a WT judgment. As before, the rule T-W-Shift applies to check the variable under an unstable judgment, and then T-Loc-Write applies to check it under a stable judgment. Notably, the T-Loc-Write rule checks if the write is valid, i.e., that the access qualifier q' resulting from writing (Wt) to the variable x is not undefined. If the write is valid, the post-context produced by this rule is based on the initial stable typing context, updated to reflect the write by replacing the access

qualifier of x with q'. Otherwise, if the write is invalid, the program would be rejected. For example, writing to a read-only variable would cause type checking to fail at this point. If the check succeeds, a post-context with the updated qualifier is obtained, which propagates down the derivation to T-W-Shift where the post-context $\Omega_1 \restriction dom(\Omega)$ (similar to T-C-Shift) enforces that the pre- and post-contexts of a judgment always match on their domains and differ only in their qualifiers. Finally, the post-context propagates to T-Assign. If there is a following command, the rules T-seq and T-seq-d would apply to pass the post-context into the initial context of checking the next command.

To check a variable read, the rule T-Loc-Read performs an analogous check, looking up the qualifier of x and using it to check whether a read is valid: $\delta(q, Rd) \neq UN$. Since the qualifiers that are allowed to be read are `RD`, `CK`, `Wtn` which do not change on read, the rule does not include a post-context.

For the version of the example atomic region which fails to checkpoint y in $y := y + z$, type checking would fail by either T-Loc-Write or T-Loc-Read, depending on the qualifier of y. For example, if y has qualifier `RD`, then its write would fail to type check by the rule T-Loc-Write, as writing to a read-only variable is undefined. Alternatively, if the initial qualifier of y is `MFstWt`, its read would fail to type check by the rule T-Loc-Read, as reading a must-first-write variable is also undefined.

Conditionals. In the rule T-If, the first premise checks that expression e has type $\mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}}$, and the second and third premises type the commands c_1 and c_2. Both branches should end with memories whose variables have the same qualifiers. Programs that have branching that causes variables to have different qualifiers are rejected by the type system, e.g., a program that writes to a `MFstWt` variable in one branch but not the other. In this scenario, the qualifiers in the post-context of each branch will be inconsistent. Even if that variable is written immediately after the branch, our type system will reject this program. This is one limitation of our type system.

4.4 Logical Relation

We prove our system correct according to a more general version of the logical relation from Chapter 3. In particular, our logical relation must be equipped to handle values leftover from prior executions that may momentarily differ from those generated by a continuously-powered execution. Therefore, in order to establish that the final state of an intermittent execution matches a continuously-powered execution, we must again show that the intermediary states of an intermittent execution are related to a state in the continuously-powered execution despite the fact that now an atomic region could re-execute on memory that does not match the corresponding continuously-powered execution state.

Policies for key operations (`Commit`, `Restore`, and `PwOff`). Before explaining details of our logical relation, we define new policies used in its definition. We formally define these policies in Fig. 4.5. The definitions match the memory operations in the dynamic rules that deal with crash, restore, and re-execution (D-S-aID, D-restore-aID, and D-P-Ckpt) for atomic regions.

As before, the power off policy is defined to lose the volatile memory state upon power failure and the commit policy finalizes the nonvolatile memory after a successful atomic region

$$\frac{\gamma' \subseteq \gamma \; s.t. \; range(\gamma') = \mathsf{dom}(\mathsf{N}_1) \qquad \mathsf{V}_2 = \emptyset}{\texttt{PwOff}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2}$$

$$\frac{\mathsf{N}_2 = \mathsf{N}'_{1\,\mathtt{ck}}, \mathsf{V}'' \qquad \mathsf{V}_1 = \mathsf{V}'_1, \mathsf{V}'' \qquad \begin{array}{c}\mathsf{N}_1 = \mathsf{N}'_{1\,\mathtt{RD},\mathtt{Wtn}}, \mathsf{N}''_{\mathtt{ck}} \\ \mathsf{dom}(\mathsf{V}'') = \mathsf{dom}(\mathsf{N}'')\end{array} \qquad \gamma' \subseteq \gamma \; s.t. \; range(\gamma') = \mathsf{dom}(\mathsf{N}_1)}{\texttt{Commit}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{N}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{\mathtt{RD},\mathtt{MFstWt}}, \mathsf{N}''_{\mathtt{ck}}, \mathsf{N}^3_{\mathtt{Wtn}} \qquad \mathsf{N}_2 = \mathsf{N}'_{\mathtt{RD},\mathtt{MFstWt}}, \mathsf{N}''_{\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} \qquad \mathsf{V}_2 = \mathsf{N}''}{\texttt{Restore}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \kappa) = \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_0}$$

Figure 4.5: `Commit`, `Restore`, and `PwOff` policy definitions for atomic regions

execution. The restore policy describes setting up the memories for re-executing an atomic region though, unlike the previous chapter, it marks all locations with the qualifier `Wtn` with `MFstWt`.

Logical relation and semantic typing. Now we are ready to explain our logical relation. First, the top-level logical relation for atomic regions remains the same as before. The term and value relations, shown in Fig. 4.6, are similar to the previous chapter but with a slight modification to accommodate our more general policies.

In particular, the value relation at type $\uparrow\mathsf{C}_{\mathtt{unit}}$ changes to accommodate changes made to the `Restore` policy. This definition again requires that the intermittent configuration runs the crash instruction $\uparrow\kappa$, which restores a continuation of the execution and continues to be related to the continuously-powered configuration in the term interpretation at its original type $\mathsf{C}_{\mathtt{unit}}$. To ensure that the program can re-execute on a state with values written by failed prior executions that will not be reverted on reboot, we extend this line of the relation with a side condition. In particular, the constraint $\mathsf{N}_0 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$ requires that the intermittent memory is the same as the continuously-powered one, modulo must-first-writes. This constraint is necessary because after a partial execution of an atomic region, memory may be updated in its written variables. This means that the memory of the intermittent configuration may differ from the continuously-powered configuration in those locations with the qualifier `Wtn`. The restore policy for atomic regions resets the qualifier `Wtn` of these locations to `MFstWt`. At this point, the values of locations marked `MFstWt` may differ between the intermittent and continuously-powered configurations. Such differences, however, do not violate the idempotency of the intermittent execution; in the next execution, these locations will not be read from before being overwritten, and if the next execution succeeds, these locations will be overwritten. This condition makes the logical relation more general and capable of accommodating more correct programs.

4.5 Metatheory

This section establishes the main properties of our extended system. At a high level, these properties are similar to those established in Chapter 3 but require extra reasoning about qualifiers.

Binary relation for commands

$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega \mid \Sigma \Vdash c_1 \leq c_2 : \mathsf{C}_{\mathtt{unit}}$
iff $\forall n, m \geq 0.\ \forall \gamma, \mathsf{N}, \mathsf{V}. s.t.\ \vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma.$
$(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1, \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$

Term relation

$$\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1} = \{(C_{o1}, C^{\infty}_{o2}) \mid \exists C'_{o1}\ s.t. C_{o1} \rightarrow^*_{irred} C'_{o1} \wedge \exists C^{\infty'}_{o2}\ s.t. C^{\infty}_{o2} \rightarrow^* C^{\infty'}_{o2} \wedge (C'_{o1}, C^{\infty'}_{o2}) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}\}$$

$$\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{0} = \{(C_{o1}, C^{\infty}_{o2})\}$$

Value relation

$$\mathcal{V}[\![\uparrow\mathtt{unit}]\!] = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma \mid \mathtt{Md} \mid n_1 \mid \mathsf{N} \mid \mathsf{skip}) \wedge C^{\infty}_{o2} = (\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{skip})\}$$

$$\mathcal{V}[\![\downarrow\uparrow\mathtt{unit}]\!] = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \mathsf{skip}) \wedge C^{\infty}_{o2} = (\gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid \mathsf{skip}) \wedge \mathtt{Commit}(\gamma_i, \mathtt{Md}, \mathsf{N}_i, \mathsf{V}_i) = \gamma'_i \mid \mathsf{N}'_i \wedge (\gamma'_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}'_1 \mid \mathsf{skip}, \gamma'_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow\mathtt{unit}]\!]\}$$

$$\mathcal{V}[\![\uparrow\mathsf{C}_{\mathtt{unit}}]\!]^m = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \uparrow\kappa) \wedge C^{\infty}_{o2} = (\gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \wedge \mathtt{Restore}(\gamma_1, \mathtt{Md}, \mathsf{N}_1, \kappa) = \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \wedge \mathsf{N}_0 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\} \wedge (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m\}$$

$$\mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}]\!]^m = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa)) \wedge \forall n{>}0. (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \uparrow\kappa, C^{\infty}_{o2}) \in \mathcal{V}[\![\uparrow\ \mathsf{C}_{\mathtt{unit}}]\!]^m\}$$

$$\mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}})]\!]^m = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa)) \wedge \mathtt{PwOff}(\gamma_1, \mathtt{Md}, \mathsf{N}_1, \mathsf{V}_1) = \gamma'_1 \mid \mathsf{V}' \wedge (\gamma'_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{V}'_{\mathsf{ck}}, \mathsf{N}_1 \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow\kappa), C^{\infty}_{o2}) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}]\!]^m\}$$

$$\mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1} = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1) \wedge \textit{either}\ n_1 = 0 \wedge (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow c_1), C^{\infty}_{o2}) \in \mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\ \mathsf{C}_{\mathtt{unit}})]\!]^m, \textit{or}\quad n_1 > 0 \wedge (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, C^{\infty}_{o2}) \in \mathcal{V}[\![\downarrow\uparrow\ \mathtt{unit}]\!]\}$$

Figure 4.6: Step-indexed logical relation for crash types

These theorems and their proofs are provided in Appendix B.

4.5.1 Statically Well-Typed Programs are Type Safe

We again prove type safety according to progress and preservation, where the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ is the same as before. The progress theorem is similar to before, stating that if a command c is well-typed, then for all energy levels n and well-formed memories N and V, either the configuration of c is a value configuration or it can take a step according to the operational semantics.

The preservation theorem for commands, however, is modified with additional invariants to reason about qualfiiers. It is formally defined below.

Theorem 5 (Preservation for Commands) *If* $\mathtt{Md} \mid b \geq 0 : nat \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$ *where*

(1) $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed,*

(2) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*,*

(3) natural number $n \geq 0$*, and*

(4) $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$

then for some Ω_0, Σ', $\mathtt{Md} \mid b \geq 0 : nat \mid \Omega_0; \Sigma' \vdash_{\mathtt{Sig}} c' : \tau \dashv \Omega'$ *where*

(5) $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *is well-formed,*

(6) $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega_0 \mid \Sigma'$*,*

(7) natural number $n' \geq 0$*,*

(8) $dom(\mathsf{N}) = dom(\mathsf{N}')$*,*

(9) if $\mathtt{Md} = \mathsf{aID}(c_0)$*, then* $\mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$*, and*

(10) $\Omega \preccurlyeq_{Wt} \Omega_0$*.*

Invariants (1) – (8) remain the same as before, while (9) and (10) are new. For the former, since an arbitrary command step could write to memory, we must also prove that the initial and resulting memories from running that command are the same excluding their must-first-write and written memory locations for the mode $\mathsf{aID}(c_0)$. The latter condition enforces that the qualifiers of the variables in contexts Ω_0 and Ω are within one δ transition of each other and is defined below in Definition 7.

Definition 7 $\Omega \preccurlyeq_{Wt} \Omega_0$ *iff* $\forall x{:}\tau@q \in \Omega$*, we have* $x{:}\tau@q_0 \in \Omega_0$ *where either* $q = q_0$ *or* $\delta(q, Wt) = q_0 \neq UN$*.*

The full proofs and theorem statements of progress and preservation can be found in Appendix B.4.

4.5.2 Statically Well-Typed Programs are Semantically Well-Typed

We prove the soundness of our static typing rules with regard to the semantic typing rules (Theorem 6). We sketch the proof below, focusing on atomic regions. We provide the full proof in Appendix B.6.

Theorem 6 (Fundamental Theorem) *If* $b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}$*, then* $b : \mathtt{nat} \mid \Omega \Vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}$*.*

The structure of the proof of Theorem 6 is similar to before: by induction on the static typing derivation for p and considers the last step in the derivation. Here, we highlight the key differences to the proof case of D-P-Ckpt.

Again, the goal is to establish that c is related to itself in the term interpretation for arbitrary n, m, γ, N and V where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega'', \Sigma_{\mathtt{ck}} \mid \Sigma$. This means that we need to prove $(\gamma \mid \mathsf{aID}(c) \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c, \gamma \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$ for all indices m given that the condition $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ holds. The last condition enforces that the static contexts match the dynamic context. In particular, $\mathsf{N} = \mathsf{N}', \mathsf{V}_{\mathtt{ck}}$ and $range(\gamma) = dom(\mathsf{N})$.

Since now nonvolatile memories could differ at intermediate stages of the simulation, we must prove a generalized version of this goal in the main proof of Theorem 6. In particular, we prove that a well-typed command c is related to itself for any two nonvolatile memories N_1 and N_2 that agree on every value of their locations *except for those with must-first-write qualifiers*. The goal is to show $(\gamma \mid \mathsf{aID}(c) \mid n \mid \mathsf{N}_1 \mid \mathsf{V} \mid c, \gamma \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$

where (a) $\mathit{range}(\gamma) = \mathit{dom}(\mathsf{N}_1)$, (b) $\mathsf{N}_1 \setminus \texttt{MFstWt} = \mathsf{N}_2 \setminus \texttt{MFstWt}$, and (c) $\mathsf{N}_1 = \mathsf{N}_1 \setminus \texttt{Wtn}$. The condition (a) asserts that γ covers the locations in the nonvolatile memory N_1. The condition (b) enforces that the nonvolatile memories N_1 and N_2 be related up to their `MFstWt` qualifiers. Lastly, the condition (c) asserts that the intermittent nonvolatile memory N_1 does not have any `Wtn` variables at the beginning of executing an atomic region. We discuss how to establish this relation in the inductive case explanation below.

First, we observe that command c being self-related is a special case of this generalized goal where the nonvolatile memories are the same ($\mathsf{N}_1 = \mathsf{N}_2 = \mathsf{N}$) and are well-formed with respect to the context initialized by inversion on the D-P-Ckpt rule. Observe that the three conditions required by the inductive hypothesis hold in this special case: (a) $\mathit{range}(\gamma) = \mathit{dom}(\mathsf{N})$ holds as described above, (b) $\mathsf{N} \setminus \texttt{MFstWt} = \mathsf{N} \setminus \texttt{MFstWt}$ holds vacuously, and (c) $\mathsf{N}_1 = \mathsf{N}_1 \setminus \texttt{Wtn}$ is true due to the definitions of InitWorld and well-formedness.

We proceed to prove the generalized result mentioned above by induction on the index m.

Base case ($m = 0$). This case is similar to before; the result follows immediately from the term interpretation at type $\mathsf{C}_{\texttt{unit}}$.

Inductive case ($m = k + 1$). We now highlight the key differences to the inductive case which is sketched in Fig. 4.7. The green conditions (d)-(g) are learned throughout the proof, and (a)-(c) are known by assumption. The places where these conditions are used in the proof are marked on the right hand side of the diagram. As before, the dashed lines represent implications.

Our goal is to prove $(\gamma \mid \mathsf{aID}(c) \mid n \mid \mathsf{N}_1 \mid \mathsf{V} \mid c, \gamma \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+1}$ which corresponds to point (1) in Fig. 4.7 with assumptions (a)-(c). By progress and preservation for commands, the first configuration can take multiple steps until it becomes a value $\gamma_1 \mid \mathsf{aID}(c) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1$ that continues to be well-typed. If $n' > 0$, the second configuration steps similarly to completion and establishes that the two resulting configurations are in the value relation. This case is omitted from Fig. 4.7. If $n' = 0$, the second configuration does not step and instead reaches point (2) in Fig. 4.7. At point (2), the proof must show that the configurations are in the value interpretation at type $\mathsf{C}_{\texttt{unit}}$.

At point (4), we apply the power failure policy for atomic regions, which drops the volatile memory V' and creates a mapping using the domain of N'. By the prior conditions established, we know the created mapping is the original mapping γ.

At point (6), we apply the restore policy for atomic regions, which creates a new volatile memory based on N' and marks all written (`Wtn`) memory locations in N' as must-first-write (`MFstWt`). By the semantics of the restore policy, we know that the volatile memory created (V) is the volatile log containing only copies of checkpointed locations, and is the same as the original volatile memory.

The goal at point (6) is similar to our original goal at point (1) but at the smaller index k. We can apply the inductive hypothesis to establish the goal at point (6). To do so, it is enough to show that conditions (a)-(c) hold at this point. First, (a) holds by the original condition (a) combined with (f) and the definition of N' given by (g). The condition (b) holds by combining the original condition (b) with (c), (e), and the observation that N', sans the locations marked `MFstWt` and `Wtn`, is the same as the final intermittent nonvolatile memory, sans the locations

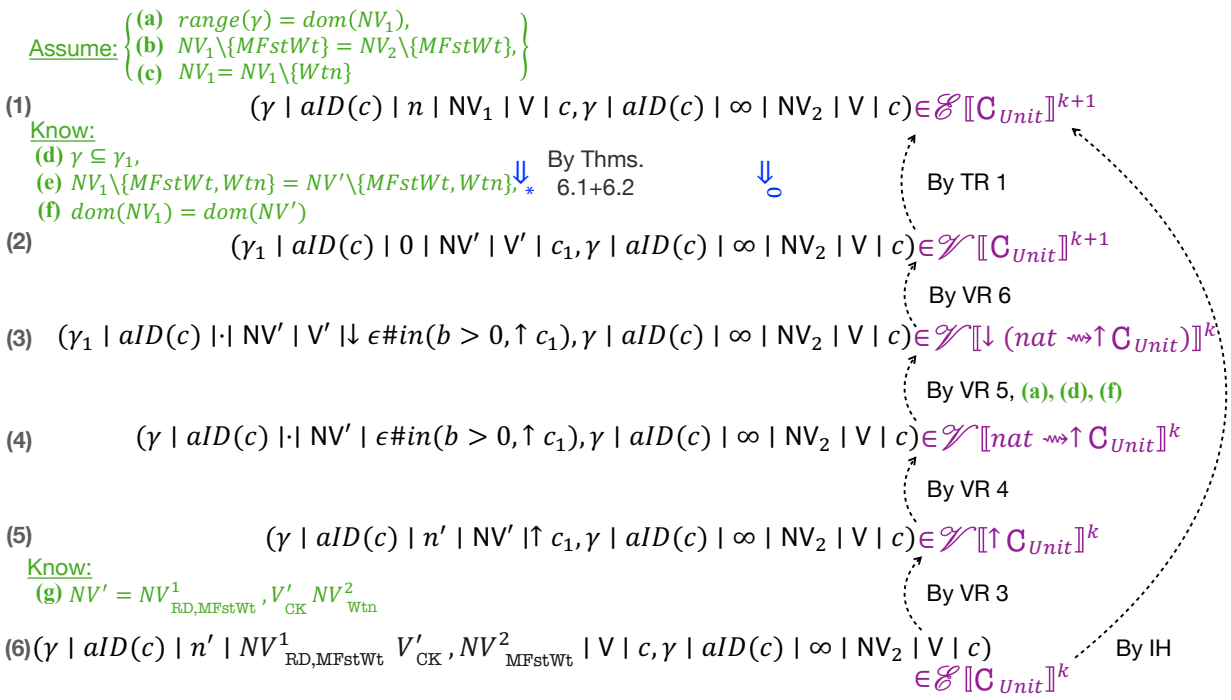

Figure 4.7: Proof of the fundamental theorem for D-P-Ckpt (inductive case)

marked `MFstWt`. (c) continues to be true for each subsequent execution due to the semantics of the restore function which replaces all `Wtn` qualifiers in nonvolatile memory with `MFstWt`.

4.5.3 Semantically Well-Typed Programs are Idempotent

Our definition of idempotency is the same as the previous chapter: a program is idempotent if every intermittent execution of it can be simulated by a continuously-powered execution. Since the semantic typing also remained the same, the adequacy theorem also remains the same as the previous chapter but requires a generalized proof to accomodate the modifications to the system.

We restate the adequacy theorem below and highlight key differences in the proof.

Theorem 7 (Adequacy) *Consider* $b : \mathtt{nat} \mid \Omega \Vdash p : \mathsf{C}_{\mathtt{unit}}$, *a nonvolatile memory* N, *and a map* γ *such that* $\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \cdot : \Omega \mid \cdot$. *The triple of* p, N, *and* γ *is idempotent.*

We include the full proof in Appendix B.7, and sketch the main ideas here. The structure of the proof is similar, building a simulation required by idempotency by induction on the number of power failures during each step of a program's intermittent execution. We show the construction of this simulation (and proof) in Fig. 4.8.

Assume that c_1 is self-related and that the configuration $[\chi_1 \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID} \mid n \mid \mathsf{N}_1 \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p$ takes a step to $[\chi_k \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid n' \mid \mathsf{N}' \mid p$ via the rule D-P-Ckpt while incurring possibly m-many power failures. We need to show that there exists a continuously-powered execution of the same program that steps the configuration $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID} \mid \infty \mid \mathsf{N}_1 \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p$ to $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid \infty \mid \mathsf{N}' \mid p$. Again, we prove this by building such a continuously-powered execution.

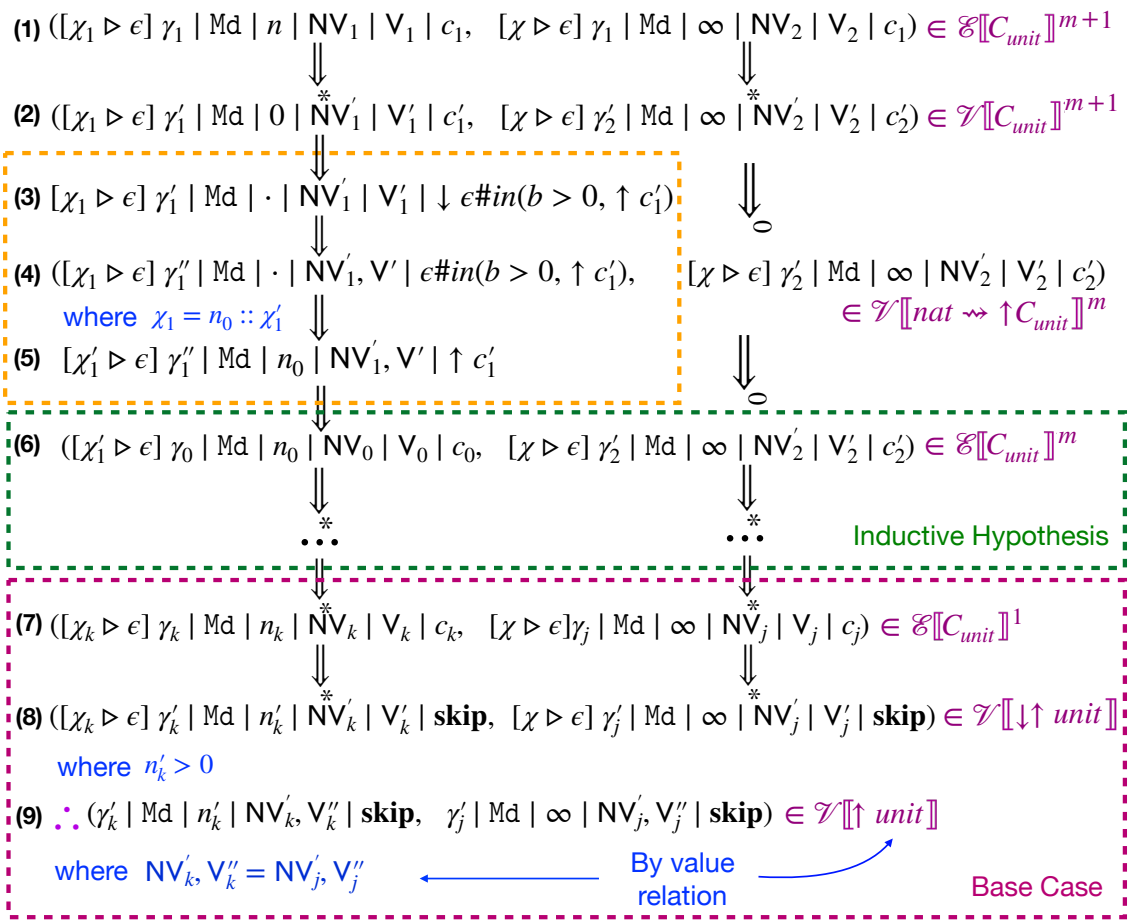

Figure 4.8: Adequacy proof sketch.

The proof assumes that program $\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p$ is semantically well-typed. Applying P-CKPT-SEMANTIC, we learn that the command c_1 is semantically well-typed and self-related at type $\mathsf{C}_{\mathtt{unit}}$. By the definition of logical relation (Fig. 4.6), it follows that the intermittent configuration of c_1 with energy level n is related to the continuously-powered configuration of c_1 with energy level ∞ at the type $\mathsf{C}_{\mathtt{unit}}$ for every index, including $m+1$. We use this relation to build the desired simulation. Towards this goal, we prove a more general result: for any two related configurations, if the first configuration takes zero or more steps, then the second configuration takes zero or more steps such that the two stepped configurations remain related. By definition of our relation, both configurations result in the same final nonvolatile memories.

The proof is by induction on the number of power failures m. Fig. 4.8 also shows the inductive case (lines (1)-(6)) and the base case (lines (7)-(9)) for the case of an atomic region. Starting with index $m+1$ (point (1) in Fig. 4.8), we build a continuously-powered execution for c_1 while showing that the intermediate configurations continue to relate to their intermittent counterparts at a smaller index m (point (6) in Fig. 4.8). At that point, we apply the inductive hypothesis to obtain the desired result. Finally, the base case establishes that the memories of the final configurations are the same. We detail each of these cases below.

Inductive Case (Fig. 4.8, lines (1)-(6)). By definition of the term interpretation at type $\mathtt{C}_{\mathtt{unit}}$, c_1 in the first configuration runs via D-STEP until the next power failure. Moreover, it follows by the term interpretation at type $\mathtt{C}_{\mathtt{unit}}$ that the second configuration can also be executed such that the resulting configurations remain related (point (2)). The first configuration takes a step from point (2) to point (3) with the rule D-CRASH. Since the energy level of the first configuration is 0, it follows by the value interpretation at type $\mathtt{C}_{\mathtt{unit}}$ that these two configurations are related by the value interpretation at type $\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathtt{C}_{\mathtt{unit}})$ (point (3)). Using the rule D-S-AID, the first configuration takes a step from point (3) to point (4). The volatile memory V_1' is dropped, and hence $\mathsf{V}' = \emptyset$. Note that this matches the power off policy for atomic mode. By assumption to D-S-AID, the mapping γ_1'' drops the discarded volatile memory locations from γ_1' such that $\gamma_1'' \subseteq \gamma_1'$. The two configurations remain related by the value interpretation at type $\mathtt{nat} \rightsquigarrow \uparrow\mathtt{C}_{\mathtt{unit}}$ (point (4)). The first configuration then takes another step to point (5) by D-CHARGE, inputting the harvested energy n_0, thus updating the energy stream to χ_1' where $\chi_1 = n_0 :: \chi_1'$. By definition of the value relation, the two configurations remain related at the type $\uparrow\mathtt{C}_{\mathtt{unit}}$ for all n_0 (point (5)).

From point (5) to point (6), the first configuration steps via D-RESTORE-AID. In doing so, it replaces all $\mathtt{Wtn}$ qualifiers in the nonvolatile memory $\mathsf{N}_1', \mathsf{V}'$ with $\mathtt{MFstWt}$ in N_0 and loads the checkpointed data of the nonvolatile memory into the volatile memory V_0. As such, the program counter resets to its original value, and the atomic region re-executes from the beginning such that $c_0 = c_1$. The variable mapping stays the same ($\gamma_0 = \gamma_1''$). Note that this matches the restore policy for atomic mode. The two configurations are then related by the term interpretation at the type $\mathtt{C}_{\mathtt{unit}}$. This is similar to what we had initially but with one fewer power failure remaining. At this point, we apply the inductive hypothesis to build the simulation at the index m.

Base Case (Fig. 4.8, lines (7)-(9)). When the first configuration finally steps to completion by the D-STEP rule (point (8) in Fig. 4.8), it follows by the definition of logical relation that the second configuration also steps to skip thus detecting a successfully completed execution. This completes the simulation of the continuously-powered execution of c_1. Here, we can apply the rule D-P-CKPT on this construction to obtain a continuously-powered execution of the program $\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p$. The $\mathsf{FinWorld}_d$ function copies the up-to-date values of volatile memory V_k' that have checkpointed locations back into nonvolatile memory N_k', removes the subscript $\mathtt{ck}$ from these locations, and changes the qualifiers of all locations in nonvolatile memory to $\mathtt{CK}$. The volatile memory is dropped and the mapping is reset to γ. This matches the commit policy defined for atomic mode. By the value interpretation at type $\downarrow\uparrow\mathtt{unit}$, the two configurations remain related at the type $\uparrow\mathtt{unit}$ (line (9)). Then, by the value interpretation at type $\uparrow\mathtt{unit}$, the final nonvolatile memories are the same ($\mathsf{N}_k', \mathsf{V}_k'' = \mathsf{N}_j', \mathsf{V}_j''$) which is the last piece we needed.

In the case of the initial execution, where $c_2 = c_1$, $\mathsf{N}_2 = \mathsf{N}_1$, and $\mathsf{V}_2 = \mathsf{V}_1$, we can apply the rule D-P-CKPT to the constructed execution to find a continuously-powered execution of the program as desired: $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID} \mid \infty \mid \mathsf{N}_1 \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_1; \mathsf{end}_{\mathsf{atom}}; p \Rightarrow_p [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid \infty \mid \mathsf{N}' \mid p$.

Chapter 5

Undo-Logging

The systems introduced thus far have all assumed a redo-logging interpretation for the crash types calculus: stable values are always stored in nonvolatile memory, and unstable values are always stored in volatile memory. However, this interpretation is tied specifically to redo-logging when in reality, intermittent systems implement many different kinds of logging styles [13, 14, 77, 82, 119, 125], with undo-logging being a popular one.

Whereas redo-logging creates volatile logs for checkpointed variables that are operated on until failure, undo-logging instead creates a separate checkpointed copy at the beginning of an atomic region's first execution and uses it to revert updates to nonvolatile memory in the event of a power failure. Therefore, since the nonvolatile memory locations that are declared as checkpointed are operated on intermittently, their values are *unstable* while their separate checkpointed versions are *stable*. Under this interpretation, unstable values could live in nonvolatile memory. Designing an undo-logging interpretation for crash types that reasons about restoring nonvolatile memory therefore requires a paradigm shift.

This chapter details an undo-logging formulation of the crash types calculus. Reasoning about restoring nonvolatile memory requires us to redefine the crash type for atomic regions to account for unstable values in nonvolatile memory. We first highlight the key intuitions behind the new crash types and present a running example (Section 5.1). Then we detail the system while focusing on the formulation for atomic regions, and explain how the typing rules are connected to adjoint logic (Sections 5.2 and 5.2.2). The chapter concludes with the progress and preservation theorems and a discussion of the logical relation and adequacy (Section 5.3).

5.1 Modal Crash Types for Undo-Logging

In this section, we present the crash types for atomic regions with undo-logging and informally motivate them with a running example. Since in undo-logging the unstable portion of nonvolatile memory is restored, we need to redefine the crash types for atomic regions to reflect the semantics of this operation. We summarize the modified crash types below.

$$\textit{atomic command crash type} \quad \mathsf{C}_{\mathtt{unit}} = (\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$$

$$\textit{atomic expression crash type} \quad \mathsf{C}_A^{\mathsf{aID}} = (\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow A$$

As before, the command crash type $\mathsf{C}_{\mathtt{unit}}$ for atomic regions is a disjunction. The left disjunct arises when the device crashes and volatile memory is wiped. Once more energy is harvested, stable values restore nonvolatile memory from the checkpointed context ($\downarrow\uparrow$), and then the device attempts to re-execute the program ($\mathsf{C}_{\mathtt{unit}}$). As before, the right disjunct represents a successful execution. The interpretation of $\downarrow\uparrow$ for restoring nonvolatile memory with stable values matches its interpretation in the right disjunct and from redo-logging: an intermittent computation that returns a stable value. Like before, the crash type for atomic region expressions matches the one for commands.

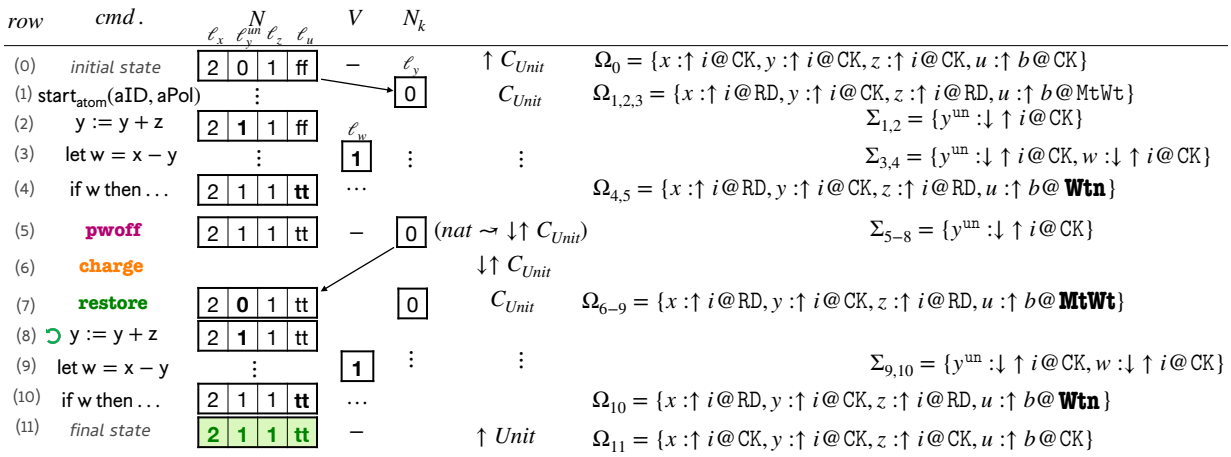

Figure 5.1: Example atomic region execution with undo-logging.

With crash types in hand, we sketch the undo-logging execution of the example atomic region in Fig. 5.1 with its typings. Upon entering an atomic region, the runtime system copies the value of y into a separate location ℓ_y in the nonvolatile checkpoint context N_k, and the original location is tagged with a subscript $\mathtt{un}$ to remember that this version is unstable and that it has a checkpointed copy in K_{undo} (row (1)). The variable y is typed as unstable according to Σ ($y_{\mathtt{un}} : \downarrow\uparrow i@\mathtt{CK}$), while its checkpointed copy in N_k is typed as stable according to Ω ($y : \uparrow i@\mathtt{CK}$). The type of the computation turns from stable ($\uparrow\mathsf{C}_{\mathtt{unit}}$) to unstable ($\mathsf{C}_{\mathtt{unit}}$), and the atomic region begins to execute. The execution writes 1 to the unstable location ℓ_y^{un} (row (2)), binding the result of $x - y$ to a new location ℓ_w (row (3)), and executing the true branch of the conditional which writes the value $\mathtt{tt}$ to ℓ_u (row (4)). At this point, power is about to fail and the device will shut off. Volatile memory is wiped (row (5)) and more energy must be harvested, as represented by the unstable type of the computation: $\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}_{\mathtt{unit}}$. Once the device charges (row (6)), the system restores nonvolatile state by copying the checkpointed values in N_k into nonvolatile memory and reverting the `Wtn` qualifiers back to `MFstWt` as before (row (7)). The combination of these steps has the effect of resetting the unstable values in nonvolatile memory to their stable checkpointed

counterparts to prepare for the next execution. The type of the computation matches this interpretation, changing from $\downarrow\uparrow \mathsf{C}_{\mathtt{unit}}$ to $\mathsf{C}_{\mathtt{unit}}$. Both types are unstable, with the former reflecting that there still remain unstable values in nonvolatile memory that will be cleared, committed, or overwritten (e.g., must-first-write variables), and the latter representing that the program will re-execute on a combination of nonvolatile and volatile state. Upon reboot, the atomic region re-executes as before (rows (8)-(10)). The atomic region execution concludes in row (11) with the correct final state $\ell_x \hookrightarrow 2, \ell_y \hookrightarrow 1, \ell_z \hookrightarrow 1, \ell_u \hookrightarrow \mathsf{tt}$. At this point, the type $\uparrow\mathtt{unit}$ indicates that the results are stable and the execution was successful.

Next, we present the undo-logging formulation of the crash types calculus. Since undo-logging and redo-logging for JIT regions are the same, JIT regions follow the same formulation as before. Therefore, we focus on the formulation for atomic regions. The full system definition can be found in Appendix C.

5.2 Syntax, Operational Semantics, and Typing Rules

In this section, we show the modifications needed to support undo-logging for atomic regions below. We highlight the key changes from Chapter 4 in yellow. The definition of crash instructions change accordingly to reflect the new ordering of modalities in the modified crash type definitions. Additionally, to distinguish nonvolatile locations from their checkpoints, we introduce a separate checkpoint context K_{undo}, as described in Chapter 2. Though since in this system we assume there is no global volatile memory, atomic regions only make use of N_k. Since the checkpoints are only used in the event of a power failure, only the closed configurations carry K_{undo}. Therefore, the operational semantics for open command configurations and values remain the same as before.

crash instructions	i	::=	$\varepsilon \# \mathsf{in}(b > 0, \downarrow\uparrow\kappa) \mid \downarrow\uparrow\kappa$
checkpoint context	K_{undo}	::=	(N_k)
closed configuration	C_c	::=	$[\chi \rhd \varepsilon] \otimes (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K}_{undo} \mid s)$
			$\mid [\chi \rhd \varepsilon] \otimes (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{K}_{undo} \mid s)$

5.2.1 Operational Semantics

We present the key operational semantic rules that must change to support undo-logging in Fig. 5.2. The rule for atomic regions D-P-Ckpt relies on a modified definition of $\mathsf{InitWorld}_d$ to set up the checkpoint context K_{undo}, making checkpointed copies of variables declared in ω (e.g., y) and tagging their original locations in N, which are unstable, with a subscript un. The definition of $\mathsf{FinWorld}_d$ is similar to before; since all updates are made directly to nonvolatile memory, the function simply removes the subscript un from the locations in N' and resets qualifiers back to $\mathtt{CK}$. These definitions can be found in Appendix C.1.

The rule D-Crash-aID models the system when its energy is depleted and where volatile memory is wiped. The configuration steps to a crash instruction that encapsulates the atomic region command c_0 preceded by the combination of modalities $\downarrow\uparrow$. After the device charges

$$\frac{n, n' > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{K}_{undo} \mid \gamma_0 \qquad [\chi \rhd \varepsilon] \otimes \gamma_0 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \cdot \mid \mathsf{K}_{undo} \mid c_0 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{K}_{undo} \mid \mathsf{skip} \qquad \mathsf{FinWorld}_d(\mathsf{N}') = \mathsf{N}''}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \cdot \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p \;\Rightarrow_p\; [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}'' \mid \cdot \mid p} \text{ (D-P-Ckpt)}$$

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K}_{undo} \mid c \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{K}_{undo} \mid c'} \text{ (D-step)}$$

$$\frac{\mathtt{Md} = \mathsf{aID}(c_0)}{\begin{array}{c}[\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K}_{undo} \mid \kappa \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{N} \mid \cdot \mid \mathsf{K}_{undo} \mid \varepsilon \,\#\, \mathsf{in}(b > 0; \downarrow\uparrow c_0)\end{array}} \text{ (D-Crash-aID)}$$

$$\frac{\mathtt{Md} = \mathsf{aID}(c_0)}{\begin{array}{c}[n :: \chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{N} \mid \cdot \mid \mathsf{K}_{undo} \mid \varepsilon \,\#\, \mathsf{in}(b > 0; \downarrow\uparrow\kappa) \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \cdot \mid \mathsf{K}_{undo} \mid \downarrow\uparrow\kappa\end{array}} \text{ (D-charge-aID)}$$

$$\frac{\mathtt{Md} = \mathsf{aID}(c_0) \qquad \mathsf{N} = \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{Wtn}}, \mathsf{N}'''_{\mathtt{un}} \qquad \mathsf{K}_{undo} = (\mathsf{N}_k)}{\begin{array}{c}[\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \cdot \mid \mathsf{K}_{undo} \mid \downarrow\uparrow\kappa \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{MFstWt}}, \mathsf{N}_k \mid \cdot \mid \mathsf{K}_{undo} \mid \kappa\end{array}} \text{ (D-restore-aID)}$$

Figure 5.2: Closed configuration operational semantic rules

and energy is replenished, the nonvolatile state is restored from the checkpoint context and these modalities are dropped, as modeled by D-RESTORE-AID.

Since in undo-logging the checkpoint context is only accessed in the event of a power failure, it does not appear in the rules for expression and command execution under an open configuration. Therefore, the operational semantic rules for expressions and commands remain the same as before. The full operational semantics can be found in Appendix C.1.

5.2.2 Typing Rules

This section highlights the changes to the type system needed to accommodate undo-logging. The full type system definition can be found in Appendix C.2. We summarize changes to the typing constructs in the grammar below.

unstable types	τ^u	::=	$T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid \texttt{nat} \rightsquigarrow \tau^u \mid \mathsf{C}_T^{\texttt{Md}}$
stable types	τ^s	::=	$\uparrow \tau^u$
type variables	$\mathsf{C}_T^{\texttt{Md}}$	::=	$\mathsf{C}_{\texttt{unit}} \mid \mathsf{C}_A^{\texttt{Md}}$
unstable store typing context	Σ	::=	$\cdot \mid x : \downarrow\uparrow A@\texttt{CK}, \Sigma \mid x_{\texttt{un}} : \downarrow\uparrow A@\texttt{CK}, \Sigma$
stable store typing context	Ω	::=	$\cdot \mid x : \uparrow A@q, \Omega$

Since nonvolatile memory is not restored until more energy is harvested, the state will remain unstable thus making $\texttt{nat} \rightsquigarrow \tau^u$ an unstable type. To type the unstable portion of nonvolatile memory, the unstable store typing context is extended with typings of the form $x_{\texttt{un}} : \downarrow\uparrow A@\texttt{CK}$. Lastly, Ω is now used to type both the stable portion of nonvolatile memory as well as the checkpoint context, and hence it consists of typings of the form $x : \uparrow A@q$. Unlike the previous systems for redo-logging, there are no typings for variables of the form $x_{\texttt{un}}$ in Ω.

The typing judgments for commands, expressions, and programs all remain the same, while the judgments for crash instructions are augmented and typed according to the corresponding crash type. Unlike previous chapters, the state throughout processing a power failure remains unstable because nonvolatile memory can contain unstable values until it is restored before the next re-execution. Next, we present a key selection of typing rules for undo-logging.

Setting up typing contexts. The typing rule for atomic regions is the same as before, but relies on a modified definition of $\mathsf{InitWorld}_t$ to suit undo-logging. We present the modified definition below. The function generates an unstable typing context with typings for all variables declared in ω, and tags these typings with the subscript $\texttt{un}$ (e.g. Σ in row (1)). These will type the original nonvolatile locations, that will store unstable values, while Ω is left untouched. The stable typings for checkpointed variables that remain in Ω^c will type the stable copies in the newly-created checkpoint context.

Definition 8 $\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0$ *iff*

- $\mathsf{dom}(\Omega) = \rho \cup \mu \cup \omega$,
- $\rho \cap \mu = \emptyset$, $\mu \cap \omega = \emptyset$, $\rho \cap \omega = \emptyset$,
- $\Omega_0 = \{x : \uparrow A@\texttt{RD} \mid x : \uparrow A@q \in \Omega^r\} \cup \{x : \uparrow A@\texttt{MFstWt} \mid x : \uparrow A@q \in \Omega^m\} \cup \Omega^c$
- $\Sigma_0 = \{x_{\texttt{un}} : \downarrow\uparrow A@\texttt{CK} \mid x : \uparrow A@\texttt{CK} \in \Omega^c\}$

where $aPol = (\rho, \mu, \omega)$ *and* $\Omega = \Omega^r, \Omega^m, \Omega^c$ *and* $\Omega^r = \Omega\restriction\rho$, $\Omega^m = \Omega\restriction\mu$, *and* $\Omega^c = \Omega\restriction\omega$.

Command, expression, and statement typing. We present a key selection of typing rules in Fig. 5.3 whose definitions differ from the previous chapters.

Most of the typing rules for commands and expressions do not change, except for a select few: T-SEQ-D, T-C-SHIFT, and T-W-SHIFT. The rules T-C-SHIFT and T-W-SHIFT are similar to before but share a slightly different interpretation; since stable checkpointed copies of nonvolatile memory are not operated on (only the original nonvolatile locations which are unstable), the premise of these rules selects and promotes the unstable typings of checkpointed nonvolatile locations that reside in Σ while disregarding the checkpointed stable typings $\Omega''_{\texttt{CK}}$. When a post-context for the premise is obtained, the rule constructs a post-context for the conclusion by replacing these

$$\frac{\begin{array}{c} W = \gamma \mid \mathsf{V} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathsf{unit}} \dashv \Omega' \qquad \Sigma = \Sigma^1, \Sigma^2_{\mathsf{un}} \qquad \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2_{\mathtt{RD,MFstWt,Wtn}} \\ \Sigma' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \cup \Sigma^2_{\mathsf{un}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''} \text{ (T-SEQ-D)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega'_{\mathtt{RD,MFstWt,Wtn}}, \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \restriction dom(\Omega'), \Omega''_{\mathsf{CK}}} \text{ (T-C-SHIFT)}$$

$$\frac{\Sigma =\downarrow \Sigma' \qquad \Omega = \Omega'_{\mathtt{RD,MFstWt,Wtn}}, \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A \dashv \Omega_1}{\mathtt{Md} \mid \ b : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega_1 \restriction dom(\Omega'), \Omega''_{\mathsf{CK}}} \text{ (T-W-SHIFT)}$$

$$\frac{\Sigma = \Sigma^1, \Sigma^2_{\mathsf{un}} \qquad \mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\mathsf{un}} \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}}{\mathsf{aID}(c_0) \mid b \ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \kappa : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-AID-}\vee\text{-CRASH)}$$

$$\frac{\varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathtt{Md} \mid \ b \ > 0 : \mathtt{nat} \mid \ \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\uparrow \kappa : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}}{\mathtt{Md} \mid \ \cdot \ \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow \kappa) \ : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}} \text{ (T-CHARGE)}$$

$$\frac{\mathsf{aID}(c_0) \mid \ b \ > 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \vdash_{\mathtt{Sig}} \kappa \ : \mathsf{C}^{\mathsf{aID}}_{T} \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid \ b \ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\uparrow \kappa : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}} \text{ (T-AID-RESTORE)}$$

Figure 5.3: Selected typing rules

typings with their stable counterparts Ω''_{CK}. This enforces that the domain of the initial typing context continues to match the post-context.

In T-SEQ-D, since the function `trim` is defined to throw out all typings from Σ that do not correspond to in-scope volatile variables, additional handling is required to recover the checkpointed variable typings (denoted Σ^2_{un}) in Σ'.

The remainder of the rules in Fig. 5.3 type crash instructions according to the corresponding crash types. The T-AID-$\vee$-CRASH rule detects a crash and generates a typing for the crash instruction $\varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0)$ according to the type $\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}$ in the premise (e.g., row (5)). The typings for volatile memory are dropped to match the dynamic rule for crash. The T-CHARGE rule inputs a new energy level from ε (e.g., row (6)) that is a natural number greater than zero.

The T-AID-RESTORE rule drops the combination of modalities to model restoring the checkpointed state, matching the dynamic rule (e.g., row (7)). The same typing contexts propagate from conclusion to premise to type the recovered continuation, which will be the atomic region command c_0. The rule checks if there are already typing derivations matching the signature.

5.2.3 Interpretations in Adjoint Logic

The typing rules for checkpoint, restore, commit, and power off continue to have interpretations in adjoint logic, though they differ from before to match the semantics of undo-logging. The

main difference regards checkpointing. In undo-logging, the newly-created checkpointed copies of nonvolatile locations are stable, whereas the original locations become unstable when tagged $\mathtt{un}$. Therefore, we cannot simply apply a $\uparrow L^*$ rule as before to copy the stable typing into the unstable typing context because the original nonvolatile locations do not remain stable. To reason about checkpointing in undo-logging, we introduce a rule $C^{\mathtt{un}}$ that is based on contraction and an interpretation of the tag $\mathtt{un}$. In particular, the tag $\mathtt{un}$ is generated when moving a typing from the stable context to the unstable context, and is removed on the inverse.

$$\frac{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \_{:}\mathtt{un} : \downarrow^s_u \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \tau^u}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \cdot \vdash \_ : \tau^u}\ C^{\mathtt{un}}$$

From conclusion to premise, the rule $C^{\mathtt{un}}$ first uses contraction to copy the stable typing of a checkpointed variable, and then tags the original with $\mathtt{un}$ which moves the typing into the unstable typing context (e.g., $y{:}\uparrow i @ \mathtt{CK}$ and $y^{\mathtt{un}}{:}\downarrow\uparrow i @ \mathtt{CK}$ in row (1), respectively). This matches the dynamics of checkpointing in undo-logging, which makes checkpointed copies of locations whose values will remain stable and tags the original location with $\mathtt{un}$.

We are now ready to sketch the adjoint logic derivations of our typing rules. We refer to the typing rule skeletons from Chapter 2 throughout.

Checkpointing (T-P-Ckpt)**.** The typing rule for checkpointing corresponds to a combination of the rules $C^{\mathtt{un}}$ and $\uparrow R$. The rule $\uparrow R$ creates an empty unstable typing context, which the rule $C^{\mathtt{un}}$ populates with unstable typings of checkpointed variables, as described above.

$$\frac{\dfrac{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \_{:}\mathtt{un} : \downarrow^s_u \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \tau^u}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \cdot \vdash \_ : \tau^u}\ C^{\mathtt{un}}}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \uparrow^s_u \tau^u}\ \uparrow R$$

Crash (T-aID-$\vee$-Crash)**.** A crash corresponds to a combination of standard weakening and logical or rules (WL_1 and $\vee R$). In the derivation, a $\vee R$ rule first applies to select the first disjunct of the type, indicating power failure. Then, a WL_1 rule applies to eliminate typings of volatile variables in the premise.

$$\frac{\dfrac{\Omega; \Sigma \vdash \_ : \tau^u}{\Omega; \Sigma, \_ : \downarrow^s_u \uparrow^s_u A^u \vdash \_ : \tau^u}\ WL_1}{\Omega; \Sigma, \_ : \downarrow^s_u \uparrow^s_u A^u \vdash \_ : \tau^u \vee \downarrow\uparrow \mathtt{unit}}\ \vee R$$

Restore (T-aID-Restore)**.** Restore corresponds to a combination of the rules $\downarrow R$, $\uparrow R$, and $C^{\mathtt{un}}$. The derivation is similar to checkpointing, but begins with a $\downarrow R$ step to lose the unstable typing context, which corresponds to forgetting the old nonvolatile values. Then $\uparrow R$ and $C^{\mathtt{un}}$ apply to regenerate the unstable nonvolatile typings.

$$\frac{\dfrac{\dfrac{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \_{:}\mathtt{un} : \downarrow^s_u \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \tau^u}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \cdot \vdash \_ : \tau^u}\ C^{\mathtt{un}}}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \uparrow \tau^u}\ \uparrow R}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \Sigma \vdash \_ : \downarrow\uparrow \tau^u}\ \downarrow R$$

Commit. Lastly, the commit operation – which drops the checkpointed copies and removes the un tags from the original nonvolatile locations – corresponds to a combination of $\downarrow R$ with a modified weakening rule $WL_2^{\mathtt{un}}$. First, the rule $WL_2^{\mathtt{un}}$ applies, where the stable typing of a checkpointed copy is lost and the tag un transports the unstable counterpart back to the stable typing context. Then, a $\downarrow R$ rule applies to close off the remaining unstable typing context. Since no volatile variables are in scope at the end of an atomic region execution, Σ is always empty.

$$\dfrac{\dfrac{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \tau^s}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \Sigma \vdash \_ : \downarrow^s_u \tau^s}\ \downarrow R}{\Omega, \_ : \uparrow^s_u A^u @ \mathtt{CK}; \Sigma, \_{\mathtt{un}} : \downarrow^s_u \uparrow^s_u A^u @ \mathtt{CK} \vdash \_ : \downarrow^s_u \tau^s}\ WL_2^{\mathtt{un}}$$

5.3 Metatheory

In this section, we discuss progress and preservation for the calculus with undo-logging (Section 5.3.1), and conjecture about the proof of logical relation (Section 5.3.2).

5.3.1 Progress and Preservation

While the progress theorem remains the same as before, the preservation theorem differs from the previous chapter in its treatment of checkpoints. We state the preservation theorem for commands below.

Theorem 8 (Preservation for Commands) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \ \geq\ 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$$

and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed,* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*, and for some (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$$

then for some Ω_0

$$\mathtt{Md} \mid b \ \geq\ 0 : \mathit{nat} \mid \Omega_0; \Sigma \vdash_{\mathtt{Sig}} c' : \tau \dashv \Omega'$$

where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega_0 \mid \Sigma$*,* $\Omega > \Omega_0$*, and* $n' \geq 0$*. Moreover* $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *is well-formed. Moreover,* $\mathit{dom}(\mathsf{N}) = \mathit{dom}(\mathsf{N}')$*, and if* $\mathtt{Md} = \mathsf{alD}(c_0)$*, then* $\mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$.

The well-formedness definition for configurations in undo-logging is similar to redo-logging. However, since in undo-logging volatile memory V houses no checkpointed copies of locations, the nonvolatile memory N is no longer mentioned in this version of the definition; the definition reasons only about local variables in volatile memory. Additionally, the store typing definition for checkpointed variables in N is extended to consume both an unstable typing and stable typing for its checkpointed counterpart. This will be needed for setting up the logical relation (to be discussed). We provide the full definitions in Appendix C.2.

Since the checkpoint context is not carried by open configurations, the proofs of progress and preservation and their supporting lemmas remain mostly the same with modifications to the cases corresponding to the few typing and dynamic rules that changed. We present the full proofs in Appendix C.3.

5.3.2 Logical Relation and Adequacy

Though the proofs of logical relation and adequacy for undo-logging are left to future work, we conjecture about their challenges relative to previous chapters. In particular, the new ordering of modalities for crash types in undo-logging will require a new logical relation for reasoning about correctness. The term relation will be similar to the ones from Chapters 3 and 4, while the value relation will reflect the new ordering of modalities and rely on new policies for `PwOff`, `Restore`, and `Commit`. We include their definitions here, which match their dynamic counterparts.

$$\frac{\gamma' \subseteq \gamma \; s.t. \; range(\gamma') = \mathsf{dom}(\mathsf{N}_1) \qquad \mathsf{V}_2 = \emptyset}{\texttt{PwOff}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{1\,\texttt{RD,Wtn}}, \mathsf{N}''_{\texttt{un}} \qquad \mathsf{N}_2 = \mathsf{N}'_{1\,\texttt{CK}}, \mathsf{N}'' \qquad \gamma' \subseteq \gamma \; s.t. \; range(\gamma') = \mathsf{dom}(\mathsf{N}_2)}{\texttt{Commit}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1) = \gamma' \mid \mathsf{N}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}''_{\texttt{un}}, \mathsf{N}^3_{\texttt{Wtn}} \qquad \mathsf{K} = (\mathsf{N}_k) \qquad \mathsf{N}_2 = \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}_k, \mathsf{N}^3_{\texttt{MFstWt}}}{\texttt{Restore}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{K}) = \mathsf{N}_2 \mid c_0}$$

Figure 5.4: `Commit`, `Restore`, and `PwOff` policy definitions for undo-logging

As before, the `PwOff` policy clears volatile memory and throws out their variable-location mappings from γ. Since all updates are made directly to nonvolatile memory in undo-logging, the `Commit` policy simply drops the `un` tag from locations. Lastly, `Restore` prepares for re-execution by resetting the `Wtn` qualifiers to `MFstWt` (like before), and the values of locations marked `un` with their checkpointed counterparts N_k. Since the checkpoint context is needed by the `Restore` policy but is omitted from open configurations, the top-level logical relation will include it in its store typing. The same definition of store typing described above, where the checkpoint context was disregarded can be used to type $\vdash \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} : \Omega \mid \Sigma$. When checked together, the checkpointed variables in N and K will consume their unstable and stable typings.

In proving the fundamental theorem of logical relation, the inductive case must again establish that the restored nonvolatile memories of the intermittent execution match the continuous execution up to `MFstWt` variables. For undo-logging, establishing this condition will hinge on the fact that `Restore` additionally reverts updates to unstable nonvolatile locations with their checkpointed copies.

Chapter 6

Non-idempotence, Inputs, and Outputs

Though the systems presented in Chapters 4 and 5 adopt more flexible and realisitic checkpointing policies into reasoning about crash types, the programs supported by those systems are still limited in expressivity, lacking supports common to many embedded systems applications, such as sensor inputs, outputs, and non-idempotence. This chapter outlines the challenges behind supporting a few such features, and extends the crash type calculus accordingly.

We begin by motivating the importance and challenges behind supporting non-idempotence, sensor inputs, and outputs with respect to correct intermittent execution in Sections 6.1 and 6.2. In Section 6.3, we introduce the key ideas behind our approach. Then, we present the syntax and semantics for our calculus in Section 6.4, followed by the type system in Section 6.5. The chapter concludes with progress and preservation, and a discussion about logical relation and adequacy (Section 6.6).

6.1 Supporting Non-idempotence

Thus far, our systems have assumed working with only data that must produce the same results on every (re-)execution. However, some pieces of data in a program must be allowed to differ across (re-)executions in order to have correct execution behavior, such as a variable that counts the number of reboots endured by an atomic region. The former variety we refer to as *idempotent*, and the latter as *non-idempotent*. In order to uphold that idempotent variables compute the same results across (re-)executions, idempotent computation must not depend on non-idempotent values. Thus, supporting the correct interaction between idempotent and non-idempotent data is essential for upholding correctness guarantees in real intermittent systems.

To support the combination of (non-)idempotent data in a program, Curricle [121] uses a special form of dependency analysis [3] called an *information flow type system* to enforce that non-idempotent data does not flow to idempotent variables. Curricle defines types for non-idempotence (Nid) and idempotence (Id) according to a join semilattice $\mathsf{Id} \sqsubseteq_{id} \mathsf{Nid}$, where join is denoted as $\sqcup_{id}$.

To better understand how such an information flow type system works, consider the two programs in Fig. 6.1. Program (a) increments a reboot counter `rb`, and writes its contents to `log`. Since `rb` purposefully records the number of power failures undergone by the atomic region, it is

```
(a) start_atom(a1, aPol)
      rb ++;
      log := rb
    end_atom
```

```
(b) start_atom(a2, aPol)
      rb++;
      if rb > 3 then
        log := 0
      else
        skip
    end_atom
```

Figure 6.1: Two incorrect programs, where aPol = {rb : Nid, log : Id}

expected to differ across re-executions of a program and is therefore specified as non-idempotent, whereas log is specified as idempotent. Therefore, the assignment log := rb would illegally cause an idempotent variable log to depend on non-idempotent data.

To automatically check whether a program is correct, Curricle's type system first encodes the atomic region policy into a typing context for checking the program, e.g., Γ = {rb : Nid, log : Id}, which it uses to type check the program. For example, to check whether an assignment $x := e$ obeys the policy, Curricle defines a rule similar to T-ASSIGN-A. The rule looks up the label of x in Γ, checks e according to the rules T-VAR, T-CONST, and T-BINARY to determine its label qID_e, and lastly checks that qID_e is less than or equal to the label of x ($qID_e \sqsubseteq_{id} \Gamma(x)$). By definition, this check would rule out dependencies of idempotent variables on non-idempotent data, such as from rb to log.

$$\text{T-CONST}\ \frac{}{\Gamma \vdash v : \mathsf{Id}} \qquad \text{T-VAR}\ \frac{\Gamma(x) = qID}{\Gamma \vdash x : qID} \qquad \text{T-BINARY}\ \frac{\Gamma \vdash e_1 : qID_1 \qquad \Gamma \vdash e_2 : qID_2}{\Gamma \vdash e_1 \odot e_2 : qID_1 \sqcup qID_2}$$

$$\text{T-ASSIGN-A}\ \frac{\Gamma \vdash e : qID_e \qquad qID_e \sqsubseteq_{id} \Gamma(x)}{\Gamma \vdash x := e : \mathsf{ok}}$$

However, programs with non-idempotence are additionally prone to control flow dependencies, which is not handled by the previous checking alone. For example, consider the program (b) in Fig. 6.1. The variable log is reset to 0 after three re-executions, i.e., if rb > 3. Though log would not read the value of rb explicitly, its computation still depends on non-idempotent data. In this case, we could either adjust our policy to allow log to rely on rb by specifying it as non-idempotent, or we could upgrade our type checking rules to reject this program. The latter involves augmenting our typing rules with a ghost variable *pc* that will represent the idempotence level of a conditional expression while checking its branches. If an assignment to a idempotent variable occurs under a non-idempotent *pc*, the program will fail to type check. We show the augmented rules below.

$$\text{T-ASSIGN-B}\ \frac{\Gamma \vdash e : qID_e \qquad qID_e \sqcup_{id} pc \sqsubseteq_{id} \Gamma(x)}{\Gamma, pc \vdash x := e : \mathsf{ok}}$$

$$\text{T-IF-B}\ \frac{\Gamma \vdash e : qID_e \qquad \Gamma, pc \sqcup_{id} qID_e \vdash c_1 : \mathsf{ok} \qquad \Gamma, pc \sqcup_{id} qID_e \vdash c_2 : \mathsf{ok}}{\Gamma, pc \vdash \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \mathsf{ok}}$$

Notably, the rule T-IF-B checks if a conditional is safe against control flow dependencies by checking the conditional expression and determining its qualifier (qID_e), raising the program counter to account for qID_e ($pc \sqcup_{id} qID_e$), and then checking whether each branch c_1 and c_2 is correct under the raised program counter. To check that no non-idempotent data could flow to idempotent variables through assignment, the rule T-ASSIGN-B checks that the label of e joined with the pc is bounded by the label of x ($qID_e \sqcup_{id} pc \sqsubseteq_{id} \Gamma(x)$). Under these typing rules, the program in Fig. 6.1 (b) will be rejected; the branching condition depends on a `Nid` variable, causing the program counter to be raised to `Nid` and the assignments within the conditional to fail type checking by T-ASSIGN-B, since non-idempotent data flows to an idempotent variable.

6.2 Challenges of Supporting I/O-Dependent Programs

Sensor inputs are also important for reasoning about real systems, but present challenges for supporting correct intermittent execution. In particular, supporting conditionals that depend on sensor inputs is important because many embedded systems applications read sensor inputs from hardware peripherals and make decisions accordingly, e.g., raising a flag if the temperature is too hot or too cold. Since sensor inputs often depend on an array of environmental conditions under which they are taken [120], programs depending on them are therefore *non-deterministic* in nature. In the context of intermittence, reliance on inputs could cause program behavior (and subsequently, observable state) to differ across (re-)executions, e.g., executing one branch initially, and the other on re-execution because a different input is obtained. If checkpointing is not handled correctly, a program running intermittently could compute results that are impossible to retrieve from a continuously-powered execution of it. Such *input-dependent* programs are not supported by the existing crash type calculus. For example, choosing not to checkpoint all writable variables forces conditionals to access variables in the same way across all possible execution paths, as in Chapters 4 and 5. Such constraints make the system difficult to use and incapable of supporting many correct programs. Lastly, effectful outputs introduce additional challenges for reasoning about program correctness in the presence of intermittence and are not currently supported by any provably correct intermittent system.

This section details the challenges involved in reasoning about programs with inputs and outputs (I/O) with respect to intermittence.

6.2.1 Repeated I/O (RIO) Bugs

Programs that rely on I/O are prone to an additional class of memory consistency bug, where re-execution could cause unintended repetition of outputs or divergent execution paths due to nondeterministic sensor inputs [108, 118]. For example, consider the atomic region in Fig. 6.2 which stores a temperature reading from the environment in a variable *temp* and branches on it; the program raises a flag h to tt if the reading is high (greater than 0), and a flag c to tt otherwise.

Let ℓ_h, ℓ_c, and ℓ_t be nonvolatile memory locations corresponding to h, c, and *temp*, and suppose that the initial state is $\ell_h \hookrightarrow \mathsf{ff}, \ell_c \hookrightarrow \mathsf{ff}$. If the sensor reads a temperature of 30, the first branch of the conditional will execute, raising the flag h to tt. Supposing that power fails immediately after, volatile memory is wiped, leaving behind a nonvolatile state of $\ell_h \hookrightarrow \mathsf{tt}, \ell_c \hookrightarrow$

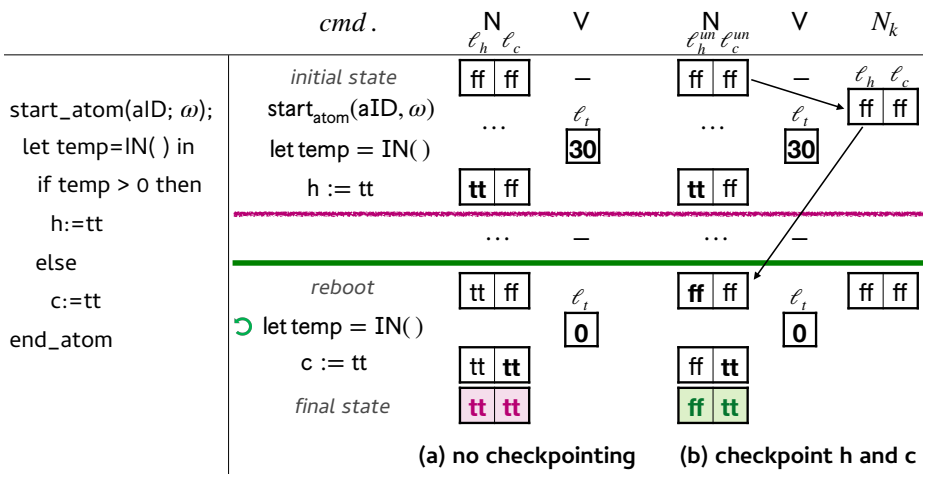

Figure 6.2: Possible atomic region executions, (a) incorrect and (b) correct.

ff. When energy is replenished, the program re-executes on this state. However, in the time it took to harvest more energy, the surrounding environmental conditions could have changed significantly. Suppose that now it is nighttime and the temperature has sunk to 0, such that the else branch of the conditional executes, setting c to tt. When the program finishes executing, the final state will be $\ell_h \hookrightarrow \text{tt}, \ell_c \hookrightarrow \text{tt}$. These results do not match the behavior of any single continuously-powered execution of the same program, therefore making it incorrect. For those results to be correct, a single continuously-powered execution would need to run both branches of the conditional, which is impossible.

This program executes incorrectly because it has a *repeated I/O (RIO) bug*, where effectful outputs are unintentionally repeated or non-deterministic sensor inputs cause program behavior to differ on each (re-)execution of the program. To make programs with sensor inputs execute correctly, prior work has established that an intermittent system additionally needs to checkpoint variables that are written on at least one path but not on all [118, 121], e.g., h and c.

Returning to the program in Fig. 6.2, suppose that h and c are checkpointed by the atomic region with undo-logging. The first execution proceeds as before; the sensor reads a value of 30, stores it in ℓ_t, causing the first branch of the conditional to run, writing tt to the nonvolatile location ℓ_h^{un}. If power fails immediately after, the buffer and volatile memory are both wiped and the unstable value stored in $\ell un_h \hookrightarrow \text{tt}$ is reverted to its checkpointed counterpart. On reboot, the program will re-execute on the nonvolatile state $\ell_h^{un} \hookrightarrow \text{ff}, \ell_c^{un} \hookrightarrow \text{ff}$, which happens to match the initial state. The re-execution reads a sensor input of 0 and executes the second branch of the conditional, writing tt to ℓ_c^{un}. Execution ends in the correct final state $\ell_h^{un} \hookrightarrow \text{ff}, \ell_c^{un} \hookrightarrow \text{tt}$.

6.2.2 State-of-the-Art RIO Bug Supports are Limited

A system that protects against RIO bugs therefore must reject programs that do not checkpoint sufficient variables and safeguard against repetition of effective outputs due to atomic region re-execution. We examine limitations in existing supports for repeated inputs and outputs separately.

Prior work's repeated input checking is too conservative. Prior work on correctly handling

sensor inputs is limited. The system introduced in Chapter 4 that only checkpoints WAR variables does not handle programs with sensor inputs. Though, if that system were extended naively to accommodate sensor inputs, the incorrect programs would still fail to type check only because we assumed that in a given atomic region the same variables are written across conditional branches, meaning that effects from failed prior executions that are stored in those variables would always be overwritten on the next execution. However, this constraint makes that system incapable of supporting many correct programs that write to different variables on some program paths but not on all, regardless of whether or not those programs depend on sensor inputs.

To accept more correct programs, we could implement a conservative merge as in Curricle [121]. To track the propagation of input-dependent and noninput-dependent data in a program, Curricle marks data as tainted (`Tnt`) or non-tainted (`Nt`), respectively, and defines a join semi-lattice over taint qualifiers $\texttt{Nt} \sqsubseteq_t \texttt{Tnt}$ with join denoted as $\sqcup_t$. In particular, Curricle co-designed an inference of access qualifiers and a type checking algorithm, which together enforce that the inferred qualifiers for a given program guarantee its freedom from RIO bugs. Together, their system enforces that if a variable is written on one branch of a tainted conditional but not on the other, then that variable *must* be checkpointed.

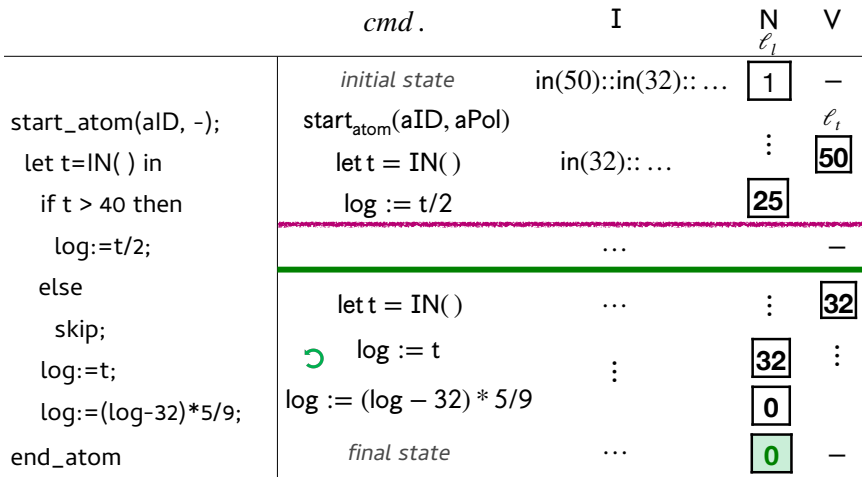

Figure 6.3: A correct program that is not supported by prior work [121].

However, their approach to identifying RIO bugs is still overly conservative, eagerly rejecting an entire class of input-dependent programs that are actually correct. In particular, prior work [121] does not consider accesses (particularly, writes) that occur after the branch merge. We show one such program in Fig. 6.3, where no variables are checkpointed. This program first reads a temperature and stores it in the variable t. If the reading is too high ($t > 40$), half of the value is written to the variable *log*. After, *log* is set to t and converted to degrees Celsius. That is, *regardless of which branch actually executes, log would be overwritten with the same value temp on all executions before it is later read by the computation*. In other words, if *log* is not checkpointed, the program would still compute the correct results.

For example, consider the execution to the left of Fig. 6.3, where, like before, variables are associated with memory locations: $log \mapsto \ell_l$. Suppose that the program executes and reads a temperature of 50, stores it in t, which causes the first branch to execute and write 25 to ℓ_l.

Suppose that power fails immediately after, wiping the volatile state. Since no variables were checkpointed, the program re-executes on the state $\ell_l \hookrightarrow 25$. If the program takes the else branch on its re-execution, the value of *log* gets overwritten to t after the conditional, and before it gets read by the conversion to Celsius. Supposing the new input is 32, the execution concludes successfully on this attempt with the final state $\ell_l \hookrightarrow 0$, which corresponds to a continuously-powered execution that reads a temperature of 32. This result is correct, particularly because the write $log := t$ would overwrite the results from prior failed executions before *log* is read.

There are no provably correct supports for repeated outputs. Currently, there are no provably correct supports for effectful outputs in intermittent computing. If handled naively, the re-execution of an atomic region could unintentionally cause outputs to repeat, potentially spelling incorrect program behavior.

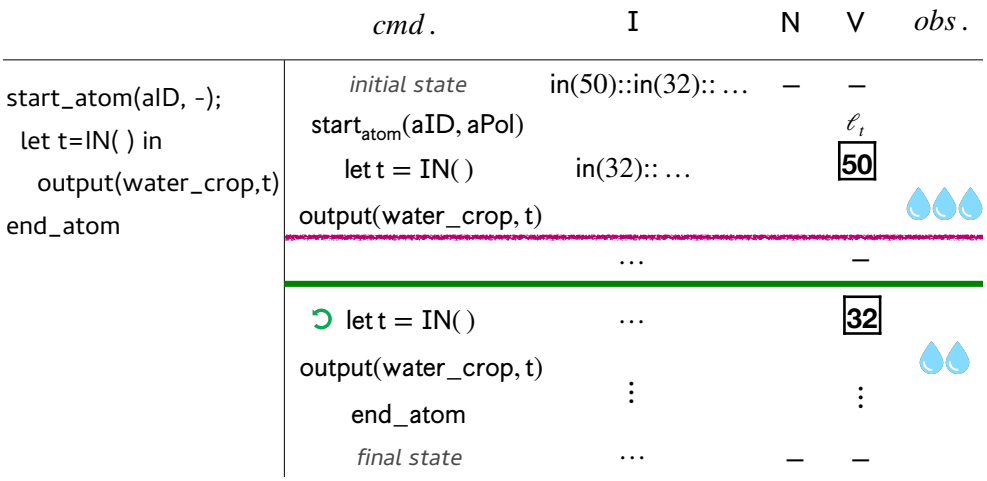

Figure 6.4: Naively supporting outputs in atomic regions is incorrect.

For example, consider the program in Fig. 6.4 which takes a temperature reading, stores it in t, and then triggers an automatic crop watering system that dispenses an amount of water proportional to t. We mark observed outputs over the course of execution, e.g., the amount of water discharged, under a column *obs*.. Suppose that the program executes, reading a temperature of 50 and watering the plants accordingly. If power fails soon after, the program would re-execute from the beginning, taking another temperature reading and watering the crops yet again. When the program finally finishes executing, the program behavior is incorrect as multiple waterings have occurred, meanwhile a programmer likely intended for the crops to be watered only once – overwatering could cause flooding, water waste, and crop ruin.

Therefore, developing a system capable of supporting outputs and accepting more correct I/O-dependent programs is an important yet challenging task, requiring runtime supports for outputs and a more flexible type checking algorithm for inputs than is considered by prior work.

6.3 Key Ideas

We present the key ideas behind our flexible RIO bug checking and supports for I/O.

6.3.1 Fine-Grained Qualifier Evolution for Repeated Inputs

This section provides intuition behind our approach to a more flexible type checking algorithm for RIO bugs in input-dependent programs. To correctly support inputs while accepting more correct programs than prior approaches [121], we must develop more fine-grained access qualifiers and a checking algorithm that does not eagerly reject spurious RIO bugs – one that is capable of inspecting variable accesses occurring after conditionals.

The key idea is to track memory accesses and the propagation of tainted data to variables and to delay failing in input-dependent conditionals until the end of the atomic region. A program will be rejected if by the end of checking an atomic region, there is a variable written on one input-dependent branch but not on all. We make use of taintedness and idempotence qualifiers from Curricle [121], and embellish idempotence qualifiers with an access qualifier describing why a variable is idempotent. To track information about variable accesses over the course of type checking and execution, we extend the access qualifier evolution from Chapter 4 with more sophisticated qualifiers and evolution. Like before, our access qualifiers describe how variables have been accessed previously according to key memory access patterns. In this chapter, these patterns become more fine-grained, including variables that are checkpointed (`CK`) and initially not read yet (`NRD`), which could evolve to other qualifiers over the course of type checking: first read on *some* path (`mayFstRD`), written in a tainted branch on *all* paths (`MTntWt`), written on *some* path but not all (`mayTntWt`), and written on *all* paths (`Wtn`). The intuition is that variables that are marked `NRD` are safe to access freely, whereas variables marked `mayFstRD` could be susceptible to WAR bugs on a subsequent write, and those that are `mayTntWt` could be susceptible to RIO bugs if they are read before being overwritten on all program paths (`MTntWt` and `Wtn`). The type system presented in this chapter will use these qualifiers to perform local checks, and combine them to perform a whole-program analysis.

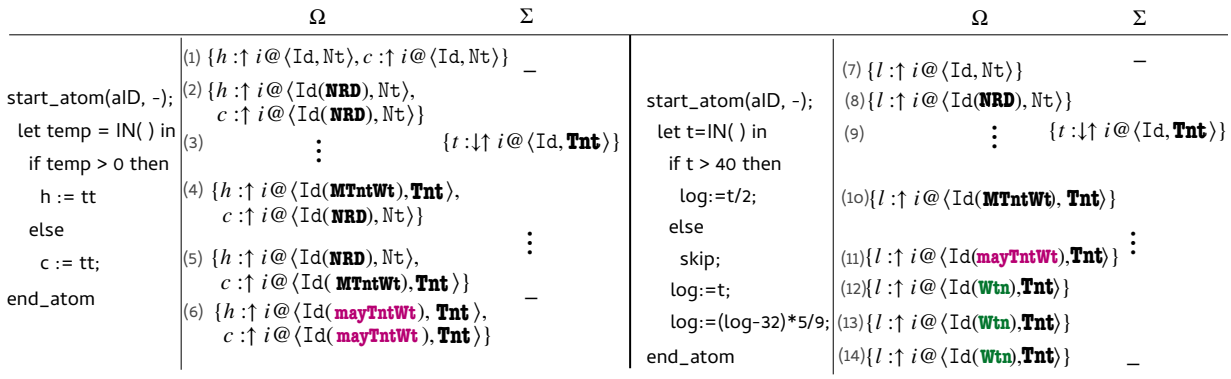

Figure 6.5: Example program typings, (a) accepted and (b) rejected.

Rejecting incorrect programs. Under this schema, a program would fail type checking if (1) a `mayFstRD` variable is written, (2) a `mayTntWt` variable is read, or (3) there are remaining `mayTntWt` variables by the end of checking an atomic region. The first case points to a WAR bug, while the other two indicate RIO bugs, where tainted data written on one branch but not on all could influence future program execution.

For example, recall the incorrect program from the previous section that failed to checkpoint variables h and c. We show the program next to its typings in Fig. 6.5 (a). In row (1), the variables h and c begin with type $\uparrow i@\langle \mathsf{Id}, \mathtt{Nt}\rangle$, indicating that they will be checked as idempotent and initially contain non-tainted data. When the atomic region begins in row (2), their types are set to $\uparrow i@\langle \mathsf{Id}(\mathtt{NRD}), \mathtt{Nt}\rangle$ indicating that the variables have not been read yet. Then, the program writes a sensor input to a variable *temp*. Since that variable now stores tainted data, its type is $\downarrow\uparrow i@\langle \mathsf{Id}, \mathtt{Tnt}\rangle$. When checking the conditional, we check the first branch where h is written and evolve its type to $\uparrow i@\langle \mathsf{Id}(\mathtt{MTntWt}), \mathtt{Tnt}\rangle$ (row (4)). The qualifier `MTntWt` indicates that at this point of the checking, h was written on a tainted branch on all program paths. In row (5), the checking computes the analogous result for the else branch where c is written. At the end of the conditional (row (6)), the program execution paths rejoin. However, on one branch only h was written and on the other only c was written, meaning that by the end of the conditional neither variable has been written on all program paths, as the qualifier `MTntWt` purports. Since c and h each were written on a tainted branch, but not on all, their access qualifiers become `mayTntWt`. The type checking therefore fails at the end of the atomic region. The leftover `mayTntWt` variables indicate that execution could end in a final state with data from failed prior executions, and that these variables need to be checkpointed for the program to execute correctly, like the Fig. 6.2 example executions from the previous section.

Accepting more correct programs. A program with `mayTntWt` variables could still pass type checking if it is overwritten by the end of the atomic region, e.g., *log* in the Fig. 6.3 program. This allows us to be more flexible than Curricle and to accept more correct programs.

We show how our type system is able to accept the actually correct program from Fig. 6.3 in Fig. 6.5 (b). The type checking begins similarly to the previous example, deriving a qualifier `mayTntWt` in row (11) after the conditional. In its subsequent write, the qualifier of *log* evolves to `Wtn`, indicating that the variable has been written on all paths. The atomic region type checking successfully concludes with no remaining `mayTntWt` variables.

Evolving access qualifiers, formally. We take the opportunity to formalize the intuitions behind our evolving access qualifiers. To evolve access qualifiers sequentially, we redefine the access transition function δ from Chapter 4 to now take as inputs the current *pc*, an idempotence qualifier, and an access. We define the function below.

$$\delta(pc, \mathsf{Id}(\mathtt{NRD}), Rd) = \delta(pc, \mathsf{Id}(\mathtt{mayFstRD}), Rd) = \mathsf{Id}(\mathtt{mayFstRD})$$
$$\delta(\mathtt{Nt}, \mathsf{Id}(\mathtt{NRD}), Wt) = \delta(\mathtt{Nt}, \mathsf{Id}(\mathtt{mayTntWt}), Wt) = \delta(pc, \mathsf{Id}(\mathtt{Wtn}), a) = \mathsf{Id}(\mathtt{Wtn})$$
$$\delta(\mathtt{Tnt}, \mathsf{Id}(\mathtt{NRD}), Wt) = \delta(\mathtt{Tnt}, \mathsf{Id}(\mathtt{mayTntWt}), Wt) = \delta(\mathtt{Tnt}, \mathsf{Id}(\mathtt{MTntWt}), a) = \mathsf{Id}(\mathtt{MTntWt})$$
$$\delta(pc, \mathsf{Id}(\mathtt{mayFstRD}), Wt) = \delta(pc, \mathsf{Id}(\mathtt{mayTntWt}), Rd) = UN$$
$$\delta(pc, \mathsf{Id}(\mathtt{CK}), a) = \mathsf{Id}(\mathtt{CK}) \qquad \delta(pc, \mathsf{Id}, a) = \mathsf{Id} \qquad \delta(pc, \mathsf{Nid}, a) = \mathsf{Nid}$$

First, writing on a qualifier $\mathsf{Id}(\mathtt{NRD})$ in a tainted context results in $\mathsf{Id}(\mathtt{MTntWt})$ (e.g., Fig. 6.5 row (10)) whereas writing on a qualifier $\mathsf{Id}(\mathtt{NRD})$ evolves it to $\mathsf{Id}(\mathtt{MFstWt})$ or $\mathsf{Id}(\mathtt{Wtn})$ in a tainted or non-tainted context, respectively. If a variable with type $\mathsf{Id}(\mathtt{NRD})$ is read under any context, its qualifier becomes $\mathsf{Id}(\mathtt{mayFstRD})$, and any future reads would result in the same qualifier $\mathsf{Id}(\mathtt{mayFstRD})$. Writing on a variable with type $\mathsf{Id}(\mathtt{mayTntWt})$ will result in a qualifier $\mathsf{Id}(\mathtt{MTntWt})$ or $\mathsf{Id}(\mathtt{Wtn})$ if written under a tainted or non-tainted *pc*, respectively (e.g., Fig. 6.5 row (12)). Any

subsequent write on these variables under the same *pc* context results in the same qualifier (e.g., Fig. 6.5 row (13)). Lastly, any access to variables with the qualifiers Id(CK), Id, and Nid is allowed and results in the same qualifier. The transitions resulting in undefined behavior *UN* would cause a program to fail type checking due to ill-checkpointed variables in WAR and RIO code patterns.

To merge a variable's access qualifiers at the end of a conditional, as we do in Fig. 6.5 row (11), we must select the most restrictive access qualifier between the two. We observe that our access qualifiers constitute a semi-lattice where selecting the more restrictive qualifier corresponds to a join operation (denoted $\vee_a$). In particular, the access qualifiers follow the partial order: $\texttt{Wtn} \leq_a \texttt{NRD}$, $\texttt{NRD} \leq_a \texttt{mayFstRD}$, $\texttt{NRD} \leq_a \texttt{mayTntWt}$ and $\texttt{MTntWt} \leq_a \texttt{mayTntWt}$. The qualifier CK is separate because it does not evolve on any access. We denote join on access qualifiers as $\vee_a$, and extend the definition of $\sqcup_{id}$ with a case for merging interior access qualifiers: $\mathsf{Id}(qAcc_1) \sqcup_{id} \mathsf{Id}(qAcc_2) = \mathsf{Id}(qAcc_1 \vee_a qAcc_2)$.

6.3.2 Buffering Prevents Repeated Outputs

Supporting correct output behavior depends on what behavior is deemed acceptable for a given application. Applications requiring outputs to occur exactly once, e.g., watering crops, could suffer from incorrect or even disastrous behavior if outputs are repeated. Alternatively, other applications with higher levels of fault tolerance may withstand or even anticipate repeated outputs, e.g., a steering wheel calibration algorithm that resets a steering wheel position on every invocation.

Therefore, a correct intermittent runtime system for outputs must ensure that outputs expected to run exactly once will do so despite atomic region re-executions. We refer to outputs required to run exactly once in an atomic region as *idempotent*, e.g., `water_crop`, and those that tolerate repetition as *non-idempotent*. To correctly support idempotent outputs in an atomic region, we make use of a volatile *output buffer*. When an idempotent output is encountered during program execution, its effects are delayed in the output buffer, to be cleared on power failure and released at the end of the atomic region. In the Fig. 6.4 program, such an output buffer would delay the first occurrence of `water_crop`, forget it on power failure, and delay its reccurrence on re-execution until the end of the atomic region. Therefore, only the second watering where t is 32 will occur in the corrected program execution. This strategy of buffering outputs allows the observed idempotent outputs made in the intermittent execution of an atomic region to match those made by a continuously-powered execution of the same program.

In the next section, we extend the crash type calculus with non-idempotence and supports for more flexible RIO bug checking. Throughout this chapter we focus on atomic regions whose re-execution primarily impacts the behavior of sensor inputs, effectful outputs, and non-idempotence. The full system definition including supports for JIT regions is given in Appendix D.

6.4 Syntax and Operational Semantics

This section presents the syntax and operational semantics for our crash types calculus with non-idempotence, inputs, and outputs. We present necessary extensions to the program and runtime syntax in Fig. 6.6. Changes from Chapter 5 are highlighted in yellow.

Commands

output id	ID	::=	n
cmds	c	::=	$\cdots \mid \mathsf{let}\, x = \mathsf{IN}()\, \mathsf{in}\, c \mid \mathsf{output}(ID, e)$
			$\mid \mathsf{if}\, e@q\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \mid x := e@q \mid \mathsf{output}(ID, e@q) \mid \mathsf{end}_{\mathsf{if}}$
idem. output ids	O	::=	$\cdot \mid ID, O$
programs	p	::=	$\cdots \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol, O); c; \mathsf{end}_{\mathsf{atom}}; p$

Program policies and qualifiers

atom. reg. policy	$aPol$	::=	$(\omega, nIds)$
access qualifier	$qAcc$	::=	$\mathsf{CK} \mid \mathtt{Wtn} \mid \mathtt{NRD} \mid \mathtt{mayFstRD} \mid \mathtt{MTntWt} \mid \mathtt{mayTntWt}$
idem. qualifier	qID	::=	$\mathsf{Id} \mid \mathsf{Id}(qAcc) \mid \mathsf{Nid}$
taint qualifier	qIO	::=	$\mathtt{Tnt} \mid \mathtt{Nt}$
qualifier	q	::=	$\langle qID, qIO \rangle$

Instructions, statements, and configurations

prog. counter	pc	::=	$\langle qID, qIO \rangle$
pc. stack	pcS	::=	$\cdot \mid pc \triangleright pcS$
crash instrs	i	::=	$\varepsilon \# \mathsf{in}(b > 0, \downarrow\uparrow\kappa\ @pcS) \mid \downarrow\uparrow\kappa\ @pcS$
nonvolatile mem	N	::=	$\cdot \mid \ell\ @\ q \hookrightarrow v, \mathsf{N} \mid \ell_{\mathsf{ck}}\ @\ \langle \mathsf{Id}(\mathsf{CK}), qIO \rangle \hookrightarrow v, \mathsf{N}$
volatile mem	V	::=	$\cdot \mid \ell\ @\langle \mathsf{Id}, qIO \rangle \hookrightarrow v, \mathsf{V} \mid \ell\ @\langle \mathsf{Nid}, qIO \rangle \hookrightarrow v, \mathsf{V}$
input stream	I	::=	$\mathsf{in}(v) :: \mathsf{I}$
output buffer	O	::=	$\cdot \mid \mathsf{O} :: (\mathsf{out}(ID, v))$
open config	C_o	::=	$(\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K}_{undo} \mid pcS_o \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{K}_{undo} \mid pcS_o \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{K}_{undo} \mid p)$
execution mode	$\mathtt{Md}$	::=	$\mathsf{aID}(c, O)$

Figure 6.6: Summary of syntactic changes from Chapter 5

As before, atomic regions are identified by a unique identifier aID and are equipped with a variable policy *aPol* for the region. In this system, the atomic region variable policy simply includes the sets of checkpointed ω and non-idempotent *nIds* variables. Though unlike our previous systems, a programmer need not specify additional variable sets depending on their anticipated accesses because access restrictions on variables are learned throughout type checking, as described previously. To support idempotent and non-idempotent outputs, a programmer will also annotate an atomic region with a set of output channels O, each uniquely identified by an *ID*, that are restricted to being idempotent, e.g., `water_crop`. The atomic region mode is extended

with O to be referenced throughout program execution and type checking.

To denote reading sensor inputs and outputs in a program, commands are extended to include let bindings of sensor inputs to volatile variables $\mathsf{let}\ x = \mathsf{IN}()\ \mathsf{in}\ c$ and an explicit output command $\mathsf{output}(ID, e)$ that includes an identifier ID of a channel and an expression to send along it. To aid in evolving qualifiers of locations at the operational semantic level, we introduce runtime commands whose expressions are tagged with their qualifiers, in addition to $\mathsf{end}_{\mathsf{if}}$ for detecting the end of a conditional's execution.

Configurations are extended with a buffer O to store the results of idempotent outputs, and a stream I of input values, each of the form $\mathsf{in}(v)$. To track the taint and idempotence level of the command or expression that is running (or being checked), configurations additionally include a stack of pc, each of the form $\langle qID, qIO\rangle$, where each pc comes from expressions and variables that are annotated with a tuple of idempotence and taintedness qualifiers that evolve over the course of type checking. This information will be useful for handling branches with tainted or non-idempotent conditional expressions. A stack is needed to enforce proper scoping for nested conditionals. Lastly, the continuation in a crash instruction is annotated with pcS_o, which must be remembered in order to properly resume execution after failure.

6.4.1 Operational Semantic Rules

We show a representative selection of operational semantic rules in Fig. 6.7. The full set of rules can be found in Appendix D.2.

Top-level Atomic Region Execution. At the beginning of a program execution, the pc stack is initialized to $\langle \mathsf{Id}, \mathtt{Nt}\rangle$. For atomic regions, the function $\mathsf{InitWorld}_d$ is redfined according to the new definition of $aPol$; since only checkpointed or non-idempotent variables are declared in the policy, $\mathsf{InitWorld}_d$ sets the qualifier of these variables in these sets to $\langle \mathsf{Id}(\mathtt{CK}), \mathtt{Nt}\rangle$ and $\langle \mathsf{Nid}, \mathtt{Nt}\rangle$, respectively. All other variables are assigned the (initial) qualifier $\langle \mathsf{Id}(\mathtt{NRD}), \mathtt{Nt}\rangle$.

The atomic region command is evaluated on these contexts and an initially empty output buffer O, which will be populated over the course of execution. The buffered idempotent outputs will be processed at the end of the atomic region where the program can no longer re-execute. This is modeled by adding a premise $\mathsf{send}(\mathsf{O}, n')$ to empty O after the atomic region's execution. We assume that there is enough energy at the end of the execution to process the entire output buffer ($n' > 0$), which the function enforces by returning an energy level $n'' > 0$.

Command Execution. The rules D-Buffer-Id-aid and D-Output-Nid-aid execute the output command in an atomic region whose expression is fully evaluated to a value. If an output is non-idempotent ($ID \notin O$) then it processes instantaneously. Otherwise, the output is declared to be idempotent, in which case it is buffered in O. The rule D-Input models reading a sensor input $\mathsf{in}(v)$ from the stream of sensor inputs, and storing its value v in a fresh location ℓ that is tagged with the qualifier $\langle \mathsf{Id}, \mathtt{Tnt}\rangle$ joined with the current pc.

The rules D-If-tt and D-If-end highlight how the pcS_o changes when executing a conditional. If the conditional expression is tt, the rule D-If-tt applies; the context is raised to the join of the qualifier q_v and the current pc and pushed to the top of the pcS_o. The runtime command $\mathsf{end}_{\mathsf{if}}$ is inserted into the execution to mark the end of the execution of c_1. The rule for executing an else branch is similar. At the end of a conditional (D-If-end), as detected by $\mathsf{end}_{\mathsf{if}}$, the pc is popped

D-P-Ckpt

$$\frac{\begin{array}{c} n, n', n'' > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{K}_{undo} \mid \gamma_0 \\ [\chi \rhd \varepsilon] \otimes \gamma_0 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{I} \mid \cdot \mid \mathsf{N}_0 \mid \cdot \mid \mathsf{K}_{undo} \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \mid c_0 \\ \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_0 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{I}' \mid \mathsf{O} \mid \mathsf{N}' \mid \cdot \mid \mathsf{K}_{undo} \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \mid \mathsf{skip} \\ \mathsf{FinWorld}_d(\mathsf{N}') = \mathsf{N}'' \qquad \mathsf{send}(\mathsf{O}, n') = n'' \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \cdot \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p \;\Rightarrow_p\; [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{I}' \mid \mathsf{N}'' \mid \cdot \mid p}$$

D-Input

$$\frac{\gamma' = \gamma, [x \mapsto \ell] \qquad \ell \, fresh \qquad n = n' + 1 \qquad \mathsf{I} = \mathsf{in}(v)::\mathsf{I}'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{let}\, x = \mathsf{IN}()\, \mathsf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{N} \mid \mathsf{V}, (\ell @ \langle \mathsf{Id}, \mathtt{Tnt} \rangle \hookrightarrow v) \mid pcS_o \mid c}$$

D-If-tt

$$\frac{n = n' + 1 \qquad Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{tt}\ @q_v) \qquad pc' = pc \sqcup_q q_v \qquad pcS_o = pc \rhd pcS^0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{if}\,\mathsf{tt}\ @q_v\ \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pc' \rhd pcS_o \mid c_1 ; \mathsf{end}_{\mathsf{if}}}$$

D-If-end

$$\frac{n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pc' \rhd pcS_o \mid \mathsf{end}_{\mathsf{if}} \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{skip}}$$

D-Assign-N

$$\frac{\begin{array}{c} Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e\ @q) \qquad q = \langle qID_e, qIO_e \rangle \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{N} = \mathsf{N}', \ell @ \langle qID, qIO \rangle \hookrightarrow v' \qquad pcS_o = \langle qID_{pc}, qIO_{pc} \rangle \rhd pcS'_o \\ qID' = \delta(pc, qID, Wt) \neq UN \qquad q' = \langle qID', qIO_e \sqcup_t qIO_{pc} \rangle \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid x := e@q \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{N}', \ell @ q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o \mid \mathsf{skip}}$$

D-Buffer-Id-aid

$$\frac{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid v@q) \qquad ID \in O \qquad n = n' + 1 \qquad \mathsf{O} = \mathsf{O}'::\mathsf{out}(ID, v)}{\gamma \mid \mathsf{aID}(c_0, O) \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{output}(ID, v@q) \to \gamma \mid \mathsf{aID}(c_0, O) \mid n' \mid \mathsf{I} \mid \mathsf{O}' \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{skip}}$$

D-Output-Nid-aid

$$\frac{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid v@q) \qquad ID \notin O \qquad n = n' + 1}{\gamma \mid \mathsf{aID}(c_0, O) \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{output}(ID, v@q) \overset{obs(ID,v)}{\longrightarrow} \gamma \mid \mathsf{aID}(c_0, O) \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{skip}}$$

Figure 6.7: Selected representative operational semantic rules for programs with I/O

from the stack.

The rule D-Assign-N stores a value e in the nonvolatile memory location of a variable x. To carry the taint of both the expressions and the context under which the write occurs, the taint qualifier of x gets updated to the join of the taint qualifier of e and the top pc. To reflect that the location is written to, the idempotence qualifier of x is additionally evolved according to δ. This

transition is not needed for volatile memory, where qualifiers do not contain an interior access qualifier.

Like previous chapters, the full operational semantics are capable of executing conditionals, assignments, and outputs with expressions that are not necessarily values. Though, since the crash types calculus relies on a small-step operational semantics for expressions in order to decrement the energy level on each step, the *pc* level for an expression must be computed gradually with each step. The method of computing these qualifiers for expressions and values is left to the Appendix D.2. The operational semantic rules for closed configuration command execution are similar to before, except that on crash, pcS_o is stashed inside the crash instruction to be recovered when energy is restored. In aID mode, since atomic regions re-execute from the beginning, the pcS_o is reset to $\langle \mathsf{Id}, \mathtt{Nt} \rangle$.

6.5 Type System

This section presents highlights of the type system for inputs, outputs and nonidempotence. The full type system definition can be found in Appendix D.3

Typing judgments. We discuss the key changes to typing judgments needed for checking programs with I/O. We show the augmented typing judgments in Table 6.1. First, unstable typing judgments for commands are extended with an unstable post-context Σ' to reflect the change in taint qualifiers that could arise when checking commands. All expression typing judgments are extended to include a stable post-context Ω'. As in the systems from Chapters 4 and 5, the purpose of the post-context is to reflect changes in access qualifiers, except that here, since access qualifiers can evolve on read, the RD typing judgments have it too. The expression typing judgments are extended with a qualifier q indicating the *pc* level of the expression. All typing judgments track a pc stack at the type level, denoted pcS_t.

Store Typing. The store typing definition is extended to type the operational semantic pc stack pcS_o with respect to the typing pc stack pcS_t. We extend the well-formedness definition with an additional rule for checking the *pc* stack. At a given step in the program, the pcS_o at the operational semantic level will match the pcS_t at the typing level.

PC

$$\frac{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pc \rhd pcS_o : \Omega \mid \Sigma \mid pc \rhd pcS_t}$$

6.5.1 Typing Rules

In this section, we discuss selected typing rules for I/O and non-idempotence.

Program Typing. The rule T-P-Ckpt for checking an atomic region is defined below, where the pc stack is initialized to $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ to match the dynamic rule. The rule is extended to check for RIO bugs at the end of an atomic region, where in a correct program there should be no variables that are written (but not on all paths) in a tainted context (i.e., that have qualifier mayTntWt).

cmd.	(J_u)	$\texttt{Md} \mid b\mathcal{R}0 : \texttt{nat} \mid \Omega;\Sigma \mid pcS_t$	$\vdash_{\texttt{Sig}} c$	$: \mathsf{C}_{\texttt{unit}}$	$\dashv \Omega' ; \Sigma'$	c could crash
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega;\Sigma \mid pcS_t$	$\vdash_{\texttt{Sig}} \mathsf{skip}$	$: \downarrow\uparrow\texttt{unit}$	$\dashv \Omega' ; \Sigma'$	c will not crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega \mid pcS_t$	$\vdash_{\texttt{Sig}} \mathsf{skip}$	$: \uparrow\texttt{unit}$	$\dashv \Omega'$	after commit
expr.	(J_u)	$\texttt{Md} \mid b\mathcal{R}0 : \texttt{nat} \mid \Omega;\Sigma \mid pcS_t$	$\vdash_{\texttt{RD;Sig}} e$	$: \mathsf{C}_A^{\texttt{Md}} @q$	$\dashv \Omega'$	e read, could crash
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega;\Sigma \mid pcS_t$	$\vdash_{\texttt{RD;Sig}} v$	$: \downarrow\uparrow A @q$	$\dashv \Omega'$	e read no crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega \mid pcS_t$	$\vdash_{\texttt{RD}} v$	$: \uparrow A @q$	$\dashv \Omega'$	e read, commit
	(J_u)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega;\Sigma \mid pcS_t$	$\vdash_{\texttt{WT}} x$	$: \downarrow\uparrow A @q$	$\dashv \Omega'$	write on x, no crash
	(J_s)	$\texttt{Md} \mid b : \texttt{nat} \mid \Omega \mid pcS_t$	$\vdash_{\texttt{WT}} x$	$: \uparrow A @q$	$\dashv \Omega'$	write on x, commit

Table 6.1: Typing judgments for commands and RD/WT expressions

To set up the typing contexts, the $\mathsf{InitWorld}_t$ function now only takes sets of checkpointed and non-idempotent variables ω and *nIds*, and sets the qualifiers of these variables to $\mathsf{Id}(\texttt{CK})$ and Nid, respectively. All other variable qualifiers are initialized to $\mathsf{Id}(\texttt{NRD})$, meaning that they are idempotent because they have not been read yet.

T-P-CKPT

$$\frac{\begin{array}{c}\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0 \\ \texttt{Sig} = \{\mathsf{aID}(c_0) \mid b \geq 0 : \texttt{nat} \mid \Omega_0;\Sigma_0 \mid \langle \mathsf{Id}, \texttt{Nt}\rangle \vdash c_0 : \mathsf{C}_{\texttt{unit}} \dashv \Omega' ; \Sigma'\} \\ \mathsf{aID}(c_0) \mid b \geq 0 : \texttt{nat} \mid \Omega_0;\Sigma_0 \mid \langle \mathsf{Id}, \texttt{Nt}\rangle \vdash_{\texttt{Sig}} c_0 : \mathsf{C}_{\texttt{unit}} \dashv \Omega' ; \Sigma' \\ \Omega' \lceil \{\texttt{mayTntWt}\} = \emptyset \qquad b : \texttt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\texttt{unit}}\end{array}}{b : \texttt{nat} \mid \Omega \vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\texttt{unit}}}$$

Definition 9 $\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0$ *iff*

- $\omega \cup nIds \subseteq \mathsf{dom}(\Omega)$,
- $\omega \cap nIds = \emptyset$,
- $\Sigma_0 = \{x_{\texttt{ck}} : \downarrow\uparrow A @ \langle \mathsf{Id}(\texttt{CK}), \texttt{Nt}\rangle \mid x : \uparrow A @ \langle \mathsf{Id}, qIO\rangle \in \Omega^c\}$
- $\Omega_0 = \{x{:}\uparrow A @ \langle \mathsf{Nid}, \texttt{Nt}\rangle | x{:}\uparrow A @ \langle \mathsf{Nid}, qIO\rangle \in \Omega^n\}$
  $\cup \{x{:}\uparrow A @ \langle \mathsf{Id}(\texttt{NRD}), \texttt{Nt}\rangle | x{:}\uparrow A @ \langle \mathsf{Id}, qIO\rangle \in \Omega^{nrd}\}$
  $\cup \{x{:}\uparrow A @ \langle \mathsf{Id}(\texttt{CK}), \texttt{Nt}\rangle | x{:}\uparrow A @ \langle \mathsf{Id}, qIO\rangle \in \Omega^c\}$

where $aPol = (\omega, nIds)$ *and* $\Omega = \Omega^c, \Omega^n, \Omega^{nrd}$ *and* $\Omega^c = \Omega\lceil\omega$ *and* $\Omega^n = \Omega\lceil nIds$.

Next, we discuss the typing rules for commands and expressions, of which we provide a representative selection in Fig. 6.8.

Expression typing rules. Constant values, e.g., tt, ff, and n, are all typed as $\langle \mathsf{Id}, \texttt{Nt}\rangle$ since their value is the same on each re-execution (i.e., idempotent) and that value does not depend on sensor inputs or come from variables (i.e., non-tainted). We summarize their typings in the rule T-VALUE.

Variables whose locations store values, however, are flexible in their qualifiers and evolve based on their accesses. For instance T-Loc-READ types a variable that is being read by looking up its type and evolving its idempotence qualifier via δ according to *pc* and *Rd*. If the result is defined, the post-context and type annotation are updated with the result. Otherwise, the type

T-VALUE

$$\frac{v \in \{\mathsf{tt}, \mathsf{ff}, \mathsf{n}\}}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathtt{RD}} v :\uparrow \mathtt{bool}\ @\langle \mathsf{Id}, \mathtt{Nt}\rangle \dashv \Omega}$$

T-LOC-READ

$$\frac{\Omega = \Omega', x : \uparrow A@\langle qID_x, qIO_x\rangle \qquad pcS_t = pc \rhd pcS'_t \qquad qID' = \delta(pc, qID_x, Rd) \neq UN}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathtt{RD}} x : \uparrow A@q' \dashv \Omega', x : \uparrow A@\langle qID', qIO_x\rangle}$$

T-LOC-WRITE

$$\frac{\Omega = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega' \qquad pcS_t = pc \rhd pcS'_t \qquad qID' = \delta(pc, qID_x, Wt) \neq UN}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathtt{WT}} x : \uparrow A@q' \dashv \Omega', x : \uparrow A@\langle qID', qIO_x\rangle}$$

T-BINARY

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1}\ @q_1 \dashv \Omega_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2}\ @q_2 \dashv \Omega_2 \qquad \odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_{T'}\ @q_1 \sqcup_q q_2 \dashv \Omega_1 \sqcup \Omega_2}$$

T-LET-IN

$$\frac{pcS_t = pc \rhd pcS'_t \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}\downarrow\uparrow A@\langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{let}\, x = \mathsf{IN}()\, \mathsf{in}\, c : \tau \dashv \Omega'; \Sigma'}$$

T-IF

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}}\ @\langle qID_e, qIO_e\rangle \dashv \Omega_0; \Sigma_0 \qquad pcS_t = \langle qID, qIO\rangle \rhd pcS'_t \\ pc' = \langle qID \sqcup_{id} qID_e, qIO \sqcup_t qIO_e\rangle \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \mid pc' \rhd pcS_t \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega_1 ; \Sigma_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \mid pc' \rhd pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega_2 ; \Sigma_2 \qquad \Omega' = \mathsf{lift}(pc, \Omega_1 \sqcup \Omega_2)\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 : \tau \dashv \Omega'; \Sigma_1 \sqcup \Sigma_2}$$

T-ASSIGN-S

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{A} @\langle qID_e, qIO_e\rangle \dashv \Omega^1; \Sigma^1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega'; \Sigma' \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \\ x \in \mathsf{dom}(\Omega) \qquad \Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0 \qquad \Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega''; \Sigma'}$$

T-OUTPUT-ID

$$\frac{\mathsf{aID}(c_0, O) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : A@\langle qID_v, qIO_v\rangle \dashv \Omega'; \Sigma' \qquad ID \in O \qquad \overline{qID_v} \sqsubseteq_{id} \mathsf{Id}}{\mathsf{aID}(c_0, O) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{output}(ID, e) : \tau \dashv \Omega'; \Sigma'}$$

Figure 6.8: Selected command typing rules for I/O and non-idempotence

checking will fail. On the other hand T-Loc-Write performs the analogous check for a variable being written, producing the appropriate qualifier if the write is allowed. Lastly, T-Binary checks a binary expression $e_1 \odot e_2$ by checking each expressions separately to get a qualifier and a post-context, and then joining these results.

Command typing rules. The rule T-Let-In checks a let binding of sensor input, enforcing that the bound variable x has the type $\langle \mathsf{Id}, \mathtt{Tnt} \rangle$ joined with the current pc when checking the command c, which matches the dynamic rule.

The rule T-If checks a conditional by first checking the conditional expressions e to determine its qualifiers and post-contexts, which the rule uses to check each branch separately. The post-contexts serve as the initial contexts of these judgments. Like the dynamic rule, the pc context is raised according to the qualifier of e and is pushed to the pc stack pcS_t. The post-contexts of checking these branches are joined together (denoted $\sqcup$), has the effect of selecting the most restrictive access and taint qualifiers between these branches according to $\sqcup_{id}$ and $\sqcup_t$ (see Appendix D for the full definition). If exiting a `Tnt` context, the lift function changes all `MTntWt` qualifiers in Ω' to `Wtn` (see Appendix D for the full definition).

The rules for checking an assignment $x := e$ enforce that no non-idempotent data from e or the pc could flow to idempotent variables ($qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x$). If this condition is met, the rules propagate taint from expression e and the context pc to the assigned variable x. The rule T-Assign-s checks an assignment to a variable that types as stable, and updates the taintedness qualifier of x to reflect the assignment to e under the context pc ($qIO_{pc} \sqcup_t qIO_e$). The access qualifier of x evolves in another rule (T-Loc-Write) when checking the `WT` judgment. Like the other command typing rules, the post-contexts from checking e are threaded through into the initial typing context for checking x to inform its checking. For example, a program $x := x + 1$ where x is neither checkpointed nor non-idempotent would be rejected since x starts with the access mode `NRD`, and evolves to `mayFstRD` when checking its read, which is then used to check the write to x. Since a write on `mayFstRD` is prohibited by δ, the type checking would fail. The checking rule for assignment to an unstable variable is similar, concerning Σ instead.

The rule T-output-id types an output command over an idempotent channel, enforcing that values (or expressions) sent are solely idempotent. The rule for checking non-idempotent outputs is analogous, but allowing outputs to be either idempotent or non-idempotent.

6.6 Metatheory

In this section, we state the key theorems of our combined calculus for I/O, and speculate about the proofs of logical relation and adequacy.

6.6.1 Progress and Preservation

Since the progress theorems do not change substantially, we focus on preservation for commands and provide intuitions for its supporting lemmas. All theorems and proofs can be found in Appendix D.4.

Theorem 9 (Preservation for Commands) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'$$

and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid c$ *is well-formed,* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o : \Omega \mid \Sigma \mid \mathit{pcS}_t$*, and for some (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid c \ \rightarrow \ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}'_o \mid c'$$

then for some Σ_0 *and* Ω_0

$$\mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega_0; \Sigma_0 \mid \mathit{pcS}'_t \vdash_{\mathtt{Sig}} c' : \tau \dashv \Omega'; \Sigma'$$

where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}'_o : \Omega_0 \mid \Sigma_0 \mid \mathit{pcS}'_t$*,* $\Omega \preccurlyeq \Omega_0$*,* $\Sigma \preccurlyeq \Sigma_0$*, and* $n' \geq 0$*. Moreover* $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}'_o \mid c'$ *is well-formed.*

Theorem 9 ensures that well-typed commands that can take a step will step to a well-typed command in a well-formed configuration. As before, to support evolving access qualifiers that may occur during the step from c to c', the initial typing context in the typing judgment of c' varies from the initial context of c. However, since the taint qualifiers of variables can also change at the command level, both stable and unstable typing contexts change. The way in which they differ from the initial contexts Ω and Σ is characterized by Definition 10.

We say that a context Ω' is *reachable* from Ω (denoted $\Omega \preccurlyeq \Omega'$) if there exists a series of sequential accesses (δ) and join operations ($\sqcup$) on idempotence qualifiers between them. The δ transitions on read and write are accounted for by (i) and (ii), and join is accounted for by (iii). Since idempotence qualifiers in the unstable typing context do not change, reachability on Σ is defined where the idempotence qualifiers are the same. Since taint qualifiers, by nature, can be replaced throughout type checking we do not reason about them here.

Definition 10 (Reachability) $\Omega \preccurlyeq \Omega'$ *iff there exists* Ω_0 *s.t.* $\Omega_0 \preccurlyeq \Omega'$ *and either*

(i) $\Omega \preccurlyeq_{Rd} \Omega_0$ *or*

(ii) $\Omega \preccurlyeq_{Wt} \Omega_0$ *or*

(iii) for all $x : \tau@\langle qID_x, qIO_x\rangle \in \Omega$ *it is the case that* $x_0 : \tau@\langle qID_0, qIO_0\rangle \in \Omega_0$ *s.t.* $qID_x \sqsubseteq_{id} qID_0$

We write $\Sigma \preccurlyeq \Sigma'$ *iff for all* $x : \tau@\langle qID_x, qIO_x\rangle \in \Omega$ *it is the case that* $x' : \tau@\langle qID', qIO'\rangle \in \Omega'$ *s.t.* $qID_x = qID'$

We define $\Omega \preccurlyeq_{Wt} \Omega_0$ similarly to Chapter 4 but according to the modified definition of δ. We define $\Omega \preccurlyeq_{Rd} \Omega_0$ similarly, but for read; we write $\Omega \preccurlyeq_{Rd} \Omega_0$ iff for every variable typing in Ω, it is the case that either its type in Ω_0 is the same or differs by a single δ transition on *Rd*. These relations are obtained from Theorem 10 and Lemma 1 for the judgments `RD` and `WT`, respectively, and are stitched together into a series of transitions at the command level. Since the operational semantic rules for expressions and method of computing qualifiers over expressions were not stated in the main body of this chapter, we just state a simplified version of these lemmas below. At a high level, since access qualifiers evolve when reading an expression or writing to a variable, the initial stable typing context of a stepped expression is shifted in a similar manner to preservation for commands.

Theorem 10 (Preservation for Expressions) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{RD;Sig}} e : \tau@q \dashv \Omega'$$

and for some $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o : \Omega \mid \Sigma \mid \mathit{pcS}_t$ *and natural number* $n \geq 0$

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid e \ \rightarrow \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid \mathit{pcS}_o \mid e'$$

then there exists a Ω_0 *such that*

$$\mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega_0; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{RD;Sig}} e' : \tau@q \dashv \Omega'$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid \mathit{pcS}_o : \Omega_0 \mid \Sigma \mid \mathit{pcS}_t$ *and* $n' \geq 0$*, and* $\Omega \preccurlyeq_{Rd} \Omega_0$.

Lemma 1 (Write Reachability) *If*

$$\mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \ \mid \ \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@q \dashv \Omega'$$

then $\Omega \preccurlyeq_{Wt} \Omega'$.

6.6.2 Logical Relation and Adequacy

We conjecture that building a logical relation for reasoning about correctness in this system would involve extending a logical relation for undo-logging with considerations for the new qualifiers. In particular, it would rely on modified policies for `PwOff`, `Commit`, and `Restore`, shown in Fig. 6.9:

$$\frac{\gamma' \subseteq \gamma \ s.t. \ \mathit{range}(\gamma') = \mathsf{dom}(\mathsf{N}_1) \qquad \mathsf{V}_2 = \emptyset}{\mathtt{PwOff}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{\mathsf{Id}(qAcc)}, \mathsf{N}''_{\mathsf{un}}, \mathsf{N}^3_{\mathsf{Nid}} \qquad \mathsf{N}_2 = \mathsf{N}'_{\mathsf{Id}}, \mathsf{N}''_{\mathsf{Id}}, \mathsf{N}^3_{\mathsf{Nid}} \qquad \gamma' \subseteq \gamma \ s.t. \ \mathit{range}(\gamma') = \mathsf{dom}(\mathsf{N}_2)}{\mathtt{Commit}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1) = \gamma' \mid \mathsf{N}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{\mathsf{Id}(qAcc)}, \mathsf{N}''_{\mathsf{un}}, \mathsf{N}^3_{\mathsf{Nid}} \qquad qAcc \neq \mathtt{CK} \qquad \mathsf{K} = (\mathsf{N}_k) \qquad \mathsf{N}_2 = \mathsf{N}'_{\mathsf{Id}(\mathtt{NRD})}, \mathsf{N}_k, \mathsf{N}^3_{\mathsf{Nid}}}{\mathtt{Restore}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{K}, \kappa) = \mathsf{N}_2 \mid \kappa}$$

Figure 6.9: `Commit`, `Restore`, and `PwOff` policy definitions for (non-)idempotence and I/O

As before, the `PwOff` policy clears volatile memory and drops the corresponding mappings from γ. At the end of an atomic region, to prepare for executing the next region, the `Commit` policy flattens the idempotence qualifiers to Id. Lastly, `Restore` resets the access qualifier of all noncheckpointed variables to `NRD`, while keeping their taint qualifiers the same.

The term relation would remain the same as before, while the value relation must change to relate intermittent and continuously-powered states throughout power failure and restoration. In

particular, after state is restored, we must show that the nonvolatile memories are equivalent in their checkpointed state, i.e., excluding variables marked after restore as Id(NRD) or Nid which could hold unstable or non-idempotent data computed by failed prior executions. At the end of an atomic region execution, we must establish that the final memories are equivalent up to their Nid variables, which are explicitly allowed to differ across executions. In order to establish that all tainted values agree by the end of the last successful execution, we rely on the check that there are no remaining mayTntWt variables. In other words, in a correct program, values written on one tainted path but not on all will be overwritten on all paths. This overwrite will enable results from the intermittent execution to match the continuously-powered execution.

Chapter 7

Formal Foundations of Concurrent Intermittent Computing

Up until this point, the previous chapters of this thesis and state-of-the-art designs for intermittent systems [20, 40, 44, 119, 121] have established rich support for provably correct *sequential* intermittent execution, including formal specifications of intermittent systems and correctness proofs. These proofs show that results of (sequential) intermittent execution of a program are equivalent to results of its continuously-powered execution, guaranteeing that sequential programs in high-assurance application domains execute correctly without intermittence corrupting results.

Equivalent support for *concurrent* intermittent systems, however, has lagged. While parallel execution is out of scope for most energy-harvesting embedded platforms, supporting event-driven concurrent execution is important. In this concurrency model, programs experience non-deterministically triggered hardware interrupts that preempt the main application and switch control over to a programmer-defined *interrupt service routine* (or *ISR*), which completes execution before control switches back to main. Many embedded systems applications rely on this concurrency model, using hardware-triggered interrupts to support basic peripheral interaction as well as more complex features like software-scheduled events and pre-emptive multi-tasking. Even implementations of common models of intermittent execution, e.g., JIT regions [13, 14], typically involve the hardware issuing a low-power signal in the form of an interrupt to save volatile state before power fails.

Yet, fully supporting intermittent execution for this concurrency model is challenging because: 1) power failures introduce unaccounted-for control flow paths and dependencies that complicate reasoning about concurrent executions; 2) interrupt triggers and peripheral sensor data are frequently ephemeral and transient, and are potentially unrecoverable after failure; and 3) shared memory accesses break existing checkpoint mechanisms designed for sequential intermittent systems. Due to these challenges, past work has provided only a few concurrent, intermittent systems that tend towards restricted programming models and limited program features [109, 129], e.g., no re-executable code regions, no volatile memory, no reading main memory from ephemeral ISRs which could abort upon failure. *Furthermore, these systems rely solely on informal correctness notions, precluding provable guarantees, even as concurrent applications are even more susceptible to subtle bugs.*

This chapter aims to lay a formal foundation for concurrent, intermittent computing. While many system designs are possible, our goal is to define a general, flexible model of intermittent execution with shared memory concurrency, co-designing a runtime system specification and checking algorithm that together ensure programs will provably execute correctly. Our model overcomes the bespoke limitations of prior systems, laying groundwork for extension to even more expressive system features, while our type system enforces proper shared memory accesses, getting away from the rigid restrictions imposed by prior work. We define a task-based program structure comprised of ISRs and events that access shared memory locations. Our key observation is that correctness of concurrent, intermittent systems is inherently intertwined with reasoning about explicit task *re-execution policies* and their effect on other concurrently executing tasks. While past intermittent systems relied on a notion of atomic re-execution where designated atomic regions re-execute until they complete without failure, this abstraction is insufficient for the concurrent setting, where some partially-executed tasks (e.g., time-sensitive ISRs) are thrown out entirely upon reboot. We thus identify a broader set of *re-execution policies* enabling flexibility in task re-execution that goes beyond simple atomic re-execution. Upon reboot, these policies allow the system to re-execute a task *immediately*, *discard* it completely, or run an *alternate* version. Our key insight is that to enable the correct intermittent execution of concurrent programs with these re-execution policies, writes to shared variables by failed partial task executions—which we call *unstable writes*—must not influence the idempotent results of the computation. This idea extends the notion of noninterference in the sequential setting from Chapter 6 and prior work [121].

Towards correct concurrent, intermittent execution, we introduce Glacier[1], a co-designed runtime system and lightweight type system that together support flexible re-execution policies while rejecting programs that could yield incorrect results when executing concurrently. Glacier builds on the idea of idempotent and non-idempotent types: partial, unstable computations and values may differ across re-executions (i.e., be non-idempotent), whereas completed, stable computations and values are idempotent, and any non-idempotent data must not interfere with idempotent results. To account for shared memory accesses in our concurrent setting, we extend the type of a variable with qualifiers that denote the idempotence of a variable both locally and *globally*, with respect to writes performed by other tasks. Furthermore, we observe that the meaning of unstable in this concurrent setting differs from its meaning in the sequential setting. While in the undo-logging interpretation of crash types for sequential, intermittent computing the modified checkpointed nonvolatile memory locations are unstable and the remainder of nonvolatile memory is stable, here all nonvolatile variables that could be accessed by an atomic region via a write are considered unstable from the perspective of other sharing processes.

Given these types, we design checking rules that account for the influence of task priorities and re-execution policies on dependencies between variables, ensuring that no non-idempotent value flows to an idempotent result even in the concurrent setting. We then prove Glacier satisfies the key correctness invariant:

> *"Every intermittent execution of a well-typed concurrent program has an equivalent continuous execution."*

[1] Glacier: "**G**eneral **l**anguage **a**bstraction for **c**oncurrent **i**ntermittent **e**xecution and **r**e-execution"

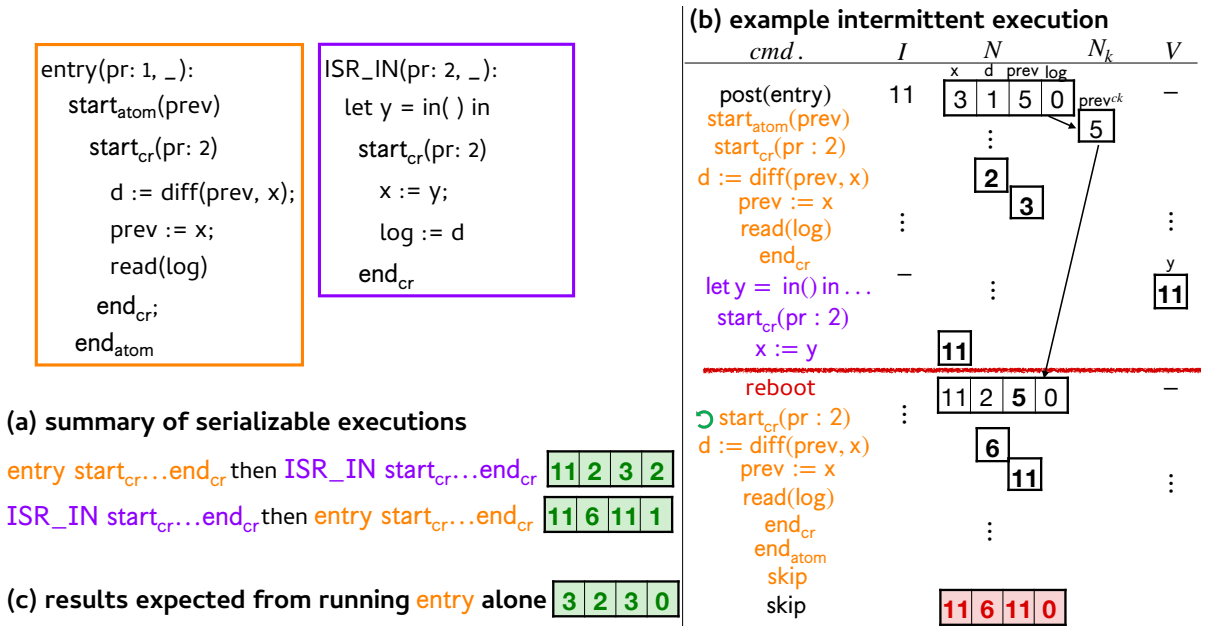

Figure 7.1: Intermittence breaks programs correctly serialized on continuous execution.

The rest of the chapter is structured as follows. Section 7.1 motivates the challenges of reasoning about concurrent intermittent execution with a concrete example, which past techniques are not equipped to handle. Section 7.2 introduces the key ideas behind the techniques needed to properly handle concurrent intermittent programs. Section 7.3 defines the syntax and operational semantics for a provably correct runtime system for concurrent programs, first for continuous program execution (Section 7.3.1) and then for intermittent program execution (Section 7.3.2). Section 7.4 formally defines our static type system, and discusses its implementation. Section 7.5 discusses an extension of the core calculus. Altogether, programs in our system that pass type checking are guaranteed to succeed at runtime, which Section 7.6 states as our key theorem.

7.1 Concurrent, Intermittent Execution by Example

To illustrate the challenges of concurrent, intermittent execution, we present a running example. Fig. 7.1 shows the example code on the left. The goal of the snippet is to process a sensor input and track the difference between the most recently processed inputs, logging the difference when a new input is taken. The work is split between two tasks: an event handler entry that is triggered by a software-scheduled command and an ephemeral interrupt handler ISR_IN that fires whenever the sensor peripheral is ready. Due to the transient nature of the sensor data, ISR_IN must abandon its execution in the event of a power failure, whereas entry can re-execute. Whenever ISR_IN fires, it first stores the input into a private volatile register y, which it then writes to x, and then logs the current value of d in the variable *log*. Meanwhile, the task entry reads x, the most recent input value, and calculates the difference between it and the previously-processed input, *prev*, storing the result in d. It then updates *prev* with x and reads *log*.

While the binding of input in ISR_IN relies only on accesses to *local* variable y, the assign-

ments to x and *log* both access *shared* variables. To prevent data races, all accesses to shared variables in both tasks must be wrapped inside critical sections. To implement synchronization, the system employs the well-established dynamic ceiling priority protocol [1, 113]: each task and critical section is assigned a priority (by the programmer or static analysis), and only a task with a strictly higher priority can preempt the currently-running task. When entering a critical section protecting a shared variable, the priority rises to the ceiling priority of any tasks that access that variable, preventing racing accesses. In this example, the critical section priority of 2 ensures that the enclosed assignments execute without interruption from other tasks accessing shared variables x, d, and *log*; ISR_IN cannot interrupt entry's critical region as its priority is not high enough.

Since all accesses to shared variables are protected by critical sections, this program is *data-race free*. As such, any continuously-powered execution of the two tasks will be *serializable*, i.e., consistent with some serial execution of the critical sections; we show the final states resulting from the two possible orderings in Fig. 7.1 (a). The example again assumes an initial nonvolatile state of $3, 1, 5, 0$. The two serial orderings are that entry's critical section executes first, followed by ISR_IN's, and vice-versa. Assuming entry is first, the critical section sets d to 2, and *prev* to 3. If the interrupt trigger then fires, the handler preempts entry and sets x to the input value 11, and *log* to 2. The handler then returns, allowing entry to continue its execution. The final memory state is $x \hookrightarrow 11, d \hookrightarrow 2, prev \hookrightarrow 3, log \hookrightarrow 2$. If the interrupt fires first, the handler sets x to 11 and *log* to 1. When entry executes, it sets d to 6 and *prev* to 5. The final memory state is thus $x \hookrightarrow 11, d \hookrightarrow 6, prev \hookrightarrow 11, log \hookrightarrow 1$. Any other result, including ones resulting from unserialized interleaving of critical regions, would be both incorrect and impossible!

Unstable values in intermittent execution cause cyclic dependence. Unfortunately, an *intermittent* execution of this program can be unserializable. We show a possible, incorrect intermittent execution in Fig. 7.1 (b). As before, the code in entry is wrapped in an atomic region, where the variable *prev* is checkpointed by the system. Accesses of x, d, and *log* are considered safe in the sequential setting; the write to d would always overwrite updates from partial executions, and x and *log* are read-only. Furthermore, the ISR_IN stores the sensor reading in a private volatile variable that cannot be saved, and accesses the shared variables d, x, and *log* safely (locally), since d is read-only and both x and *log* must be written first.

With the stage set, let us consider an intermittent execution where ISR_IN fires after the critical section of entry has completed, but before the atomic region finishes. Power fails right after the successful completion of the critical section in ISR_IN, resulting in the state $x \hookrightarrow 11, d \hookrightarrow 2, prev \hookrightarrow 3, log \hookrightarrow 0$. Since the atomic region in entry has not yet completed, on reboot, the program re-executes from the start of the atomic region, resetting the value of *prev* to its checkpointed version. Hence, the re-execution reads the values of x and *log* that were updated by the ISR, creating a dependence from ISR_IN to entry. However, the read of d in the ISR already created a dependence from entry to ISR_IN. As a result, this trace has a *happens-before* cycle between entry and the ISR, which makes the execution inherently *un*serializable. We can see this in the final result $x \hookrightarrow 11, d \hookrightarrow 6, prev \hookrightarrow 11, log \hookrightarrow 0$, which is not produced by either serializable execution. Intermittence broke an otherwise data-race free program! We call the writes to x and *log* by the non re-executable task ISR_IN *unstable* writes and the value written by the to-be-discarded ISR_IN an *unstable* value, which caused the incorrect result.

Conflict between unstable values and checkpointing. At first glance, it may appear that this issue could be remedied by extending existing checkpointing techniques to consider potential WAR access patterns across concurrent tasks. However, naively extending analyses and runtime systems of checkpointing intended for sequential programs to concurrent ones could not only break the modularity of the analysis, but also lead to new memory consistency bugs, particularly when the pre-empting thread must also re-execute on failure.

Consider another example in Fig. 7.2. The program code is the same as above, except that ISR_IN is periodically scheduled and its code is wrapped in an atomic region that re-executes after failure. Both tasks checkpoint x, d, and *log* because they have a potential WAR dependence when considering concurrent executions: x and *log* could be WAR variables if the entry critical section runs before ISR_IN, and vice versa for d. That is, all variables accessed by these tasks are checkpointed by them.

Let us consider an execution trace where ISR_IN preempts entry between the completions of the critical section and the atomic region. The atomic region in entry writes the values 2 and 3 to d and *prev*, respectively, before the atomic region in ISR_IN overwrites x to 11. Power fails. On reboot, all saved values are restored from their checkpoints back into memory, and the ISR re-executes first. For simplicity, the sensor returns the value 11 as before, and this time, stores the value 2 in *log*. When the atomic region finishes executing, it commits its values to the checkpoint context *as they are part of the completed, stable execution trace*. Consequently, after ISR_IN returns, the atomic region in entry re-executes on the state $x \hookrightarrow 11, d \hookrightarrow 2, prev \hookrightarrow 5, log \hookrightarrow 2$. Execution ends in another incorrect state $x \hookrightarrow 11, d \hookrightarrow 6, prev \hookrightarrow 11, log \hookrightarrow 2$. The problem is that the write to d by entry before the atomic region completes is *unstable*, and therefore checkpointing at the beginning of the atomic region in ISR_IN allows an unstable value to persist.

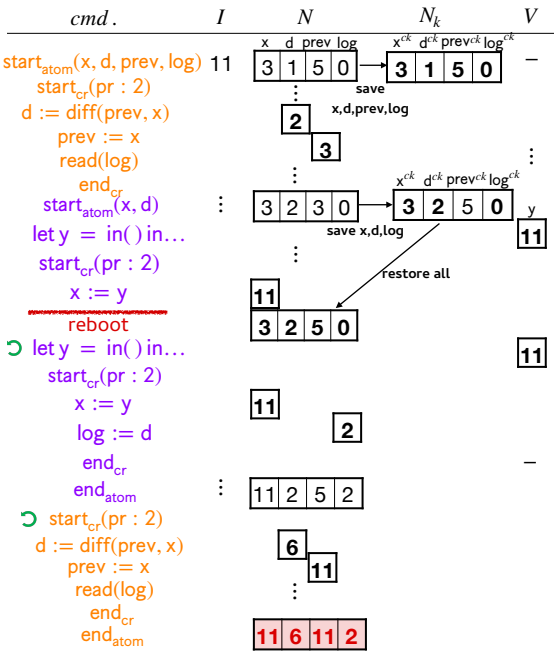

Figure 7.2: Basic checkpointing "fixes" also break.

The motivation behind this work is to design an intermittent system language and runtime to guarantee correct concurrent, intermittent execution. There are multiple ways to approach designing such a system; for example, one could use a type system to disallow this program completely or develop a more flexible runtime system that makes smarter choices about when to update the checkpointed variables. This work explores the former, enforcing that unstable writes do not tamper with stable concurrent computation at the type level to forewarn programmers of such potential issues.

7.2 Key Ideas

Using the example from Fig. 7.1, this section introduces the key ideas of our co-designed type and runtime system that together provably ensure correct concurrent intermittent execution.

Modular programming with flexible re-execution policies. To overcome the limitations of prior work, Glacier supports tasks that adopt a broad spectrum of *re-execution policies*, ranging from those with JIT and atomic regions [40, 44, 119, 121] which continue their execution immediately, to those that are completely discarded (as in ephemeral ISRs), to those that run an alternative (perhaps more energy-efficient [80]) version of the original task. Glacier supports any chaining of re-execution policies, e.g., first attempting to re-execute the original task immediately, and then attempting a series of alternative tasks before finally discarding after too many failures. Sections 7.3- 7.4 focus on the first two, immediate and discard, whose behaviors are fundamental to the core system design, leaving the more expressive alternative to be examined afterward in Section 7.5.

Our modular, task-based programming model with dynamic re-execution policies is intentionally similar to popular frameworks for continuous, interrupt-driven concurrency (e.g., RTIC [1]). A programmer would write their program as a collection of tasks, including hardware-triggered interrupts and software-scheduled events. Each task specification (e.g., for entry and ISR_IN from Fig. 7.1) consists of: an assigned priority number *pr* (e.g., 1 or 2), re-execution policy *rPol* (e.g., immediate or discard), and sets of shared variables *ShAcc* (e.g., $\{x, d, log\}$).

Local and global (non-)idempotence. The examples of incorrect executions in the previous section highlight the impact of *unstable writes*—variables written by tasks that could be discarded due to power failure. A key insight is that unstable writes could cause values of variables to differ across executions, and that to rule out incorrect behaviors, the type system should enforce the policy that unstable values must not interfere with idempotent data. Since non-idempotent data follows a similar invariant, our observation is that we can label the effects of unstable writes as non-idempotent, or Nid. In Fig. 7.1, x is idempotent for entry, but may depend on the unstable (Nid) writes by ISR_IN, therefore violating the policy above.

To statically catch such potential dependencies, our type system describes and reasons about variable (non-)idempotence with respect to both *local* usage within a given task, as well as *global* usage by others. For global usage, we consider writes from tasks with higher or equal priority versus lower priority separately, each interacting with shared variables in different ways; tasks with a higher priority relative to a given task *could preempt* it, thus seeing unstable values generated by preempted tasks, whereas those with a strictly lower priority *could have instead been preempted*, thus revealing unstable writes to their preemptors. Therefore, our types consist of *triples* of idempotence qualifiers $\langle \texttt{qlocal}, \texttt{qlow}, \texttt{qheq} \rangle$, where qualifier `qlocal` represents the local (non-)idempotence of a variable, and `qlow` and `qheq` represent its (non-)idempotence with respect to updates from tasks of lower and greater or equal priority, respectively, from the perspective of the local task. Each task is checked separately under its own typing context, which is built according to local policies, task re-execution policies, and priorities, which we detail in Section 7.4. Since types are determined relative to task priority, the typing context for two tasks may map the same variable to different types. For example, x and *log* both have type $\mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Nid} \rangle$ in the typing context of entry because x is idempotent within entry, there are no tasks with lower

priority than entry (and thus no non-idempotent effects for `qlow`), and the higher priority discard task ISR_IN has an unstable write to x, producing a non-idempotent effect for `qheq`.

Therefore, our static typing must check that non-idempotent data, whether it be coming from local *or* global sources, could not violate local idempotency requirements. For an assignment $x := y$ where x:int@$\langle$ld, $\texttt{qlow}_x$, $\texttt{qheq}_x\rangle$ and y:int@$\langle\texttt{qlocal}_y, \texttt{qlow}_y, \texttt{qheq}_y\rangle$, the typing rules would need to check that none of $\texttt{qlocal}_y$, $\texttt{qlow}_y$, $\texttt{qheq}_y$ is Nid. For example, when type checking entry in Fig. 7.1, the assignment prev := x would fail, as *globally* non-idempotent data from the variable x flows to the locally idempotent variable prev.

A type system and runtime system co-design to ensure correctness. Our correctness definition follows the same principle as existing work [119]: every intermittent execution of a correct program has a corresponding continuously-powered execution. The problem with the program in Fig. 7.1 is that the unstable value written to x by ISR_IN flows to idempotent variables when entry re-executes. We have the choice of disallowing this behavior either by ruling out this program by our type system, or by implementing a runtime system that subverts the effects of aborted discard tasks (e.g., writes to x in ISR_IN). There are many similar scenarios where rejecting programs by our type system is too restrictive and we instead rely on the runtime to ensure correct behavior.

Our observation is *processes that do not successfully finish executing should behave as if they never ran at all, with respect to their effects on idempotent computation*. To erase the effects of x's writes from ISR_IN's failed execution, the runtime system can save a stable version of x in a checkpoint to be reverted upon reboot. More concretely, Glacier relies on *task-level checkpointing*, allowing tasks with discard policy to checkpoint variables at the beginning of their execution. If power fails before such a task finishes executing, its writes will be reverted similar to checkpointing in atomic regions, but unlike atomic regions, the task will *not* re-execute on reboot. Each task specification is further extended with a set of checkpointed shared variables, e.g., x in ISR_IN.

Our runtime system assumes an undo-logging interpretation for atomic regions. As in Chapters 5 and 6, the checkpoint context K_{undo} is denoted explicitly. Though in this system we additionally consider global volatile memory, and therefore K_{undo} also contains checkpointed versions of volatile state. Glacier maintains the invariant that the checkpoint context must *always* contain only the most up-to-date stable (idempotent) data. To do so, the runtime must only update those checkpointed values to be in sync with the current state: after every write in JIT regions (which are stable), and only at the beginning and successful conclusion of atomic region and discard task execution.

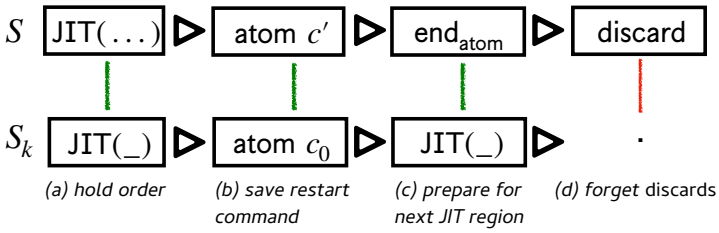

Figure 7.3: Sketch of runtime and checkpointed stacks.

Following prior work [121], Glacier allows variables to differ in value, for instance, to retain information about re-execution state (e.g., *log* which records *every* computed d). Which variables are allowed to differ across re-executions of a program is specified for each task by the

programmer, which we refer to as non-idempotent variables.

Checkpointed execution stack. To properly implement task-based concurrency, our runtime system schedules a *frame* for each invocation of a task, which we call a process. Each frame stores the necessary context such as policies, local variable mappings, and execution context. To remember which processes should resume execution after failure, and in what order, Glacier's checkpoint context additionally includes a *checkpointed stack*, denoted S_k. Upon reboot, S_k will be used as the new execution stack. Similar to checkpointed memory, Glacier maintains the invariant that S_k only includes frames for resumable processes. We explain the use cases of S_k in Fig. 7.3 in relation to each frame on S. The top of the figure shows an execution stack S with four different processes, the leftmost of which is the currently executing frame. For each frame pair, the frame on the left preempted the frame on the right. In the figure, S includes frames running (a) a JIT process, (b) an atomic region command c', (c) the end of an atomic region, and (d) a discard process that should abort on failure. When frames are pushed to the stack, the runtime builds a checkpointed counterpart in S_k, shown on the bottom of the figure and each pair connected by a green line.

The first three processes which are running JIT and atomic regions, which live in immediate tasks, will be recovered on reboot, and thus have checkpointed counterparts in S_k. Since JIT regions re-execute from the point where power failed, and the runtime saves state right before this failure, each JIT process on S (Fig. 7.3 (a)) has a corresponding *placeholder frame* to hold its position relative to other processes in S_k that will be replaced with its frame from S right before power failure. It can also be replaced by an atomic region frame (Fig. 7.3 (b)) when an atomic region executes for the first time to remember from where to re-execute. When the atomic region completes (Fig. 7.3 (c)), the atomic region frame is no longer needed and is thus removed from S_k. At this point, the runtime prepares for executing the next JIT region by replacing the JIT placeholder frame, as described in (a). On the other hand, the last frame (Fig. 7.3 (d)) is a discard task, which does not resume execution on reboot. For this reason, it has no corresponding frame in S_k.

7.3 Syntax and Operational Semantics

This section introduces the syntax and operational semantics of our core calculus for interrupts. We first provide a basic operational semantics for interrupts and critical regions, and then an intermittent, concurrent operational semantics that accounts for non-deterministic power failures. For ease of explanation, we present simplified versions of the formalism. Differences between the formalism presented here and the full formalism are summarized in Appendix E.1. The full syntax and operational semantics can be found in Appendices E.2, E.3, and E.4.

7.3.1 Continuously-Powered Execution

Syntax. The simplified syntax is shown in Fig. 7.4. Values and expressions are standard. Commands are extended with critical sections, denoted $\mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}}$, where pr is its priority and $ShAcc$ is the set of shared variables to be accessed by the enclosed command c. Each

Program syntax

Values	v	::=	$n \mid \mathsf{true} \mid \mathsf{false}$
Commands	c	::=	$\mathsf{skip} \mid x := e \mid \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{let}\ x = e\ \mathsf{in}\ c \mid \mathsf{let}\ x = \mathsf{in}()\ \mathsf{in}\ c$ $\mid c_1; c_2 \mid \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \mid \mathsf{ret}$
Expressions	e	::=	$x \mid v \mid e_1 \odot e_2 \mid \mathsf{uop}\, e$
Int. Srv. Rt.	isr	::=	$\mathsf{isr}(ID, pr, ShAcc, \lambda x.c)$

Runtime syntax

Interrupt	irq	::=	$\mathsf{interrupt}(ID, v)$
Exec. ctx.	E	::=	$\cdot \mid \mathsf{end}_{\mathsf{if}} \triangleright E \mid \mathsf{end}_{\mathsf{cr}} \triangleright E \mid ([\,]; c) \triangleright E$
Interrupt queue	IRQ	::=	$\cdot \mid irq :: IRQ$
Local vars	υ	::=	$\cdot \mid \upsilon, x \mapsto (v, pID)$
Inputs	I	::=	$\cdot \mid \mathsf{in}(v) :: I$
Priorities	$\overrightarrow{pr}$	::=	$\cdot \mid pr \triangleright \overrightarrow{pr}$
Stack	S	::=	$\cdot \mid F \triangleright S$
Frame	F	::=	$(pID, ID, \overrightarrow{pr}, ShAcc, \upsilon, E, c)$
Queue	Q	::=	$\cdot \mid irq :: Q$
Nonvol. mem.	N	::=	$\cdot \mid \mathsf{N}, x \mapsto (v, pID)$
Vol. mem.	V	::=	$\cdot \mid \mathsf{V}, x \mapsto (v, pID)$
Cts. Config	C^{∞}	::=	$(I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q)$

Figure 7.4: Summary of syntax for programs and continuously-powered runtime system.

variable is tagged with its access mode, wt ("*writable*") or rd ("*read-only*"). Interrupt service routines are denoted *isr*, which is a tuple of a unique interrupt identifier *ID*, its priority *pr*, a list of shared variables *ShAcc* to be accessed, and the function to be executed $\lambda x.c$, where x will be bound to the value of the triggering interrupt. Finally, ret is the last command of an *isr*. A program is a collection of interrupt handlers *IDecl* with a dedicated entry point (e.g., the main function to run).

Runtime constructs for defining the operational semantics appear below the line in Fig. 7.4. A continuously-powered runtime configuration C^{∞} is similar to previous chapters but extended with a stream of incoming interrupts *IRQ*, the currently-executing code enveloped in a frame F, a stack S of frames containing preempted processes, and a queue Q of deferred interrupts whose priority is insufficient to trigger execution.

Each interrupt (*irq*) is composed of a pair of the interrupt *ID* and the value relevant to the interrupt. For example, a timer interrupt can be interrupt(5, 1773576000), where 5 is the unique *ID* for timer interrupts and 1773576000 is the unix time stamp of when the timer is triggered. Though in a real system interrupts arise nondeterministically, our semantic model uses *IRQ* to list the interrupts that could trigger in a given program, and likewise for modeling sensor inputs *I*. Each input is denoted in(v), where v is the sensor reading.

A frame F captures the relevant runtime state of an *isr*. In addition to the current command c and execution context E, F includes the *ID* of the ISR that is currently running, a unique process identifier *pID* to distinguish it from any others (e.g., when there are several timer interrupts triggered at different times), a set of shared variables *ShAcc* of the ISR, a mapping υ of local variables to values, and a priority stack $\overrightarrow{pr}$ to handle runtime priorities (we will explain more

$\boxed{IDecl \vdash C^{\infty} \rightarrow C^{\infty\prime}}$

$$\frac{\begin{array}{c} F = (pID, ID, \overrightarrow{pr}_c, ShAcc_c, \upsilon_c, E_c, c) \\ irq = \mathsf{interrupt}(ID, v) \qquad \mathsf{isr}(ID, pr, ShAcc, \lambda x.c_i) \in IDecl \qquad pr > top(\overrightarrow{pr}_c) \\ fresh(pID) \qquad F_n = (ID, pr, ShAcc, x \mapsto v, \cdot, c_i; ret) \qquad o = \mathsf{trigger}(pID) \end{array}}{IDecl \vdash I, (irq :: IRQ), \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{o} I, IRQ, \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q} \textsc{TriggerInt-C}$$

$$\frac{\begin{array}{c} F = (pID_c, ID_c, \overrightarrow{pr}_c, ShAcc_c, \upsilon_c, E_c, c_c) \qquad irq = \mathsf{interrupt}(ID, v) \\ \mathsf{isr}(ID, pr, ShAcc, \lambda x.c_i) \in IDecl \qquad pr \leq top(\overrightarrow{pr}_c) \qquad o = \mathsf{delay}(pID) \end{array}}{IDecl \vdash I, (irq :: IRQ), \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{o} I, IRQ, \mathsf{N}, \mathsf{V}, S, F, (Q :: irq)} \textsc{DelayInt-C}$$

$$\frac{\begin{array}{c} F_c = (pID_c, ID_c, \overrightarrow{pr}_c, ShAcc_c, \upsilon_c, \cdot, \mathsf{ret}) \qquad S = (ID, \overrightarrow{pr}, ShAcc, \upsilon, E, c) \rhd S' \\ \mathsf{isr}(ID, pr, ShAcc, \lambda x.c_i) \in IDecl \qquad pr > top(\overrightarrow{pr}) \qquad irq = \mathsf{interrupt}(ID, v) \qquad fresh(pID) \\ F_n = (pID, ID, pr, ShAcc, x \mapsto v, \cdot, c_i; ret) \qquad o_1 = \mathsf{complete}(pID_c) \qquad o_2 = \mathsf{trigger}(pID) \end{array}}{IDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F_c, (irq :: Q) \xrightarrow{o_1, o_2} I, IRQ, \mathsf{N}, \mathsf{V}, S, F_n, Q} \textsc{Ret-QInt-C}$$

$$\frac{F_c = (pID_c, ID_c, \overrightarrow{pr}_c, ShAcc_c, \upsilon_c, \cdot, \mathsf{ret}) \qquad o = \mathsf{complete}(pID_c)}{IDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, (F \rhd S), F_c, Q \xrightarrow{o} I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q} \textsc{Ret-S-C}$$

$$\frac{\begin{array}{c} F = (pID, ID, \overrightarrow{pr}_c, ShAcc, \upsilon, E, \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}}) \\ F_n = (pID, ID, pr \rhd \overrightarrow{pr}_c, ShAcc, \upsilon, \mathsf{end}_{\mathsf{cr}} \rhd E, c) \end{array}}{IDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F_n, Q} \textsc{StartCrit-C}$$

$$\frac{F = (pID, ID, pr \rhd \overrightarrow{pr}_c, ShAcc, \upsilon, \mathsf{end}_{\mathsf{cr}} \rhd E, \mathsf{skip}) \qquad F_n = (pID, ID, \overrightarrow{pr}_c, ShAcc, \upsilon, E, \mathsf{skip})}{IDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F_n, Q} \textsc{EndCrit-C}$$

Figure 7.5: Selected concurrent operational semantic rules

when introducing the semantic rules). The execution context E remembers the boundaries of if statements and critical sections, and enforces execution ordering.

Lastly, for simplicity, the nonvolatile and volatile memories will directly map program variables to values tagged with an identifier of the process that computed it (a detail needed for proof purposes).

Operational semantics. Continuously-powered semantic rules are of the form: $IDecl \vdash C^{\infty} \xrightarrow{\vec{o}} C^{\infty\prime}$, where $\vec{o}$ is the list of observable events emitted by the step (to be used in defining and proving correctness). Key rules for processing interrupts and critical sections are shown in Fig. 7.5.

The rule TriggerInt-C applies when the priority of an incoming interrupt (pr) is strictly greater than the currently running process (F). The *isr* corresponding to this interrupt preempts the current process. The third premise looks up the corresponding *isr* via the *ID*. A fresh *pID* is generated and a new frame F_n is constructed for the *isr*, where x is bound to input v. In the conclusion, the preempted frame F is pushed on top of the stack and the new frame F_n starts to execute. Finally, the observable event documents that a new process *pID* is triggered via

o = trigger(pID). The rule DelayInt-C applies when the priority of the incoming interrupt is *not* strictly greater than the currently running process. In that case, the interrupt is enqueued to Q.

The next two rules handle completion of the current frame. A new frame is selected either from the delayed interrupt queue Q or the stack S, depending on the priorities. If there is an interrupt in Q with a strictly higher priority than the top frame of S, then the rule Ret-QInt-C applies, triggering the execution of the delayed interrupt. Otherwise, Ret-S-C applies, resuming the top frame on S. If both Q and S are empty, execution halts, which we omit.

Following the ceiling priority protocol [113], when a critical section is encountered, the process's priority is raised sufficiently high to prevent it from being preempted. The rule StartCrit-C pushes the critical section's priority pr to the frame's priority stack $\overrightarrow{pr}_c$ and $\mathsf{end}_{\mathsf{cr}}$ to the execution context E to mark the end of the section. The pr is checked to be sufficiently high by typing rules. For example, once the critical section in entry begins executing, the critical section priority 2 is pushed on top of the task priority 1, so that other processes with priority no higher than 2 (e.g., ISR_IN) must wait until this critical section completes its execution. However, another *isr* with priority 3 could still preempt, and the protocol ensures the safety of it. The rule EndCrit-C applies when the end of a critical section is reached (i.e., a skip command in the frame and $\mathsf{end}_{\mathsf{cr}}$ on top of the execution context). The current priority is popped off the priority stack, thus allowing lower priority processes to begin executing. For example, any task whose priority is strictly greater than the entry task priority of 1, i.e., ISR_IN with a priority of 2, can preempt at this point.

7.3.2 Syntax for Intermittent, Concurrent Programs

To support *intermittence*, our core calculus for concurrency requires constructs to specify desired re-execution behavior in the event of a failure. This includes task-level re-execution policies to specify whether to resume or abort a task's execution, atomic regions within task bodies, and checkpoints for restoring execution from failure. The necessary extensions to syntax are highlighted in Fig. 7.6.

Atomic regions. Commands are extended to include atomic regions, where *aPol* includes sets of variables that are checkpointed ω (e.g., *prev*), read-only ρ (e.g., x), must be first written μ (e.g., d), and writable in volatile memory β.

ISR policies. Each ISR is extended with a re-execution policy *rPol*, which is a program $\lambda x.d$ that is evaluated based on the current system state r (e.g., number of reboots) to a concrete policy name *pname*: discard (do not re-execute if encounters a power failure) or immediate (re-execute using the latest checkpoint), as motivated in Section 7.2. For example, a programmer could specify a task to re-execute up to three times, and then discard after the third attempt. ISRs are also extended with a programmer-specified variable policy *vPol* consisting of shared variables to be accessed *ShAcc*, task-level checkpointed variables *ShCP* (e.g., x in ISR_IN), and non-idempotent variables *nIds* (e.g., *log* in ISR_IN).

Checkpoints and runtime configurations. To support progress and correct (re-)execution, continuously-powered runtime configurations are augmented with a checkpoint context K_{undo}, that is extended from previous chapters with a checkpointed stack S_k. Note that to ensure memory consistency and adherence to re-execution policies, K_{undo} must contain only *stable data* and

Program syntax

Re-ex pol	*rPol*	::=	discard | continue($r, \lambda x.d$)
Pol name	*pname*	::=	discard | immediate
Atom pol	*aPol*	::=	$(\omega, \mu, \beta, \rho)$
Cmd	c	::=	$\ldots \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}$
Var pol	*vPol*	::=	$(ShAcc, ShCP, nIds)$
Int. Srv. Rt.	*isr*	::=	$\mathsf{isr}(ID, pr,$ *vPol*, *rPol*, $\lambda x.c)$

Runtime syntax

Mode	Md	::=	atom(aID, $aPol, NVw, Vw, rb$) | jit | discard(NVw, Vw)
Stack	S	::=	$\cdot \mid F_J(pID) \triangleright S \mid F \triangleright S$
Inst. pid.	*ipID*	::=	$pID \mid (pID, \mathsf{aID}, rb)$
JIT pl fr	$F_J(pID)$	::=	(pID, jit)
Ckpt ctx	K	::=	$(\mathsf{N}_k, \mathsf{V}_k, S_k)$
Frame	F	::=	$(pID, ID, \overrightarrow{pr},$ *vPol*, $\upsilon, E,$ Md, $c) \mid (I, IRQ, \mathsf{K}_{undo}, \mathsf{N}, \mathsf{V}, S, \mathsf{reboot}, Q)$
Int Conf	C^i	::=	$(I, IRQ,$ K, $\mathsf{N}, \mathsf{V}, S, F, Q)$
Nvol mem	N	::=	$\cdot \mid \mathsf{N}, x \mapsto (v,$ *ipID* $)$
Vol mem	V	::=	$\cdot \mid \mathsf{V}, x \mapsto (v,$ *ipID* $)$

Figure 7.6: Summary of syntax for programs and intermittent runtime system. Extensions from continuously-powered syntax highlighted.

resumable processes at all times. Intermittent runtime configurations C^i are extended to include a state right after power failure, where the command to be executed is reboot.

Other runtime constructs are augmented with additional bookkeeping; some are key to the runtime system, while others are used only for proof purposes. Frames are extended with an execution mode Md, which is discard if the task re-execution policy is discard, or is otherwise jit and atom for JIT and atomic regions, respectively. The sets NVw and Vw track which variables have been written to by the current frame, which the runtime system uses to update checkpoints (more details to come in the next section). For proof purposes, we introduce an instance process identifier *ipID*, distinguishing task executions via their *pID* and re-execution instances of the same atomic region via an atomic region identifier aID and reboot count rb.

7.3.3 Intermittent Operational Semantic Rules

Selected operational semantic rules for intermittent execution are shown in Fig. 7.8. We focus on explaining how the runtime operates on the checkpoint context to prepare for and recover from power failure, and handle re-execution policies, assuming an undo-logging system. We explain these rules via an example program execution shown in Fig. 7.7. Expression evaluation relies a big-step operational semantics. Though they are not shown in the main text, the rules are similar to the ones introduced in Chapter 2 but only rely on N, V, and the local state υ_c. The full oeprational semantics and detailed example execution can be found in Appendices E.4 and E.5, respectively. Each line of Fig. 7.7 highlights key components of a runtime configuration and frame. The rule names used in steps from one configuration to the next are shown in the leftmost

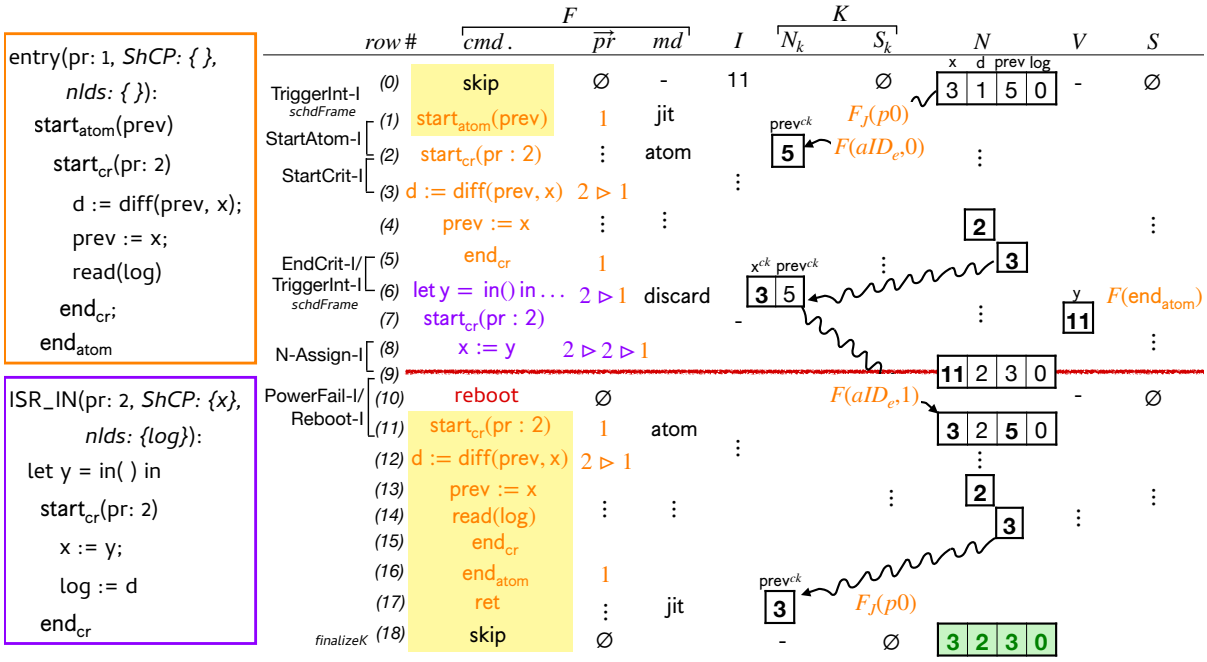

Figure 7.7: Fully annotated program and sketch of its correct intermittent execution trace.

column.

Start (rows (0)-(1)). The program begins its execution by triggering the *isr* entry. The configuration in row 0 is the initial configuration, and row 1 follows from applying the rule TRIGGERINT-I. The main differences between this rule and TRIGGERINT-C from Fig. 7.5 are that the new frame F_n needs an execution mode $\mathtt{Md}_n$ and the checkpoint context K_{undo} needs to be updated, based on the entry's re-execution policy. The *schdFrame* function, whose rules are shown on the bottom of Fig. 7.8, takes as input the current checkpoint context K_{undo}, nonvolatile memory N, and a few other components of the new frame, and returns the current mode $\mathtt{Md}_n$, and an updated checkpoint context K'_{undo}.

If the re-execution policy of the new frame is immediate (e.g., entry), the rule SCHDFRAME-IMMED applies. The policy immediate indicates that the task's code will resume its execution upon reboot from power failure, and since the top-level checkpoint policy of an immediate task is JIT, the returned mode is jit. To help JIT regions resume upon failure, a *JIT placeholder frame* $F_J(pID)$ (e.g., $F_J(p0)$) corresponding to the new process *pID* (e.g., $p0$) is placed on top of the checkpointed stack. This placeholder will be replaced by its corresponding live frame right before power failure or when the execution enters an atomic region, which we explain later. Once the new frame is set up, execution continues with running the entry task code.

Starting an atomic region (rows (1)-(2)). Next, entry enters an atomic region, which requires setting up the checkpoints, as shown in the rule STARTATOM-I. The checkpointed nonvolatile locations N_k are initialized with their current values in nonvolatile memory, denoted as the projection $\mathsf{N}{\downarrow}_{aPol.\omega}$ (e.g., $prev^{ck} \mapsto 5$). We denote updating checkpointed nonvolatile memory with contents from their corresponding locations in nonvolatile memory as $\mathsf{N}_k \leftsquigarrow \mathsf{N}$. The rule also replaces the frame F_k on top of the checkpointed stack (e.g., $F_J(p0)$), which is always the top-level JIT placeholder frame, with the atomic region frame F' (e.g., abbrev. F_{aID_e}), allowing the system to

Selected operational semantic rules.

$$\textsc{TriggerInt-I}\quad \frac{F = (pID_c, ID_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \qquad irq = \mathsf{interrupt}(ID, v) \qquad \mathsf{isr}(ID, pr, vPol, rPol, \lambda x.c_i) \in IDecl \quad pr > top(\overrightarrow{pr}_c) \qquad fresh(pID) \qquad (\mathtt{Md}_n, \mathsf{K}'_{undo}) = schdFrame(\mathsf{K}_{undo}, \mathsf{N}, pID, ID, rPol, v) \quad F_n = (pID, ID, pr, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, c_i; ret) \qquad o = \mathsf{trigger}(pID, \mathtt{Md}_n)}{IDecl \vdash I, (irq :: IRQ), \mathsf{K}_{undo}, \mathsf{N}, \mathsf{V}, S, F, Q \overset{o}{\Longrightarrow} I, IRQ, \mathsf{K}'_{undo}, \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q}$$

$$\textsc{N-Assign-I}\quad \frac{F = (pID_c, ID_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \qquad \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, S_k) \quad x \in \mathsf{dom}(\mathsf{N}) \qquad \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow v \qquad F' = (pID_c, ID_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}], \mathsf{skip}) \quad \mathsf{K}'_{undo} = (\mathsf{N}_k[x \mapsto v], \mathsf{V}_k, S_k) \text{ if } \mathtt{Md}_c = \mathsf{jit} \text{ else } K}{IDecl \vdash I, IRQ, \mathsf{K}_{undo}, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, \mathsf{K}'_{undo}, \mathsf{N}[x \mapsto v], \mathsf{V}, S, F', Q}$$

$$\textsc{StartAtom-I}\quad \frac{F = (pID, ID, \overrightarrow{pr}, vPol, \upsilon, E, \mathsf{jit}, \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}) \qquad \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, F_k \rhd S_k) \quad F' = (pID, ID, \overrightarrow{pr}, vPol, \upsilon, E, \mathsf{atom}(\mathsf{aID}, aPol, \cdot, \cdot, 0), c; \mathsf{end}_{\mathsf{atom}}) \qquad \mathsf{K}'_{undo} = (\mathsf{N}_k \leftsquigarrow \mathsf{N}{\downarrow_{aPol.\omega}}, \mathsf{V}_k, F' \rhd S_k)}{IDecl \vdash I, IRQ, \mathsf{K}_{undo}, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, \mathsf{K}'_{undo}, \mathsf{N}, \mathsf{V}, S, F', Q}$$

$$\textsc{EndAtom-I}\quad \frac{F = (pID, ID, \overrightarrow{pr}, vPol, \upsilon, E, \mathsf{atom}(\mathsf{aID}, aPol, NVw, Vw, rb), \mathsf{end}_{\mathsf{atom}}) \qquad \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, F_k \rhd S_k) \quad F_k = (pID, ID, \overrightarrow{pr}, vPol, \upsilon, E, \mathsf{atom}(\mathsf{aID}, aPol, NVw_0, Vw_0, rb), c; \mathsf{end}_{\mathsf{atom}}) \qquad F' = (pID, ID, \overrightarrow{pr}, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip}) \quad \mathsf{K}'_{undo} = (\mathsf{N}'_k{\downarrow_{ckVarOf(S_k)}}, \mathsf{V}_k \leftsquigarrow \mathsf{V}{\downarrow_{Vw}}, F_J(pID) \rhd S_k) \qquad \mathsf{N}'_k = \mathsf{N}_k \leftsquigarrow \mathsf{N}{\downarrow_{NVw}} \qquad o = \mathsf{endAtom}(pID, \mathsf{aID}, rb)}{IDecl \vdash I, IRQ, \mathsf{K}_{undo}, \mathsf{N}, \mathsf{V}, S, F, Q \overset{o}{\Longrightarrow} I, IRQ, \mathsf{K}'_{undo}, \mathsf{N}, \mathsf{V}, S, F', Q}$$

$$\textsc{PowerFail-I}\quad \frac{F = (pID, ID, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, c) \qquad \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad \mathsf{K}'_{undo} = (\mathsf{N}_k, \mathsf{V}_k, updJitS(S_k, F \rhd S))}{IDecl \vdash I, IRQ, \mathsf{K}_{undo}, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, \mathsf{K}'_{undo}, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q}$$

$$\textsc{Reboot-I}\quad \frac{\mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, F \rhd S_k) \qquad (F' \rhd S'_k) = incRb(F \rhd S_k) \qquad \mathsf{K}'_{undo} = (\mathsf{N}_k, \mathsf{V}_k, F' \rhd S'_k) \qquad o = \mathsf{rb}}{IDecl \vdash I, IRQ, \mathsf{K}_{undo}, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, (Q_{\mathsf{discard}}, Q_{\mathsf{immed.}}) \overset{o}{\Longrightarrow} I, IRQ, \mathsf{K}_{undo}, \mathsf{N}_k \rightsquigarrow \mathsf{N}, \mathsf{V}_k, S'_k, F', Q_{\mathsf{immed.}}}$$

Selected definitions for setting up a new task frame.

$$\textsc{schdFrame-immed}\quad \frac{appPol(rPol) = \mathsf{immediate} \qquad \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad \mathsf{K}'_{undo} = (\mathsf{N}_k, \mathsf{V}_k, F_J(pID) \rhd S_k)}{schdFrame(\mathsf{K}_{undo}, \mathsf{N}, pID, ID, rPol) = (\mathsf{jit}, \mathsf{K}'_{undo})}$$

$$\textsc{schdFrame-discard}\quad \frac{\mathsf{isr}(ID, pr, vPol, rPol, \lambda x.c) \in IDecl \qquad appPol(rPol) = \mathsf{discard} \qquad \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad \mathsf{K}'_{undo} = (\mathsf{N}_k \leftsquigarrow \mathsf{N}{\downarrow_{vPol.ShCP}}, \mathsf{V}_k, S_k)}{schdFrame(\mathsf{K}_{undo}, \mathsf{N}, pID, ID, rPol) = (\mathsf{discard}(\cdot, \cdot), \mathsf{K}'_{undo})}$$

Figure 7.8: Selected intermittent, concurrent operational semantic rules and definitions.

re-execute from the beginning of the region c; $\mathsf{end}_{\mathsf{atom}}$ in the event of a power failure.

Preempted execution (rows (5)-(6)). Skipping ahead, the interrupt with higher priority arrives and the rule TriggerInt-I applies to run ISR_IN (i.e., ISR_IN preempting entry). The preempted

task frame F (abbrev. $F(\mathsf{end}_{\mathsf{atom}})$) is pushed to the stack $F \triangleright S$, to be resumed later. Based on the interrupt's re-execution policy (e.g. discard), the *schdFrame* function again computes a mode $\mathtt{Md}_n$ (e.g., discard) and prepares the checkpoint context accordingly. Here the rule SCHDFRAME-DISCARD applies and since discard tasks do not resume on reboot from failure, the checkpointed stack S_k remains the same. The *schdFrame* function also checkpoints variables at the beginning of a task execution (e.g., x in *ShCP*), updating the checkpointed variables in N_k (e.g., x^{ck}) with their current values in N (e.g., 3). The resulting mode has *NVw* and *Vw* sets initialized to empty, as the task has not yet written to any locations.

Writing to nonvolatile memory (rows (8)-(9)). When a variable in nonvolatile memory is written (e.g., x := y in ISR_IN), the rule N-ASSIGN-I, first evaluating the expression (e.g., y) to a value (e.g., 11) and then storing it in the assigned variable (e.g., x). The *NVw* set is updated with the assigned variable to reflect that the current process has written to it. The last premise updates the checkpointed nonvolatile memory when needed. Since JIT regions do not re-execute, their writes are considered stable and hence values are written through to K'_{undo} on every write, whereas unstable writes by atomic regions and discard tasks should not impact K_{undo}. The write through in a JIT region is also necessary; otherwise progress will be lost when the JIT region is preempted by another (discard) task and then a power failure occurs and the execution restarts from a stale checkpointed memory state (more when explaining the reboot). In the example, the assignment x := y is performed in a discard task, the value of y is not written through to N_k.

Power failure and reboot (rows (9)-(11)). When power is about to fail (e.g., after x := y in ISR_IN), the rule POWERFAIL-I instantaneously saves the execution progress on the checkpointed stack. This models hardware behavior that supports JIT checkpointing. The function $updJitS(S_k, S)$, which takes as arguments the checkpointed stack S_k and current stack S and returns an updated checkpointed stack that will be restored on reboot. To make sure *every* JIT process on the stack will resume after reboot as expected, $updJitS(S_k, S)$ replaces every JIT (placeholder) frame in S_k with its corresponding in-progress frame in S, while leaving other frames in S_k the same.

The rule REBOOT-I models the system recovery from power failures. To distinguish the re-executed atomic region from its failed prior executions (e.g., the new $F(\mathsf{aID}_e, 1)$ versus the failed $F(\mathsf{aID}_e, 0)$), the function $incRb(S)$ increments the reboot count of all atomic regions on the stack. Execution resumes with the top frame which provides a context for an interrupted process. For example, since the atomic region frame $F(\mathsf{aID}_e, 1)$ is on top of the checkpointed stack, the atomic region in entry re-executes from the beginning. The recovered program runs on memories and a stack restored from the checkpoint context, N_k, V_k, and S'_k (e.g., x and prev restored to 3 and 5 from x^{ck} and $prev^{ck}$, respectively). Here $\mathsf{N}_k \rightsquigarrow \mathsf{N}$ is the result of updating the nonvolatile memory N with the checkpointed part N_k.

Frames running in discard mode do not have a corresponding frame in S_k, and are therefore not included in the restored stack, effectively causing all in-progress discard processes to be forgotten on reboot (e.g., ISR_IN). Since discard tasks are thrown out on power failure, the delayed interrupt queue Q only recovers those with policy immediate ($Q_{\mathsf{immed.}}$).

Ending an atomic region (rows (16)-(17)). At the end of an atomic region, when command $\mathsf{end}_{\mathsf{atom}}$ is reached, the rule ENDATOM-I applies. This rule commits updates to memory to the checkpoint context by updating the checkpoints in N_k (e.g., prev^{ck}) and V_k with their values in

N (e.g., prev) and V, for only the variables written by this last atomic region execution, i.e., variables in the tracked *NVw* and *Vw* sets. This is correct because the atomic region is now complete and therefor all the writes are stable. This is also necessary for maintaining progress, similar to why writes in JIT regions are reflected in the checkpoint context. Next, to prepare for executing the next JIT region, the checkpointed atomic region frame (e.g., F_{aID_c}) is replaced with a placeholder frame (e.g., $F_J(p0)$), and the mode is reset to jit.

Returning from a task and halting execution (rows (17)-(18)). At the end of a program execution, the last task finishes executing with no remaining tasks on the stack or in the queue. We omit the counterpart of RET-S-C, as the only addition is setting up the checkpoint context. At this point, like atomic regions, the runtime cleans up the checkpoint context, either popping the corresponding JIT placeholder frame from S_k if the concluding process is immediate, or committing the computed results to N_k and V_k if the concluding process is discard. The example execution ends in the final state of $x \hookrightarrow 3, d \hookrightarrow 2, prev \hookrightarrow 3, log \hookrightarrow 0$, the state from running entry alone, as shown in Fig. 7.1 (c).

Events and more. Definitions for events are similar to interrupts, with the only difference being that events are scheduled by a command rather than firing non-deterministically. The full system definition which handles both interrupts and events, as well as inputs, variable initialization, and alternate versions of tasks can be found in Appendix E.

7.4 Type System

Our type checking relies on three main components: (1) identifying unstable writes of each *isr* (Sec. 7.4.1), (2) constructing typing contexts reflecting unstable write effects from all *isr*s (Sec. 7.4.2), and (3) checking the program (Sec. 7.4.3). To simplify presentation, this section focuses on checking global access patterns (e.g. preventing unstable and non-idempotent writes of other tasks from influencing idempotent data), as shown in Sec. 7.2, which is the main contribution of this work. Checking local access patterns (e.g. preventing WARs) is also necessary to ensure correctness, but we elide discussion to Appendix E.6 as these checks follow prior work [119, 121]. The full type system definition can be found in Appendix E.6.

7.4.1 Identifying Unstable and Non-idempotent Writes from an *isr*

The stability of write effects of a given *isr* depends on several factors, namely, the policy of the *isr* and the priority of any task that reads its write effects. As such, our type system is equipped with two key functions, *EffToLeq*(*isr*) and *EffToHigher*(*isr*), that each return the set of shared variables written by *isr* that have unstable or non-idempotent effects on tasks with a lower or equal priority than *isr* and higher priority than *isr*, respectively. To explain how policies and priorities interact with stability, we illustrate two scenarios in Fig. 7.9, where some current task *isr* has preempted a lower priority task isr_l and is preempted in turn by a higher priority task isr_h. In column (a) the current task *isr* has re-execution policy discard, and in column (b) it has re-execution policy immediate. In each scenario, *isr* writes to some variable x that is read by isr_h before *isr* resumes control and by isr_l after *isr* has either been discarded after power failure

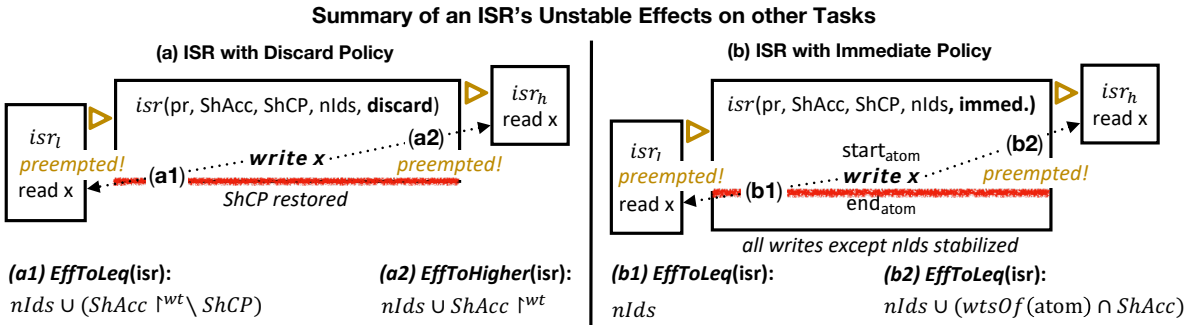

Figure 7.9: Unstable and non-idempotent writes (repr. by dotted lines) from *isr* to other tasks with lower priority vs. higher priority, depending on its re-execution policy: (a) for discard and (b) for immediate.

(scenario (a)) or re-executed successfully (scenario (b)), with each of these data flows marked with a labeled dotted arrow.

We now discuss in what situations such a shared variable has unstable effects and must be returned by *EffToLeq*(*isr*) or *EffToHigher*(*isr*), as indicated on Fig. 7.9, bottom. First, if a variable is declared locally in *nIds* (e.g., *log*) it can hold non-idempotent data and naturally have a non-idempotent effect. If a variable is declared locally as Id, identifying its potentially unstable effects on other tasks is more complex.[2] We begin examining the pair isr_l and *isr* in Fig. 7.9 (a). If *isr* is interrupted by a power failure before completion, any writes from the partial execution of *isr* to nonvolatile memory will persist, except for variables declared in *ShCP* (such as *x* in ISR_IN), which will be restored to checkpointed (stable) values. Thus, arrow (a1) reflects that after a reboot, tasks with priority less than or equal to the priority of *isr* can see unstable values written to any non-checkpointed variables by the partial execution of *isr*. For example, in Fig. 7.7, *log* is declared in *nIds*, so *EffToLeq*(ISR_IN) = {*log*}. Next, we examine the pair *isr* and isr_h. Since isr_h can preempt *isr* at any time, isr_h may instantly read values from any shared variables written by *isr*; however, if the system subsequently loses power before *isr* completes, then *isr* will be discarded, and the values read by isr_h will be unstable. Arrow (a2) reflects that tasks with a higher priority than *isr* risk reading an unstable value from any shared variables written by *isr*, even checkpointed ones.

Now we turn to scenario (b), where the difference is that *isr* has re-execution policy immediate and includes an atomic region. Examining the pair isr_l and *isr*, we note that tasks with a lower or equal priority can only resume their execution after *isr* successfully completes, since by the re-execution policy *isr* is resumed (or re-executed from the start of the atomic region) upon power failure, not discarded. Therefore, all of *isr*'s writes will be stabilized before returning to isr_l; arrow (b1) reflects that no unstable writes from *isr* will be visible to isr_l. On the other hand, for the pair *isr* and isr_h, writes from failed partial executions of the atomic region could be read by isr_h, which is problematic. Consider if isr_h preempts and reads a value written by the atomic region, but the system reboots and the atomic region is restarted. If isr_h preempts *isr* again, it could incorrectly read a value written by some command *earlier* in the atomic region. Thus,

[2]This discussion focuses on unstable writes in *nonvolatile* memory. Writes to volatile memory are much simpler and can be found in Appendix E.6.

arrow (b2) reflects that tasks with a higher priority than *isr* may observe unstable values from any variable written by an atomic region in *isr*. For example, $\mathit{EffToHigher}(\mathsf{entry}) = \{d\}$. Since d is written in the atomic region of entry, d is unstable to ISR_IN.

7.4.2 Building Typing Contexts for each Task

With the sets of unstable and non-idempotent writes in hand, our type system constructs a typing context for each task that accounts for effects from all other tasks. We summarize the types and typing contexts below.

$$
\begin{array}{lllll}
\textit{idempotence qualifier} & \mathit{qID} & ::= & \mathsf{Id} \mid \mathsf{Nid} \\
\textit{type qualifier} & q & ::= & \langle \mathit{qID}_c, \mathit{qID}_l, \mathit{qID}_h \rangle \\
\textit{task typing context} & \Gamma & ::= & \cdot \mid \Gamma, (x : t@q) \\
\textit{program typing context} & \Psi & ::= & \cdot \mid \Psi, (\mathit{ID}, \mathtt{Md}) \mapsto \Gamma
\end{array}
$$

The base types t are int or `bool`. As in the previous chapter, idempotence qualifiers form a join semi-lattice $\mathsf{Id} \sqsubseteq_{id} \mathsf{Nid}$ where join is denoted $\sqcup_{id}$. First, the local qualifier qID_c determines whether or not *isr* is permitted to write a value to x that is potentially derived from non-idempotent data. qID_c is determined by *isr*'s variable policy for x; if x is declared in *nIds* (e.g., *log* in ISR_IN), then qID_c will be Nid, otherwise it is Id. Next, the qualifier qID_l represents whether any strictly lower-priority tasks might write an unstable or non-idempotent value to the given variable. If x is in any of the computed sets $\mathit{EffToHigher}(\mathit{isr}')$ for any isr' whose priority is less than or equal to *isr*, then qID_l is Nid. Otherwise, it is Id. Lastly, the qualifier qID_h represents the effects coming to *isr* from tasks with greater than or equal priority to *isr*. If x is in any of the computed sets $\mathit{EffToLeq}(\mathit{isr}')$ for isr' with greater than or equal priority to *isr* (including *isr* itself), then qID_h is Nid. Otherwise, it is Id.

The program typing context Ψ maps each pair of an *isr*'s *ID* and `Md`, as evaluated from its re-execution policy, to a typing context Γ. The two typing contexts for the same *isr* with different modes differ in their treatment of task-level shared checkpointed variables *ShCP*, which are only used when the policy is discard. Glacier maintains the invariant that checkpointed variables (e.g., *ShCP* and ω), must never contain unstable or non-idempotent effects. Therefore, when constructing a typing context Γ for an *isr* with discard policy, for every $x \in \mathit{ShCP}$, $\Gamma_{\mathit{isr}}(x) = \langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle$; the type checker will report an error if the computed qID_l or qID_h is Nid. The typing contexts for the tasks in Fig. 7.7 are as follows:

$$\Gamma_{\mathsf{en}} = \{x : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle, d : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle, \mathit{prev} : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle, \mathit{log} : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Nid} \rangle\}$$
$$\Gamma_{\mathsf{IN}} = \{x : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle, d : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Nid}, \mathsf{Id} \rangle, \mathit{log} : \mathsf{int}@\langle \mathsf{Nid}, \mathsf{Id}, \mathsf{Id} \rangle\}$$

If d were checkpointed in ISR_IN, then by construction of Γ_{IN} checking would fail because $\mathit{EffToHigher}(\mathsf{entry}) = \{d\}$, causing $\Gamma(d).\mathit{qID}_l$ to be Nid within ISR_IN, however $d \in \mathit{ShCP}$ requires it to be fully idempotent.

7.4.3 Type Checking a Program

We present key typing judgments in Fig. 7.10.

Ψ	$\vdash isr$	: ok	*isr* is typed
$\Gamma, MFWts$	$\vdash^{\mathsf{atom}} e$	$: t@q$	e is typed under atom
Γ	$\vdash^{md} e$	$: t@q$	e is typed under jit or discard
$\Gamma, MFWts$	$\vdash^{\mathsf{atom}}_{ShAcc_t} c$	$\dashv MFWts'$	c is typed under atom
Γ	$\vdash^{md}_{ShAcc_t} c$	: ok	c is typed under jit or discard

Figure 7.10: Typing judgment summary

A program is well-typed if each *isr* is well-typed under the program typing context Ψ, computed based on the algorithm discussed in the previous section. The typing judgments for expressions and commands are indexed by the mode `Md` of the execution and the set of shared variables $ShAcc_t$ accessed by the surrounding task command. These judgments take two forms: one for checking within an atomic region (atom) and the other for any code not within an atomic region (`Md` $\in$ {jit, discard}). This distinction is needed because following prior work [119, 121], typing commands in an atomic region need to check access patterns such as WAR. However, this is not the contribution of this work, so such checks for atomic regions are elided. Expressions e are typed under the same contexts as commands and they are of a simple type int or `bool`, with a type qualifier q reflecting the (non-)idempotence of the expression e. Next we explain the key simplified typing rules, summarized in Fig. 7.11.

Tasks. Each task is type checked individually via the rule T-Task. The rule first evaluates the re-execution policy *rPol* to determine an initial mode `Md`, and uses it to look up the task typing context in Ψ. For tasks with policy discard (e.g., ISR_IN), the initial mode will be discard. Otherwise (e.g., entry), it is jit (e.g., corresponding to Γ_{en}). Then, the rule checks the task command c under that typing context and `Md` (e.g., Γ_{IN} and discard, and Γ_{en} and jit respectively). The input argument to a task is idempotent, i.e., x : int@⟨Id, Id, Id⟩, which is included in Γ.

Atomic regions. The rule T-Atom-Begin-J checks the enclosed command c under a typing context Γ', which is constructed from the task's context and additionally includes an interior *access qualifier* to describe why the access is idempotent: e.g., Id(`CK`) for variables in ω, Id(`WT`) for variables in μ, or Id(`RD`) for variables in ρ. The rule checks that every variable in ω is not locally non-idempotent, and is not an unstable write by other tasks (the last premise), to prevent non-idempotent data from being checkpointed by an atomic region, similar to the check done for *ShCP* when constructing Ψ. Variables in μ should not have unstable writes coming from higher-priority tasks whose preemption and subsequent power failure could also cause the atomic region to read unstable values, but a similar check is not needed for lower priority tasks, because such a variable would be overwritten by an idempotent value before it is read by the atomic region.

Critical sections. To protect against data races, the rule T-Cr-A enforces that critical sections cannot be preempted by tasks that access common shared variables (e.g. the critical section in entry should run without preemption from ISR_IN). The checks performed by this rule are inspired by the ceiling priority protocol of popular frameworks [1], enforcing that the priority pr of a critical section is at least as high as the priority of any task that accesses common shared variables: $pr \geq pr_{\mathsf{c}}$ (e.g. entry's critical section with priority 2 and ISR_IN with priority 2). A task that does not contend for common shared variables is allowed to preempt a critical section if its priority is greater than the critical section, causing the task to trigger. Since their sets of

$$\frac{\begin{array}{l}\text{if } rPol = \mathsf{discard} \text{ then } \mathtt{Md} = \mathsf{discard},\ \text{else } \mathtt{Md} = \mathsf{jit} \\ (\Psi(ID, \mathtt{Md}), x : \mathsf{int}@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle) \vdash^{\mathtt{Md}}_{vPol.ShAcc} c : \mathsf{ok}\end{array}}{\Psi \vdash \mathsf{isr}(ID, pr, vPol, rPol, \lambda x.c) : \mathsf{ok}}\ \text{T-Task}$$

$$\frac{\begin{array}{l}\Gamma' = mkCtxAtom(\Gamma, \omega, \mu, \rho) \\ \Gamma', \cdot \vdash^{\mathsf{atom}}_{ShAcc_t} c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \qquad \omega, \rho, \mu \text{ pairwise disjoint} \\ \forall\, x_w \in \mu.\ \Gamma(x_w) = t_w@\langle \_, \_, \mathsf{Id}\rangle \qquad \forall\, x_k \in \omega.\ \Gamma(x_k) = t_k@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle\end{array}}{\Gamma \vdash^{\mathsf{jit}}_{ShAcc_t} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}}\ \text{T-Atom-Begin-J}$$

$$\frac{\begin{array}{l}pr_{\mathsf{c}} = ceilPr(\{pr' \mid \mathsf{isr}(\ldots, pr', vPol) \in IDecl \wedge (ShAcc \cap vPol.ShAcc \neq \emptyset)\}) \\ pr \geq pr_{\mathsf{c}} \qquad ShAcc \subseteq ShAcc_t \\ \Gamma' = \{x : t@q \mid x \in \mathsf{dom}(\Gamma) \wedge x \notin ShAcc_t \setminus ShAcc\} \qquad \Gamma', MFWts \vdash^{\mathsf{atom}}_{ShAcc_t} c \dashv MFWts'\end{array}}{\Gamma, MFWts \vdash^{\mathsf{atom}}_{ShAcc_t} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \dashv MFWts'}\ \text{T-Cr-A}$$

$$\frac{\begin{array}{l}\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \\ \Gamma \vdash^{\mathsf{atom}}_{ShAcc_t} e : t@\langle qID^e_c, qID^e_l, qID^e_h\rangle \qquad (qID^e_c \sqcup_{id} qID^e_l \sqcup_{id} qID^e_h) \sqsubseteq_{id} qID_c\end{array}}{\Gamma, MFWts \vdash^{\mathsf{atom}}_{ShAcc_t} x := e \dashv MFWts'}\ \text{T-Assign-N-A}$$

Figure 7.11: Selection of simplified typing rules.

shared variables are disjoint, data races with shared variables would be impossible.

Additionally, the rule checks that the critical section's *ShAcc* annotation is correct: its shared accesses are (1) a subset of the task's shared accesses $ShAcc_t$, and (2) that no other shared variables are accessed by it. The former is enforced by checking that $ShAcc \subseteq ShAcc_t$, and the latter by restricting the typing context Γ' to exclude variables appearing in $ShAcc_t$ but not in *ShAcc*.

Assignments. The rule T-Assign-N-A enforces that our policy that no non-idempotent data from local or global sources flows to locally idempotent variables. To gather the effects from local and global sources, the rule joins the expression's qualifiers q_e, and then checks that the result is bound by the local qualifier of the variable qID_c. For the assignment prev := x, $\Gamma_{\mathsf{en}}(x).qID_c \sqcup_{id} \Gamma_{\mathsf{en}}(x).qID_l \sqcup_{id} \Gamma_{\mathsf{en}}(x).qID_h \sqsubseteq_{id} \Gamma_{\mathsf{en}}(prev).qID_c$. If x were not checkpointed by ISR_IN, it would have the type $\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Nid}\rangle$ in Γ_{en}, and the assignment prev := x would have caused a type error because the global effects of x (Nid) are not bounded by the required local usage of *prev* (Id). Likewise, if *log* were not declared in *nIds* by ISR_IN, its type would be $\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle$ and the assignment log := d would flow globally non-idempotent data from d to locally idempotent variable *log*, causing another type error.

7.5 Alternate Versions of Tasks

The core calculus presented is flexible and extensible to additional system behaviors and features. For example, rather than aborting a task completely or insisting on running the same task over and over again until it succeeds, some concurrent intermittent systems will instead opt for running

an *alternate* (perhaps more energy-efficient [82]) version of a task. This section presents the changes to the system needed to support alternate versions of tasks. We highlight key changes to the syntax in Fig. 7.12 in yellow.

...			
Version number	*version*	::=	$n : \texttt{nat}$
Policy name	*pname*	::=	$\mathsf{discard} \mid \mathsf{immediate} \mid \mathsf{alternative}(ID, version)$
Interrupt Service Routine	*isr*	::=	$\mathsf{isr}(ID, version, pr, vPol, rPol, \lambda x.c)$
Mode	Md	::=	$\mathsf{atom}(\mathsf{aID}, aPol, NVw, Vw, rb) \mid \mathsf{jit} \mid \mathsf{discard}(NVw, Vw)$
			$\mid \mathsf{altOf}(pID, rPol) \mid \mathsf{next}(pID, NVw, Vw)$
Frame	*F*	::=	$(pID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, c)$

Figure 7.12: Syntax extensions needed to support alternate versions of tasks.

Syntax. To support alternate versions of a task, we introduce a new policy name alternative which includes an interrupt *ID* and *version* number, that together uniquely identify the alternate task that will run in this task's place in the event of a power failure. All tasks in the program declaration and their frames are extended with a similar *version* number, which is 0 for the initial (or only) version of a task. The mode next corresponds to a task with policy alternative, where *pID* identifies the alternate frame to run in the event of a failure. Like discard tasks and atomic regions, alternatives make use of *NVw* and *Vw* sets to dynamically track which variables are written on the task's execution. Alternative versions of tasks are initially assigned the mode altOf, identifying the initial version (*pID*) and its re-execution policy *rPol*, which will be evaluated when the alternate frame begins to execute. All versions of a given task have the same priority *pr* to maintain the well-formedness of the stack throughout execution.

Scheduling tasks. To prepare for unexpected failure, such an alternate task is scheduled to the checkpoint context before the original task begins executing. This is accomplished by extending the definition of *schdFrame* with an additional case for when the re-execution policy of a task to be scheduled evaluates to $\mathsf{alternative}(ID, version_a)$, and with an additional input v. We show the definition below.

$$\frac{\begin{array}{c} appPol(rPol) = \mathsf{alternative}(ID, version_a) \\ \mathsf{isr}(ID, version_a, pr_a, vPol_a, rPol_a, \lambda x.c_a) \in IDecl \\ \mathsf{K}_{undo} = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad \mathsf{K}_{undo}{}' = (\mathsf{N}_k \leftsquigarrow \mathsf{N}{\downarrow}_{vPol_a.ShCP}, \mathsf{V}_k, F_a \rhd S_k) \\ F_a = (pID_a, ID, version_a, pr_a, vPol_a, x \mapsto v, \cdot, \mathsf{altOf}(pID, rPol_a), c_a; ret) \\ fresh(pID_a) \end{array}}{schdFrame(\mathsf{K}_{undo}, \mathsf{N}, pID, ID, rPol, v) = (\mathsf{next}(pID_a, \cdot, \cdot), \mathsf{K}'_{undo})} \text{ schdFrame-Alt}$$

The rule schdFrame-Alt combines ideas from schdFrame-immed and schdFrame-discard to (1) schedule an alternate version and (2) make sure the current version will be forgotten should power fail during its execution. Like scheduling an immediate task, this definition must too schedule a new task, which is specifically an alternate version ($version_a$) of the same type of task being triggered, i.e, has the same *ID*. To accomplish this, the rule looks up the alternate task in *IDecl*, generates a fresh pID_a for it, and inserts a frame corresponding to the alternate task into

the checkpointed stack. The policy binds the local variable x to the same input data v that the original task would receive, so that the alternate task would run on the same input.

Like scheduling a discard task, the task with policy alternative will be treated like a discard and be thrown out on power failure. The new frame is assigned a mode next($pID_a, \cdot, \cdot$), where pID_a identifies the alternate task frame and the *NVw* and *Vw* sets are initialized to empty, like a discard.

Finalizing a task execution. If an alternative task, i.e., with frame mode next(pID_a, NVw, Vw), finishes executing successfully, the function *finalizeK* applies to finalize the memory state.

The definition of *finalizeK* for alternative tasks also combines ideas from finalizing the memory state after running a discard task or an atomic region. Since the task successfully completed, its alternative is no longer needed. To clean up the state, the rule pops the no longer needed alternative task frame F_a from the checkpointed stack, in a similar fashion to cleaning up the checkpoint stack at the end of an atomic region execution. Since this task behaves similar to a discard task, this one also commits the nonvolatile data written by the task to their checkpoints and scopes down the checkpoints to only the ones needed for the remaining processes.

Typing. Since the success and failure behaviors of alternative tasks match those of discards, all the same typing rules apply. Each version of a task will have its own typing context. To distinguish these different versions, Ψ is extended to take a *version* number as input.

7.6 Correctness

In this section, we define and prove the correctness of our system. Following prior work [40, 44, 121], every intermittent execution of a well-typed program has an equivalent continuously-powered execution. However, the concurrently executing tasks and flexible re-execution policies make finding such a continuously-powered execution more challenging than in the sequential setting.

7.6.1 Defining Correctness

We provide high-level intuition behind key constructs used for defining and proving correctness. The detailed formalism can be found in Appendices E.7- E.10.

Effective steps. Intermittent executions include steps that continuously-powered executions do not have: partial executions of tasks that are eventually discarded, e.g., the first failed execution of the atomic region in entry. Such steps, which we refer to as *ineffective*, should have no effect on idempotent state. In contrast, steps taken by completed processes, which we refer to as *effective*, are matched by a continuously-powered execution (e.g., the highlighted steps in the Fig. 7.7 execution). To distinguish effective from ineffective steps, we first define *effective processes* for a given intermittent execution trace. We write T_I to denote an intermittent execution trace of the form $C^i_0 \overset{o_0}{\Longrightarrow} C^i_1 \cdots \overset{o_n}{\Longrightarrow} C^i_{n+1}$, as induced by the operational semantic rules. The function *ePidOf*(T_I) returns the identifiers of effective processes which either have completed successfully or execute in jit mode.

$$ePidOf(T_I) = \{pID \mid \mathsf{complete}(pID) \in T_I\} \cup \{pID \mid \mathsf{trigger}(pID, \mathsf{jit}) \in T_I\} \cup \{(pID, \mathsf{aID}, rb) \mid \mathsf{endAtom}(pID, \mathsf{aID}, rb) \in T_I\}$$

Formally, effective steps are those taken by frames with an effective instance process identifier.

Equivalent results. A concurrent intermittent execution is correct if it computes the same results as a continuously-powered execution. Towards proving this, we augment the return statement of a task and end of atomic regions with a result argument: $\mathsf{ret}(res)$ and $\mathsf{end}_{\mathsf{atom}}(res)$, where the type of res is $t@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle$. Observations in a trace are augmented with the value of the result: $\mathsf{complete}(ipID, v)$ $\mathsf{endAtom}(ipID, v)$. We write $T{\downarrow_{\mathsf{res}}}$ as the sequence of results of T. Since res can be computed using any expression with appropriate type, comparing results between an intermittent and a continuously-powered trace is a proxy for equivalence between their memory states.

Equivalent configurations. To show that an intermittent execution computes the same results as a continuously-powered execution, we define an equivalence relation between the configurations $C^i \approx C^\infty$ in terms of their memories, execution stack, queue, inputs, and interrupts. Intuitively, the continuously-powered state need only match stable state in the intermittent state.

The relation between nonvolatile memories involves N_C, the memory from continuously-powered execution, N_I, the memory from intermittent execution, and N_k, the checkpointed memory. For a given nonvolatile location x, its value in N_I may differ from the one stored in N_C, if only x were most recently written to by an ineffective process. Further, $\mathsf{N}_C(x)$ may differ from $\mathsf{N}_k(x)$ if its value was also written on the intermittent execution ($\mathsf{N}_I(x) = \mathsf{N}_C(x)$), meaning that it has yet to be committed to the checkpoint context. This relation is general enough so that after reboot when the checkpointed memory N_k is written to N_I and when atomic regions and discard tasks complete and N_k is updated, the relation still holds.

Conceptually, the execution stack of C^∞ matches the stack recovered from S_k, which contains no frames for in-progress discard tasks or partially-executed atomic regions, and only contexts for resumable or re-executable processes, as motivated in Section 7.2. The task Q is similar, where only immediate tasks are recovered on reboot.

Correctness theorem. To formally state the theorem, we first introduce a few auxiliary notations. We write $\mathcal{R}_I(C_0^i)$ and $\mathcal{R}_C(C_0^\infty)$ to denote the set of traces from configurations C_0^i and C_0^∞, respectively, and assume that the program begins execution with the frame F_{en}. We write $K = (\cdot, \mathsf{V}, F_{en})$ to denote the initial checkpoint context with empty nonvolatile memory, a predefined initial volatile memory, and the frame of the entry task. Corollary 2 states our main result, that for any intermittent execution trace of a well-typed program, there exists a continuously-powered execution trace that computes the same result.

Corollary 1 (Intermittent results are correct) $\vdash IDecl : \mathsf{ok}$, $C_0^i = (I, IRQ, K, \mathsf{N}, \mathsf{V}, \cdot, F_{en}, \cdot)$, $C_0^\infty = (I, IRQ, \mathsf{N}, \mathsf{V}, \cdot, F_{en}, \cdot)$, *imply* $\forall T_I \in \mathcal{R}_I(C_0^i)$, $\exists T_C \in \mathcal{R}_C(C_0^\infty)$, *s.t.* $T_I{\downarrow_{\mathsf{res}}} = T_C{\downarrow_{\mathsf{res}}}$.

7.6.2 Proving Correctness

Corollary 2 follows from a simulation proof, sketched in Fig. 7.13, where an intermittent execution trace T_I is shown on top, its corresponding continuously-powered execution trace T_C on the bottom, and the purple lines connect related configurations. Effective and ineffective steps are

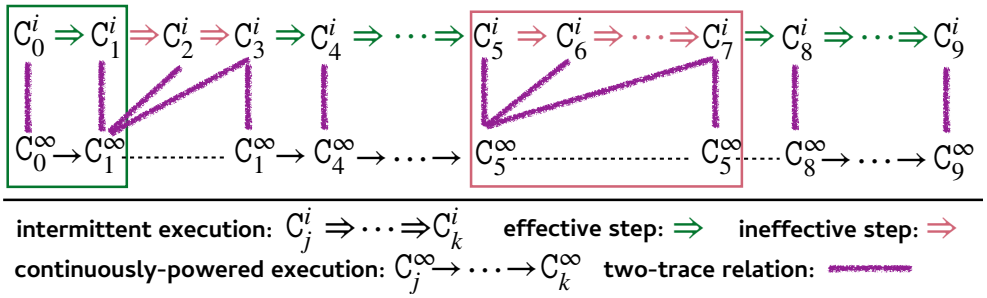

Figure 7.13: Correctness proof roadmap

highlighted by the green and red arrows respectively. The proof is by induction on the length of T_I and relies on two key lemmas: one for effective steps and one for ineffective steps. If a configuration C_j^i is related to a continuously-powered execution state C^∞ and $IDecl \vdash C_j^i \Longrightarrow C_{j+1}^i$ is an effective step, we can show that C^∞ can take the same step to $C^{\infty\prime}$ such that C_{j+1}^i is related to $C^{\infty\prime}$. If the step is ineffective, then we can show that C_{j+1}^i is related to C^∞. Proving the latter is straightforward since the equivalence relation is set up to ignore differences caused by ineffective processes. For the former, we explain a few interesting cases at a high level, where memory or the execution stack is updated.

Consider an assignment $x := e$ to nonvolatile memory by an effective process where x's local qualifier is Id. The configuration relation ensures that C^∞ is also ready to execute the same assignment, and we must show that e evaluates to the same value by C_j^i and C^∞. Since x is locally Id and the program is well-typed, the assignment typing rule enforces that e contains only variables of the type $\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle$. The memory relation tells us that N_I in C_j^i and N_C in C^∞ can only differ in variables that store unstable values (last written by ineffective processes). Since unstable writes have at least one qualifier as Nid, it follows that e contains no effects from unstable writes and therefore it evaluates to the same value by C_j^i and C^∞.

Next, consider the case where an atomic region finishes executing, at which point its results are committed to N_k. We need to show that these values are stable, i.e., they are written by only effective processes. The corresponding operational semantic rule EndAtom-I only updates locations in N_k that have been written in N_I by the atomic region. We have proved that if x is in NVw, it can only be written (1) by this atomic region, or (2) by a higher-priority task. In the first case, since the atomic region finishes executing, the values written to N_k are all the results of an effective process. In the second case, the higher-priority task cannot be ineffective, for otherwise the type of x would be $\langle \mathsf{Id}, \_, \mathsf{Nid} \rangle$, meaning it must not be a checkpointed variable based on our typing rules.

The correctness proof (in Appendix E.10) handles other subtle cases such as branching on a non-idempotent expression, where re-execution may cause different branches to execute.

Chapter 8

Discussion and Related Work

This thesis builds upon prior work that establishes the correctness of intermittent and fault tolerant systems. To position our contributions, we provide a broad survey of related work on intermittent computing and related systems, and explain their contributions relative to this thesis.

8.1 Correctness of Sequential, Intermittent Systems

The first intermittent systems [82, 106, 125] relied on informal notions of correctness, leaving their programs susceptible to two classes of memory consistency bugs: reading computations from partial executions [77] and allowing stale sensor readings from prior executions to persist in nonvolatile memory [118]. Surbatovich et al. [119] provide the first formal framework for reasoning about intermittent execution. They give the correctness definition that we use and identify precise memory invariants needed to ensure correct sequential, intermittent execution. By defining a separate semantics for a variety of logging styles and proving that each system simulates undo-logging, they are able to establish the correctness of many different flavors of intermittent systems. This thesis aims to define a language whose type system is focused on reasoning about the key operations of intermittent computing. By taking this approach, the proposed language would also provide a framework for reasoning about the correctness of different logging styles and policies in intermittent systems. Unlike [119], we are able to establish these guarantees *statically* by taking a type-based approach.

8.1.1 Type Systems for Safe Intermittent Computing

To our knowledge, Curricle [121] is the only other type system for supporting correct, sequential intermittent computing. To reason about inputs and non-idempotence, Curricle formulates correctness for intermittent computing as a non-interference property, asserting that non-idempotent data does not flow to idempotent variables. They use an information flow type system to enforce that well-typed programs are correct relative to non-interference. In their system, variables are also typed with a taint qualifier and idempotence qualifiers are annotated with an interior access qualifier representing basic access modes, e.g., read-only, writable, and a sum type for conditionals. The qualifiers in Curricle are more coarse-grained than the ones introduced in Chapter 6,

with their typing rules designed to eagerly reject correct programs. It is the design of our fine-grained type qualifiers that allows us to safely delay the failure to type check until the end of an atomic region which, in turn, accepts more correct programs. While crash types uses a similar correctness criteria established by prior work [119, 121], we are the first to use a logical relation based on the key operations of intermittent computing to reason about correctness. Furthermore, with crash types, we are also the first to reason about crashes in expression evaluation, which were assumed to execute atomically in prior work [119, 121], as well as effectful outputs.

8.1.2 Enforcing Timeliness and Correct Peripheral Interactions

Other works that investigate the formal properties of intermittent computing either reason about the effects of intermittent execution on peripheral interactions [20] or enforce timeliness constraints on sensor readings [120], which are orthogonal to ours. To reason about computations depending on external peripheral devices involves logging peripheral interactions in a nonvolatile log to be replayed on reboot [20]. Proving correctness then comes down to additionally supporting this logging mechanism in conjunction with the one used by the intermittent system. To enforce freshness and temporal consistency in an intermittent program, prior work [120] proposes using a single atomic region to surround where an input is taken and its subsequent usages, which on reboot, would retrieve and re-execute on new inputs. They provide a tool that automatically infers placement of these atomic regions, and instruments programs accordingly. Enforcing the same constraints in our systems would involve developing a similar static inference.

8.2 Related Programming Languages Theory

The crash types calculus design finds its roots in several areas of programming languages theory, such as adjoint logic, logical relations, and algebraic effect handlers. In this section, we provide more discussion of these areas.

8.2.1 Adjoint Modal Logic

Benton et al. [17, 18] provided the first categorical foundation for using adjoint functors to combine linear and nonlinear logics and showed that a well-behaved calculus requires an independence principle: linear formulae cannot appear in the assumptions of a nonlinear sequent. Follow-up works further generalized the system [74, 75, 107]. There, the relation to Pfenning and Davies's [95] formulation of the lax $\bigcirc$ modality was noted; $\bigcirc$ corresponds to UF, where F and U are adjunctions between truth and validity categories. Short of a full Curry-Howard correspondence for our crash type system and underlying logic, we designed the rules for $\uparrow^s_u$ and $\downarrow^s_u$ based on the above calculi. Our stable and unstable contexts correspond to the validity and truth contexts, respectively. Thus, we speculate that the combination $\uparrow^s_u\downarrow^s_u$ in our system corresponds to the lax modality.

Several prior works used type systems with adjoint modalities to model switching between program modes [15, 38, 101], e.g., switching a process's mode between shared and unshared [15], or adding multicasting, replicable services, and cancellation modes to a session-typed message

passing system [101]. We are the first to use these modalities to handle unforeseen shut-downs and distinguish between stable and power-failure prone modes.

8.2.2 Information Flow Types

Abadi et al. proposed the dependency core calculus framework for tracking program dependencies [3], laying a foundation for reasoning about information flow types. Information flow types have typically been used for security purposes; for example, by tagging program variables as either high security or low security, information flow types enforce that high security data does not flow to low security variables.

Prior work grounds information flow control in modal [11, 84] and epistemic [57] logic, most recently developing a framework for information flow types based upon modal logic [53]. Other work grounds information flow types in graded modal types, which are based on quantitative type theory [12], for tracking how variables are used according to a semiring of quantities. Prior work has observed that information flow types fit this framework, formulating the lattice of security levels (or grades) as a partially ordered semiring [4, 33, 50, 90]. In particular, graded types have been used to formulate information flow types in terms of erasure according to a two-point information flow lattice $\mathsf{L} \leq \mathsf{H}$ [5], as well as quantitative information flow [54]. Though orthogonal to the modalities used to treat our stable and unstable types, formulating our types Id and Nid (where $\mathsf{Id} \sqsubseteq_{id} \mathsf{Nid}$) in terms of graded types [5] seems a natural fit.

8.2.3 Logical Relations

Step-indexing was first introduced by Appel and McAllester [10] who characterize type inhabitation relative to a predicate indexed by the remaining number of steps allotted for a computation. Since then, other works [9, 122] use step-indexing to ensure the well-foundedness of logical relations that handle heaps with cyclic references, dynamic memory allocation, or recursive types. Our crash types model the infinite computation that an atomic region can experience under a non-deterministic number of power failures and re-executions. This recursion necessitates an indexed relation that limits the number of execution attempts a program can make. In our setting, the notion of an allotted number of steps from traditional step-indexing corresponds to an allotted number of intermittent execution attempts.

Jung and Tiuryn introduced a logical relation for lambda definability that allows varying arities [66]. The idea is to increase the arity when passing to later worlds instead of starting with a large arity. Our logical relation can also be viewed as a relation with different arities; the initial type of the relation is binary, while after a crash the type of the value relation only corresponds to the intermittent configuration. During these value steps, the relation is unary, with the continuous configuration acting as a Kripke world for the intermittent configuration. After restoration, the relation reverts to binary.

Logical relations have been widely used to prove program equivalence, e.g., [8, 9, 22, 45]. At a high level, idempotency is similar to program equivalence, but it handles re-execution and requires us only to prove simulation from an intermittent to continuous run, not vice-versa.

8.2.4 Algebraic Effect Handlers

Algebraic effect handlers [85, 96, 97, 98] give a unified theory for computational effects, e.g., exceptions and interactive input/output. A handler accesses the continuation to transform the computation. Following effect handler syntax, we write effectful environmental interactions of our system as $\varepsilon\#\mathsf{in}(b > 0, \uparrow_u^s \kappa)$ or $\varepsilon\#\mathsf{in}(b > 0, \downarrow_u^s \uparrow_u^s \kappa)$ depending on mode and logging style, where b refers to a natural number returned by the environment and κ is the continuation that will run after state is restored. Our `Restore` policy resembles a handler, in that it has access to the continuation, but an atomic region with redo-logging may dismiss the continuation, restarting from a saved command.

8.3 Correctness of Concurrent, Intermittent Systems

To our knowledge, Glacier is the first provably correct concurrent, intermittent system, while providing a more flexible programming model than is allowed by prior work. This section provides more detailed discussion of such works here.

8.3.1 Concurrent, Intermittent System Implementations

Our concurrent, intermittent system design draws inspiration from state-of-the-art implementations of concurrent, intermittent systems [27, 109, 129], while providing a more general abstraction to give additional flexibility in task policies and allowed memory accesses. Conversely, none of these works present precise, formal definitions of intermittent concurrency or proofs of correctness which is the core motivation and contribution of this work.

Our system semantics are closest to the task-based concurrency model of Coati [109]. Coati supports a "split-phase" [2] model of concurrency, where interrupts are split into a short top half that fires immediately, e.g., to copy peripheral registers to nonvolatile memory, and a longer bottom half with the actual processing logic that executes later. Using our system's terminology, ISRs that fire in the top half are discard tasks that enqueue an event containing the actual handler logic that triggers upon completion. This pair of tasks shares an exclusive set of variables, forming a unidirectional private channel. Unlike our model, Coati assumes (but does not check) very restricted access patterns between these two tasks. The discard task must not read or write to main memory and can only write to the shared variables, while the event can access main memory freely, but can only read from the shared variables. Intuitively, since the event can only execute if the discard task finished, the values in the shared variables are stable when read. Finally, Coati's notion of tasks is rigid, not allowing interior atomic regions or critical sections, to reduce the complexity of nesting. The non-standard programming model and strong restrictions on shared memory accesses imposed by Coati are not present in our system.

Karma [27] specifically seeks to support asynchronous peripheral accesses, not general concurrent tasks. They provide a low-level system layer between the application and peripheral API to manage checkpointing and rollbacks, focusing primarily on the problem of recovering the implicit state machine of a peripheral's accesses—orthogonal to the challenge of shared memory concurrency. We do not model detailed peripheral actions. Incorporating such detailed periph-

eral models similar to prior work [20] present an interesting avenue for future work. Combining these models is not straightforward as Karma's method of maintaining consistency requires the system to take a checkpoint before calling a peripheral API, which can break existing atomic region abstractions. Moreover, their checkpoint function for peripheral tasks relies on interrupts and peripheral functions not writing to shared non-volatile variables. While this design choice is reasonable for their setting, it does not scale to general, asynchronous programming tasks since peripheral library APIs often have strict, limited interfaces.

Another concurrent system, Immortal Threads [129] reasons about preemptive multithreading, but does not handle volatile memory state. Their system does not support atomic regions, instead taking a lightweight checkpoint after every instruction that writes to memory, similar to our semantics for JIT regions, though differing under the hood. Their semantics for interrupts are analogous to our discard tasks. For memory consistency, interrupts can only write to memory, committing upon completion, again similar to the restrictions of Coati, which are stricter than ours. Our system supports a superset of the tasks and accesses allowed by Immortal Threads; the interaction between discard tasks and atomic regions, which re-execute, is a novel challenge addressed by our system.

Most recently, IntOS [128] introduces a deeply-embedded operating system layer for intermittent computing. It operates over both nonvolatile and volatile memory, yet their execution model only handles transactions that re-execute from the beginning in the event of a power failure. They do not handle discard tasks and have a number of programming model restrictions, e.g., all accesses to shared memory must be within transactions, locks cannot be used within a transaction. While these restrictions provide a basis for their correctness argument, they make no formal guarantees, and their restrictions limit support for the fine-grained critical sections and flexible re-execution behaviors that we support. Conversely, as an OS, IntOS also contains features that we do not model, such as system calls, heap management, and kernel-level bypass mechanisms. Extending support to these features is beyond the scope of this work.

8.3.2 Concurrent, Embedded Systems

Research on concurrency control for embedded systems has a long history [41, 46, 73], surveyed here [117]. We are not attempting to provide a new method of concurrency control; rather we are trying to make it possible for intermittent systems to use the same concurrency models and frameworks as continuously-powered embedded systems. We do not assume specific implementations for priority levels or critical sections, and base our synchronization on the priority ceiling protocol [113], frequently used in real-time systems. We particularly base our continuously-powered semantics on RTIC [1], a popular real-time interrupt-driven concurrency framework for Rust, itself based on Real Time for the Masses [47, 76].

8.4 Correctness of Related Systems

Correctness of intermittent systems bears high-level resemblance to correctness of other crash-consistent systems, e.g., persistent memory systems [16, 43, 52, 55, 60, 61, 69, 102], file systems [26, 29, 29, 48, 72, 89, 112, 115], or micro-services on the cloud [42, 68]. These appli-

cations all have some notion of a persistent store and compute tasks that should not corrupt the store, even if power fails, a hardware component crashes, or a service is abruptly descheduled; the specifics of each setting vary widely, though, making a formal model from one setting lack abstractions needed by another.

8.4.1 File Systems

For example, Bornholt et al. [26] develop crash-consistency models, similar to memory consistency models, to characterize the failure and recovery patterns of file systems across crashes. Their framework Ferrite can be used to verify formal specifications of file systems by both exhaustively executing litmus tests against all possible crash behaviors and symbolically executing litmus tests against a specification. Like intermittent computing, their memory model considers both volatile and nonvolatile state, providing rules that define the interaction between these memories in the presence of a crash. They rely on a similar notion of correctness for crash-consistency models, relating a crashy (intermittent) trace of a program to its canonical (continuously-powered) trace. However, our work provides formal proofs that instead automatically verify idempotence *once and for all* programs in our language.

Crash Hoare logic (CHL) [29] ensures the correctness of crash and restore operations in a file system by extending Hoare logic with a crash condition and a recovery procedure. The crash condition describes what happens to the state in the event of a crash, while the recovery procedure runs after the crash and manipulates the state before execution resumes. The system checks that if a program crashes, the storage system will recover to a state consistent with the prescribed specifications. Unlike us, however, they require manual effort to formalize the crash condition and recovery policy for each individual procedure they wish to reason about. Therefore, this strategy lacks the once and for all guarantees provided by the type systems built in this thesis, and the flexibility in policies offered by more general abstractions. While CHL reasons only about file system crashes, this thesis sets out to provide abstractions for reasoning about general program crashes.

8.4.2 Persistent and Software Transactional Memory

Other works study fault tolerance with respect to different memory models. Notably, persistent memory models rely solely on nonvolatile memory to perform a computation [34, 79, 86, 126]. Work in this area builds persistent memory interfaces that constrain write orders for executing concurrent [93, 94] or parallel [25] programs. Our work reasons about sequential and concurrent program execution on processors with both volatile and nonvolatile memories.

Weak persistency semantics formalize the behavior of programs running under persistency models for specific processors [102, 103, 104]. They prove their systems are correct according to persistent linearizability, which characterizes the correctness of concurrent program execution according to write orders [43, 62]. Our formalism is at a higher level than theirs, not considering specific processors. To our knowledge, our work is the first to characterize the logical interaction between volatile and nonvolatile memories.

Our proofs of correct concurrent, intermittent execution draw inspiration from prior work using durable linearizability [61] to prove [43] the correctness of concurrent programs executing on

persistent memory systems, decoupling reasoning about correct persistence from linearizability (for us, serializability). Furthermore, Memento [32] uses types to reason about thread-safety and crash-safety simultaneously for concurrent data structures in persistent memory systems. Their language provides a checkpoint primitive, though the meaning is different, saving the results of a particular operation, not the entire system state, as in intermittent computing. Since the data structures Memento targets are explicitly lock-free and accessed by threads (not tasks), they do not reason about transactions and critical sections, as we must. We are interested in providing similar guarantees, but for intermittent systems which operate according to different semantics and concurrency models.

Another related area is software transactional memory (STM) which treats memory accesses like database transactions: committing if an operation succeeds, and retrying if it fails [21, 28, 105]. Unlike STM, Glacier by design does not dynamically detect and recover from write conflicts. When one task interrupts another, neither task is aborted or rolled back. Glacier statically reasons about which variables contain potentially-unstable data. Our re-execution policies are meant to recover from power failure rather than reconcile conflicting transactions, making this area orthogonal to Glacier.

8.4.3 Serverless Applications

Other work uses idempotence to reason about the correctness of concurrent operations in serverless applications [42]. Serverless applications are less fine-grained than intermittent systems. They operate over regions that either entirely fail or succeed, and do not allow memory to persist across failures. The partial updates to memory caused by failed intermittent executions are what make reasoning about concurrent, intermittent program execution so challenging.

Chapter 9

Future Work

The contributions of this thesis build a frontier for designing programming languages, type systems and runtime systems for intermittent computing, a specific class of provably correct fault tolerant systems. In the current technological climate, building dependable, robust, secure, and efficient systems – ones that you can bet your life on and that minimize computational waste – is more crucial than ever before, making reasoning about failures across many applications a necessity. This progression of this thesis, from a basic sequential, intermittent system to gradually examine more sophisticated program features and complex behaviors, forms a solid foundation for adaptations of this work that extend reasoning about intermittency, and beyond to other applications. Studying more complex system behaviors and analogous correctness guarantees in other domains will bring the contributions of this thesis closer to being adopted by real-world applications.

For example, though Glacier provides provably correct supports for concurrent intermittent computing that surpass existing art in terms of expressivity, such as, flexible re-execution policies and variable accesses, it is still quite restrictive in other aspects, such as its conservativity of type checking, that should be addressed in order to support more realisitic programs. As another example, our work on crash types exposes the underlying logical behavior of power failure and recovery while taking intermittent computing as an example. By investigating other applications with similar constraints, we could apply observations from crash types to suit the needs of other application domains.

This chapter outlines several important avenues for future work, including *more flexible models of concurrency* and *distributed systems* in the setting of intermittent computing, as well as examining other applications that could benefit from similar static correctness guarantees.

9.1 Flexible and Adaptable Intermittent Concurrency

This thesis provides the first formal reasoning and provably correct supports for concurrent intermittent systems. Though to manage the inherent complexity of combining concurrency with intermittence, we adopted several simplifying assumptions which we plan to relax in future work.

9.1.1 Relaxing Volatile Variables Typing Constraint

Though not discussed in the main body of this thesis, we assume that volatile variables always have the local qualifier ld which places unintended restrictions on the programming model. For example, unstable or non-idempotent nonvolatile data could never be written to volatile memory, and volatile memory could never be written in a conditional that branches on non-idempotent or unstable data. To remove this restriction and allow variables in volatile memory to be non-idempotent, future work would need to weaken the key invariant that only idempotent values are stored in checkpoints.

9.1.2 Posting Events in Ineffective Processes

The current formulation of Glacier allows events such as a programmer-triggered event via the command post(ID, e), where events can be posted anywhere in the code *except for atomic regions and discardable tasks*. The reason is that posting events is an effectful computation that could modify memory shared by other processes. In particular, because events are scheduled by the software, posting events in atomic regions could cause duplicates to be posted on each (re-)execution of the program. Depending on whether intended by the programmer of an application, this behavior could lead to incorrect results. In future work, we plan to investigate designing and implementing flexible event posting policies, similar to the re-execution policies of tasks. We observe that the behavior of posting events in atomic regions bears resemblance to idempotent and non-idempotent outputs from Chapter 6. In this vein, supporting event posting in atomic regions could be solved by similarly classifying events as idempotent or non-idempotent, and buffering idempotent events to the end of an atomic region execution to guarantee they are run only once. This strategy would require a more flexible correctness proof that reasons about reorderings of tasks in an execution trace.

9.1.3 More Flexible Memory Accesses and Reasoning

To ease the proofs and formulation, the current formulation of Glacier's type system overapproximates both true non-idempotent variables as well as tainted variables with the same qualifier Nid, which places unnecessary restrictions on our type checking. For example, conditionals that branch according to variables written by lower priority atomic regions execute under a non-idempotent context and therefore cannot write to idempotent variables within their branches. As future work, we will explore relaxing the type system to distinguish different types of unstable writes to allow such reordering.

9.1.4 Energy Consumption and Fair Scheduling

In our systems, we show that if there is enough energy, the program will eventually finish executing in a finite number of attempted power cycles. However, we cannot in general guarantee that there is always enough energy to do so. Reasoning about actual energy consumption for embedded systems is highly complex and should be addressed by future provably correct intermittent systems.

Reasoning about energy consumption will be especially challenging in multi-tenant settings. For example, a concurrent system may schedule tasks according to fair energy usage as well as available hardware resources. To prevent energy-hungry processes from starving other processes, a programmer could schedule alternative versions of a task that are more energy efficient than the original [110], yet it is up to the programmer to specify the desired re-execution behavior of tasks and to determine suitable alternate versions. As a more robust approach, future work should develop energy consumption models for concurrent, intermittent programs, as well as definitions of fair scheduling. Integrating these definitions with correctness criteria for concurrent, intermittent systems will be challenging especially with regard to scheduling constraints already imposed by process priorities.

9.1.5 Crash Types for Concurrent and Extensible Intermittent Computing

All of the extensions to Glacier mentioned above will require more flexible correctness criteria, proof strategy, and type system design. From our development of more flexible qualifier evolution and checking from Chapter 6, we extracted more fine-grained qualifiers that could be helpful in tracking variable instability in the concurrent setting to accept more correct programs. Therefore, a concurrent interpretation of modal crash types has potential to bring more expressivity to provably correct concurrent, intermittent systems. By approaching reasoning about intermittent concurrency with a logical relation, we expect encoding policies and invariants of the concurrent system to become more principled and the proofs better structured, which could provide a foundation for more flexible reasoning about concurrent, intermittent systems.

Heterogeneous crash types. There are a few challenges in bringing concurrency to crash types. First, we would need to develop a semantic typing and policies for discard and alternative tasks, which abort or run another piece of code with a potentially different re-execution behavior on reboot. The latter requires examining a form of exception handling for logical relations capable of diverting a computation from a failed discard task to the checkpointed stack. For the latter, we would need to develop crash types that are not self-recursive, but rather that are mutually-recursive, allowing the crash type of one policy to depend on the crash type of another. Furthermore, to prove that computation by ineffective processes does not affect effective processes, the system must use principles from information flow similar to Glacier. Bringing similar guarantees to crash types would require building reasoning about information flow into the logical relation, for which prior work [39, 53] on establishing noninterference via logical relations will serve as a starting point.

Investigating this nature of crash type also lays groundwork for reasoning about failures in *heterogeneous* systems where power could fail on one machine and resume (or re-execute) on another with potentially different system features, e.g., undo- versus redo-logging for atomic regions and hardware capabilities to issue low-power signals for JIT regions.

Shared-memory in logical relations. Additionally, reasoning about shared-memory concurrency with a logical relation will be another challenge to address. In particular, when relating two configurations via our crash type logical relation, we rely on establishing that memories are equivalent up to certain locations at each step of the simulation to establish that the memories are equivalent in the end. However, in a setting with shared-memory concurrency, this

approach no longer works because multiple threads could be accessing the same variable at the same time. Extending our logical relation from crash types to reason about shared-memory concurrency requires extending the calculus with critical sections to enforce fine-grained locks on memory accesses, as we did in Glacier. Prior work builds linear logical relations for message-passing concurrency [124] and for concurrent systems with higher-order stores [23]. However, a concurrent logical relation for additionally reasoning about failures has not yet been examined.

Multi-modal crash types. Lastly, future work should explore integrating the modalities of crash types with a modal interpretation of our qualifiers according to adjunctions and graded types [5]. In particular, the mode structure of adjoint logic could be applied to model the qualifier lattices from Chapter 6. While graded types have been used to reason about information flow [5, 54], to our knowledge, they have not yet been used to reason about memory accesses. Such a formulation would provide an extensible framework for reasoning about intermittent computation that is logically grounded in all aspects of its types.

9.2 Bridging the Gap from Theory to Real-World Adoption

To bring the contributions of this thesis closer to being adopted in real-world scenarios, future work should implement the type systems and runtime systems introduced in this dissertation, investigate their accessibility by non-expert language practitioners, and study how compilation affects guarantees proved for high-level programs.

9.2.1 Accessibility and Implementation

Developing prototypes of our type systems for users to interact with would allow us to study the use cases of our systems as well as the effectiveness of checkpointing error messages. In our type systems for sequential intermittent computing, type checking will fail at the particular point of a program where an access is violated, which also points to an ill-checkpointed variable. However, for the concurrent intermittent system in Chapter 7, where task executions are intertwined and types depend heavily on a task's priority and re-execution behavior relative to others, developing useful error messages for ill-checkpointed variables will be even more challenging. For our concurrent intermittent runtime system, it will be important to investigate whether the proposed checkpointing strategies of the runtime system realistically fit the energy constraints of the target hardware and application domains.

9.2.2 Compiler Optimizations

As intermittent systems typically rely on standard compilers such as LLVM, future work should investigate correctness for intermittent systems while accounting for compiler optimizations. For example, in SSA while it is safe to store temporary variables in volatile memory (which is wiped on power failure), future work should examine how to leverage nonvolatile memory for such temporary variables correctly. Future work should also investigate optimizations such as dead code elimination and instruction combining, which could change the structure and access patterns of a program.

9.3 Distributed Systems

Future work should bring similar guarantees about correct intermittent computing to reasoning about distributed systems of EHDs. Reasoning about failures in distributed systems is important for many applications, including networks of air quality monitoring sensors in smart civil infrastructures [6] to networks of agents in an AI application [24]. While reasoning about distributed systems is an important task, additionally reasoning about failures is especially challenging because the failure and recovery of one node in a network could cause it to resume execution on an inconsistent state from the rest of the network. In each of these settings, failures are expensive to recover from in terms of resources and impact on life [91, 114], therefore making static checking of programs prior to deployment a necessity.

Correctness via consensus. In distributed systems where work is split amongst multiple nodes in a network, the failure of one node could cause inconsistent data to propagate to other nodes, causing incorrect program behavior. Therefore, to allow a network to remain operational and compute correct results despite failures, correctness reasoning and system designs for intermittent computing should be extended with *consensus protocols* that will allow nodes in a distributed system to agree on shared state and execution order.

9.4 Beyond Intermittent Computing

The work presented in this thesis builds a logical foundation for reasoning about crashes, which initiates first steps for providing similar guarantees for other systems that also must endure failures. Writing programming languages and type systems that enforce correct behavior for other fault tolerant applications can bring once-and-for-all correctness guarantees to systems that would be otherwise difficult or tedious to verify. This section outlines a few such applications.

9.4.1 Fault-Resilient Cyberphysical Systems

One field that would benefit from correctness reasoning regarding failures is *cyberphysical systems*. Cyberphysical systems encompass a realm of safety-critical applications, ranging from collision avoidance detection for aircraft [116], to safe braking and acceleration algorithms for autonomous vehicles [92] and trains [67], to ground robot navigation [83]. In each of these applications, forward progress is crucial for efficiency and even safety, even in the face of failure. For example, if a component were to fail in an airplane's collision avoidance algorithm mid-flight, the system should handle the failure and not shut off completely, which could pose even more risk to passengers on the plane.

However, many cyberphysical systems applications rely on commercial off-the-shelf hardware rather than fault tolerant specialized hardware that could provide, e.g., system support to issue low power signals for handling JIT program execution. These systems are thus considered *fault-resilient*; implicit fault tolerance on the hardware level can no longer be assumed, making provisioning for failures on the software level prior to deployment especially essential [31].

In future work, safety-critical reasoning for cyberphysical systems should be extended with reasoning for fault resiliency, and runtime designs with proper checkpointing for failure-susceptible

components. For example, while prior work established a type system that brings provable static safety guarantees to cyberphysical systems [30], there does not yet exist a type system that combines safety guarantees with fault-resilient reasoning. Designing such a language would require marrying the refinement types from prior work [30] with the crash type system.

Chapter 10

Conclusion

This dissertation explores the logical foundations of intermittent computing. Throughout this work, we found that the design of general type-based abstractions rooted in logical reasoning forms a solid foundation for reasoning about complex and realistic behaviors in intermittent systems and furthermore gives rise to dependable system designs.

In Chapter 3, we identified the logical underpinning of intermittent computing as adjoint logic. In particular, we observed that correctness for intermittent computing embodies the independence principle from adjoint logic: stable computation must not depend on unstable values. From these observations, we defined crash types for intermittent computing that encodes the key operations of intermittent computing in terms of shifts from adjoint logic. We developed a type system and operational semantics based on crash types, and a logical relation for reasoning about correctness.

The framework and logical relation introduced for the basic crash type system was flexible enough to be extended. In Chapter 4, we showed that crash types could be extended to accommodate a more flexible checkpointing policy, requiring that only write-after-read variables must be checkpointed in a program to avoid memory consistency bugs. To support this more flexible checkpointing policy, we introduced qualifiers for written variables that are not checkpointed, and developed a basic qualifier evolution; together, these inform us of how variables are accessed in previous stages of type checking to rule out memory accesses that could violate memory consistency.

In Chapter 5, we adapted crash types to support undo-logging, a more popular logging style across intermittent systems. We observed that the behaviors of checkpoint and restore in undo-logging differ from redo-logging, not only in the operational semantics, but also in the defintion of crash types. In particular, we found that in undo-logging, unstable values can persist in non-volatile memory throughout power failure and be restored on reboot. Therefore, the combination of $\downarrow\uparrow$, previously interpreted as a computation that returns a stable value, is used to type the state prior to restore, where stable values replace unstable values. We found that the key operations for undo-logging continue to have interpretations in adjoint logic.

Chapter 6 introduced nondeterministic inputs and effectful outputs; though we could have built upon either the redo- or undo-logging interpretation, we chose the latter because it is more common amongst intermittent systems. With inputs and outputs, RIO bugs could arise from the re-execution of atomic regions with ill-checkpointed variables, causing unintended duplication

of outputs, as well as divergent program behavior on re-execution for inputs. We observe that the former variety of RIO bug has not been addressed by any other provably correct intermittent system. We classified two varieties of outputs, idempotent and nonidempotent, and prevented RIO bugs in idempotent outputs by buffering them until the end of an atomic region where they are guaranteed to execute exactly once.

Programs with RIO bugs on inputs, on the other hand, have been studied by prior work [121], but in an overly conservative manner that requires more variables to be checkpointed than necessary for an entire class of correct programs: programs that overwrite their tainted effects on all paths before they can be read. To support these programs, we delayed rejecting these programs until the end of an atomic region in order to observe program behavior after a branch. To accomplish this goal, we identified an overarching set of memory access patterns in input-dependent programs and generalized the qualifier evolution from Chapter 4 to accommodate the demands of these qualifiers. Moreover, the resulting type checking does not require the programmer or a prior type inference to specify variable accesses, which was required by all other provably-correct system designs.

Lastly, Chapter 7 introduced the first system for supporting concurrent, intermittent programs. There, we co-designed a runtime system and type system capable of supporting diverse re-execution behaviors and flexible variable accesses. In a concurrent system with shared memory, correct behavior requires enforcing that unstable values could not affect other task executions. In doing so, we observed that the checkpoint context must always contain only stable data, meaning that the unstable results of partial ineffective task executions should never affect the checkpoints. We provided the first formal definitions of correctness for concurrent, intermittent computing and proofs of correctness.

Overall this thesis provides a sound basis for supporting intermittent computing in a range of complex system behaviors, all progressing towards more sophisticated models of computation that bring the systems investigated by this thesis closer to benefitting real-world application. Looking ahead, future intermittent systems will be capable of supporting more complex program behaviors and execution conditions, ranging from supporting more flexible concurrent behaviors, to other multi-tenant applications such as distributed or heterogenous systems. Though at its core, the contributions of this thesis identify the fundamental logical behaviors underlying system failures and recovery, which we hope will guide future designers of correct fault tolerant systems towards more dependable, adaptable, and accessible designs.

Bibliography

[1] Real-time interrupt-driven concurrency. https://rtic.rs/1/book/en/by-example/app_priorities.html. Visited September 4, 2025. 7.1, 7.2, 7.4.3, 8.3.2

[2] TinyOS. http://www.docs.tinyos.net/. Visited October 23, 2025. 8.3.1

[3] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, page 147–160, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130953. doi: 10.1145/292540.292555. URL https://doi.org/10.1145/292540.292555. 6.1, 8.2.2

[4] Andreas Abel and Jean-Philippe Bernardy. A unified view of modalities in type systems. *Proc. ACM Program. Lang.*, 4(ICFP), August 2020. doi: 10.1145/3408972. URL https://doi.org/10.1145/3408972. 8.2.2

[5] Andreas Abel, Nils Anders Danielsson, and Oskar Eriksson. A graded modal dependent type theory with a universe and erasure, formalized. *Proc. ACM Program. Lang.*, 7(ICFP), August 2023. doi: 10.1145/3607862. URL https://doi.org/10.1145/3607862. 8.2.2, 9.1.5

[6] Joshua Adkins, Bradford Campbell, Branden Ghena, Neal Jackson, Pat Pannuto, and Prabal Dutta. The Signpost network: Demo abstract. In *Proceedings of the 14th ACM Conference on Embedded Networked Sensor Systems CD-ROM*, SenSys '16, page 320–321, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342636. doi: 10.1145/2994551.2996542. URL https://doi.org/10.1145/2994551.2996542. 1, 9.3

[7] Mikhail Afanasov, Naveed Anwar Bhatti, Dennis Campagna, Giacomo Caslini, Fabio Massimo Centonze, Koustabh Dolui, Andrea Maioli, Erica Barone, Muhammad Hamad Alizai, Junaid Haroon Siddiqui, and Luca Mottola. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, SenSys '20, page 368–381, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375900. doi: 10.1145/3384419.3430722. URL https://doi.org/10.1145/3384419.3430722. 1

[8] Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, page 340–353, New York, NY, USA,

2009. Association for Computing Machinery. doi: 10.1145/1480881.1480925. 3.4.1, 8.2.3

[9] Amal Jamil Ahmed. *Semantics of types for mutable state*. PhD thesis, 2004. URL https://www.ccs.neu.edu/home/amal/ahmedsthesis.pdf. 8.2.3

[10] Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.*, 23(5):657–683, September 2001. ISSN 0164-0925. doi: 10.1145/504709.504712. URL https://doi.org/10.1145/504709.504712. 8.2.3

[11] Aslan Askarov, Sebastian Hunt, Andrei Sabelfeld, and David Sands. Termination-insensitive noninterference leaks more than just a bit. In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS '08, page 333–348, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 9783540883128. doi: 10.1007/978-3-540-88313-5_22. URL https://doi.org/10.1007/978-3-540-88313-5_22. 8.2.2

[12] Robert Atkey. Syntax and semantics of quantitative type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 56–65, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355834. doi: 10.1145/3209108.3209189. URL https://doi.org/10.1145/3209108.3209189. 8.2.2

[13] Domenico Balsamo, Alex S Weddell, Geoff V Merrett, Bashir M Al-Hashimi, Davide Brunelli, and Luca Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18, 2015. doi: 10.1109/LES.2014.2371494. 2.1, 2.3.2, 5, 7

[14] Domenico Balsamo, Alex S. Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M. Al-Hashimi, Geoff V. Merrett, and Luca Benini. Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):1968–1980, 2016. doi: 10.1109/TCAD.2016.2547919. 2.1, 2.3.2, 5, 7

[15] Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In Luís Caires, editor, *Programming Languages and Systems*, pages 611–639, Cham, 2019. Springer International Publishing. ISBN 978-3-030-17184-1. 8.2.1

[16] Naama Ben-David, Guy E. Blelloch, Michal Friedman, and Yuanhao Wei. Delay-free concurrency on faulty persistent memory. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, pages 253–264, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6184-2. doi: 10.1145/3323165.3323187. 8.4

[17] Nick Benton and Philip Wadler. Linear logic, monads and the lambda calculus. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 420–431. IEEE, 1996. 8.2.1

[18] P Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*, pages 121–135. Springer, 1994. 1.2.1, 2.4, 8.2.1

[19] Gautier Berthou, Tristan Delizy, Kevin Marquet, Tanguy Risset, and Guillaume Salagnac. Peripheral state persistence for transiently-powered systems. In *2017 Global Internet of Things Summit (GIoTS)*. IEEE, jun 2017. doi: 10.1109/giots.2017.8016243. 2.1

[20] Gautier Berthou, Pierre-Évariste Dagand, Delphine Demange, Rémi Oudin, and Tanguy Risset. Intermittent computing with peripherals, formally verified. In *The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '20, pages 85–96, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370943. doi: 10.1145/3372799.3394365. 1.1, 7, 8.1.2, 8.3.1

[21] Eleni Bila, Simon Doherty, Brijesh Dongol, John Derrick, Gerhard Schellhorn, and Heike Wehrheim. Defining and verifying durable opacity: Correctness for persistent software transactional memory. In *Formal Techniques for Distributed Objects, Components, and Systems: 40th IFIP WG 6.1 International Conference, FORTE 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15–19, 2020, Proceedings*, page 39–58, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-50085-6. doi: 10.1007/978-3-030-50086-3_3. URL https://doi.org/10.1007/978-3-030-50086-3_3. 8.4.2

[22] Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. Realizability semantics of parametric polymorphism, general references, and recursive types. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures*, pages 456–470, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-00596-1. 8.2.3

[23] Lars Birkedal, Filip Sieczkowski, and Jacob Thamsborg. A concurrent logical relation. In *Computer Science Logic (CSL 2012)*, volume 16 of *LIPIcs*, pages 107–121. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012. doi: 10.4230/LIPIcs.CSL.2012.107. URL https://dagstuhl.de. 9.1.5

[24] B. Bittel, M. Shamsa, B. Inkley, A. Gur, D. Lerner, and M. Adams. Data center silent data errors: Implications to artificial intelligence workloads and mitigations. In *2024 IEEE International Reliability Physics Symposium (IRPS)*, pages 1–5, 2024. doi: 10.1109/IRPS48228.2024.10529375. 9.3

[25] Guy E. Blelloch, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. The parallel persistent memory model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 247–258, 2018. doi: 10.1145/3210377.3210381. 8.4.2

[26] James Bornholt, Antoine Kaufmann, Jialin Li, Arvind Krishnamurthy, Emina Torlak, and Xi Wang. Specifying and checking file system crash-consistency models. *SIGARCH Comput. Archit. News*, 44(2):83–98, March 2016. ISSN 0163-5964. doi: 10.1145/2980024.2872406. 8.4, 8.4.1

[27] Adriano Branco, Luca Mottola, Muhammad Hamad Alizai, and Junaid Haroon Siddiqui. Intermittent asynchronous peripheral operations. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, page 55–67, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369503. doi: 10.1145/

3356250.3360033. URL https://doi.org/10.1145/3356250.3360033. 8.3.1

[28] Dhruva R. Chakrabarti, Hans-J. Boehm, and Kumud Bhandari. Atlas: leveraging locks for non-volatile memory consistency. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, page 433–452, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325851. doi: 10.1145/2660193.2660224. URL https://doi.org/10.1145/2660193.2660224. 8.4.2

[29] Haogang Chen, Daniel Ziegler, Tej Chajed, Adam Chlipala, M. Frans Kaashoek, and Nickolai Zeldovich. Using Crash Hoare Logic for certifying the FSCQ file system. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 18–37, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3834-9. doi: 10.1145/2815400.2815402. 8.4, 8.4.1

[30] Jiawei Chen, José Luiz Vargas de Mendonça, Bereket Shimels Ayele, Bereket Ngussie Bekele, Shayan Jalili, Pranjal Sharma, Nicholas Wohlfeil, Yicheng Zhang, and Jean-Baptiste Jeannin. Synchronous programming with refinement types. *Proc. ACM Program. Lang.*, 8(ICFP), August 2024. doi: 10.1145/3674657. URL https://doi.org/10.1145/3674657. 9.4.1

[31] Kuan-Hsun Chen, Jing Li, Federico Reghenzani, and Jian-Jia Chen. Introduction to the special issue on fault-resilient cyber-physical systems—Part I. *ACM Trans. Cyber-Phys. Syst.*, 8(3), July 2024. ISSN 2378-962X. doi: 10.1145/3677021. URL https://doi.org/10.1145/3677021. 9.4.1

[32] Kyeongmin Cho, Seungmin Jeon, Azalea Raad, and Jeehoon Kang. Memento: A framework for detectable recoverability in persistent memory. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 7, New York, NY, USA, jun 2023. Association for Computing Machinery. doi: 10.1145/3591232. 8.4.2

[33] Pritam Choudhury, Harley Eades, and Stephanie Weirich. A dependent dependency calculus. In Ilya Sergey, editor, *Programming Languages and Systems*, pages 403–430, Cham, 2022. Springer International Publishing. ISBN 978-3-030-99336-8. 8.2.2

[34] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, 2011. ISBN 978-1-4503-0266-1. doi: 10.1145/1950365.1950380. 8.4.2

[35] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, page 767–781, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450349116. doi: 10.1145/3173162.3173210. URL https://doi.org/10.1145/3173162.3173210. 1

[36] Alexander Curtiss, Blaine Rothrock, Abu Bakar, Nivedita Arora, Jason Huang, Zachary Englhardt, Aaron-Patrick Empedrado, Chixiang Wang, Saad Ahmed, Yang Zhang, Nabil Alshurafa, and Josiah Hester. Facebit: Smart face masks platform. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(4), December 2022. doi: 10.1145/3494991. 1

[37] Manjeet Dahiya and Sorav Bansal. Automatic verification of intermittent systems. In Isil Dillig and Jens Palsberg, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 161–182, Cham, 2018. Springer International Publishing. ISBN 978-3-319-73721-8. 1.1

[38] Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. Resource-aware session types for digital contracts. In *IEEE 34th Computer Security Foundations Symposium*, CSF '21, pages 1–16, 2021. 8.2.1

[39] Farzaneh Derakhshan, Stephanie Balzer, and Limin Jia. Session logical relations for noninterference. *CoRR*, abs/2104.14094, 2021. URL `https://arxiv.org/abs/2104.14094`. 9.1.5

[40] Farzaneh Derakhshan, Myra Dotzel, Milijana Surbatovich, and Limin Jia. Modal crash types for intermittent computing. In Thomas Wies, editor, *Programming Languages and Systems*, pages 168–196, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-30044-8. 1.2.1, 2.3.2, 7, 7.2, 7.6

[41] Edsger W. Dijkstra. *Cooperating sequential processes*, page 65–138. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 0387954015. 8.3.2

[42] Haoran Ding, Zhaoguo Wang, Zhuohao Shen, Rong Chen, and Haibo Chen. Automated verification of idempotence for stateful serverless applications. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 887–910, Boston, MA, jul 2023. USENIX Association. ISBN 978-1-939133-34-2. URL `https://www.usenix.org/conference/osdi23/presentation/ding`. 8.4, 8.4.3

[43] Emanuele D'Osualdo, Azalea Raad, and Viktor Vafeiadis. The path to durable linearizability. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '23, page 748–774, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3571219. 8.4, 8.4.2

[44] Myra Dotzel, Farzaneh Derakhshan, Milijana Surbatovich, and Limin Jia. Modal crash types for WAR-aware intermittent computing. *ACM Trans. Program. Lang. Syst.*, 47(2), April 2025. ISSN 0164-0925. doi: 10.1145/3716311. URL `https://doi.org/10.1145/3716311`. 1.2.1, 2.3.2, 7, 7.2, 7.6

[45] Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. page 143–156, 2010. doi: 10.1145/1863543.1863566. URL `https://doi.org/10.1145/1863543.1863566`. 3.4.1, 8.2.3

[46] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, 2004. doi: 10.1109/LCN.2004.38. 8.3.2

[47] Johan Eriksson, Fredrik Häggström, Simon Aittamaa, Andrey Kruglyak, and Per Lind-

gren. Real-time for the masses, step 1: Programming API and static priority SRP kernel primitives. In *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 110–113, 2013. doi: 10.1109/SIES.2013.6601482. 8.3.2

[48] Gidon Ernst, Jörg Pfähler, Gerhard Schellhorn, and Wolfgang Reif. Inside a verified flash file system: Transactions and garbage collection. In Arie Gurfinkel and Sanjit A. Seshia, editors, *Verified Software: Theories, Tools, and Experiments*, pages 73–93, Cham, 2016. Springer International Publishing. ISBN 978-3-319-29613-5. doi: 10.1007/978-3-319-29613-5_5. 8.4

[49] Francesco Fraternali, Bharathan Balaji, Yuvraj Agarwal, Luca Benini, and Rajesh Gupta. Pible: battery-free mote for perpetual indoor BLE applications. In *Proceedings of the 5th Conference on Systems for Built Environments*, pages 168–171. ACM, 2018. 1

[50] Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. Combining effects and coeffects via grading. *SIGPLAN Not.*, 51(9):476–489, September 2016. ISSN 0362-1340. doi: 10.1145/3022670.2951939. URL https://doi.org/10.1145/3022670.2951939. 8.2.2

[51] Graham Gobieski, Amolak Nagi, Nathan Serafin, Mehmet Meric Isgenc, Nathan Beckmann, and Brandon Lucia. Manic: A vector-dataflow architecture for ultra-low-power embedded systems. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 670–684, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369381. doi: 10.1145/3352460.3358277. 1

[52] Vaibhav Gogte, William Wang, Stephan Diestelhorst, Peter M. Chen, Satish Narayanasamy, and Thomas F. Wenisch. Relaxed persist ordering using strand persistency. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 652–665. IEEE Press, 2020. ISBN 9781728146614. doi: 10.1109/ISCA45697.2020.00060. URL https://doi.org/10.1109/ISCA45697.2020.00060. 8.4

[53] Hemant Gouni, Frank Pfenning, and Jonathan Aldrich. Structural information flow: A fresh look at types for non-interference. *Proc. ACM Program. Lang.*, 9(OOPSLA2), October 2025. doi: 10.1145/3764116. URL https://doi.org/10.1145/3764116. 8.2.2, 9.1.5

[54] Hemant Gouni, Frank Pfenning, and Jonathan Aldrich. Security reasoning via substructural dependency tracking. *Proc. ACM Program. Lang.*, 10(POPL), January 2026. doi: 10.1145/3776669. URL https://doi.org/10.1145/3776669. 8.2.2, 9.1.5

[55] Gromadzki, Tomasz and Michalski, Jan and Sałyk, Oksana. PMDK: Persistent Memory Development Kit. https://github.com/daos-stack/pmdk, 2019. Visited April 15th, 2026. 8.4

[56] Xiaochen Guo, Engin Ipek, and Tolga Soyata. Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. *SIGARCH Comput. Archit. News*, 38(3):371–382, June 2010. ISSN 0163-5964. doi: 10.1145/1816038.1816012. 1

[57] Joseph Y. Halpern and Kevin R. O'Neill. Secrecy in multiagent systems. *ACM Trans.*

Inf. Syst. Secur., 12(1), October 2008. ISSN 1094-9224. doi: 10.1145/1410234.1410239. URL https://doi.org/10.1145/1410234.1410239. 8.2.2

[58] Josiah Hester, Travis Peters, Tianlong Yun, Ronald Peterson, Joseph Skinner, Bhargav Golla, Kevin Storer, Steven Hearndon, Kevin Freeman, Sarah Lord, Ryan Halter, David Kotz, and Jacob Sorber. Amulet: An energy-efficient, multi-application wearable platform. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*, SenSys '16, pages 216–229, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4263-6. doi: 10.1145/2994551.2994554. URL http://doi.acm.org/10.1145/2994551.2994554. 1

[59] Josiah Hester, Kevin Storer, and Jacob Sorber. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, SenSys '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354592. doi: 10.1145/3131672.3131673. URL https://doi.org/10.1145/3131672.3131673. 2.1

[60] Joseph Izraelevitz, Terence Kelly, and Aasheesh Kolli. Failure-atomic persistent memory updates via justdo logging. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 427–442, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4091-5. doi: 10.1145/2872362.2872410. 8.4

[61] Joseph Izraelevitz, Hammurabi Mendes, and Michael L. Scott. Linearizability of persistent memory objects under a full-system-crash failure model. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing*, pages 313–327, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53426-7. 8.4, 8.4.2

[62] Joseph Izraelevitz, Hammurabi Mendes, and Michael L. Scott. Linearizability of persistent memory objects under a full-system-crash failure model. In Cyril Gavoille and David Ilcinkas, editors, *Distributed Computing*, pages 313–327, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg. ISBN 978-3-662-53426-7. 8.4.2

[63] Neal Jackson, Joshua Adkins, and Prabal Dutta. Capacity over capacitance for reliable energy harvesting sensors. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, IPSN '19, page 193–204, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362849. doi: 10.1145/3302506.3310400. URL https://doi.org/10.1145/3302506.3310400. 1

[64] Junyoung Jang, Sophia Roshal, Frank Pfenning, and Brigitte Pientka. Adjoint Natural Deduction. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, volume 299 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-323-2. doi: 10.4230/LIPIcs.FSCD.2024.15. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FSCD.2024.15. 2.4

[65] Hrishikesh Jayakumar, Arnab Raha, and Vijay Raghunathan. QuickRecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently pow-

ered computers. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, 2014. doi: 10.1145/2700249. 2.1

[66] Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In *International Conference on Typed Lambda Calculi and Applications*, pages 245–257. Springer, 1993. 3.4.1, 8.2.3

[67] Aditi Kabra, Stefan Mitsch, and André Platzer. Verified train controllers for the federal railroad administration train kinematics model: Balancing competing brake and track forces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4409–4420, 2022. doi: 10.1109/TCAD.2022.3197690. 9.4.1

[68] Konstantinos Kallas, Haoran Zhang, Rajeev Alur, Sebastian Angel, and Vincent Liu. Executing microservice applications on serverless, correctly. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 7 of *POPL '23*, page 367–395, New York, NY, USA, 2023. Association for Computing Machinery. doi: 10.1145/3571206. URL https://doi.org/10.1145/3571206. 8.4

[69] Aasheesh Kolli, Vaibhav Gogte, Ali Saidi, Stephan Diestelhorst, William Wang, Peter M. Chen, Satish Narayanasamy, and Thomas F. Wenisch. Language support for memory persistency. *IEEE Micro*, 39(3):94–102, 2019. doi: 10.1109/MM.2019.2910821. 8.4

[70] Vito Kortbeek, Abu Bakar, Stefany Cruz, Kasim Sinan Yildirim, Przemysław Pawełczak, and Josiah Hester. BFree: Enabling battery-free sensor prototyping with Python. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(4), December 2020. doi: 10.1145/3432191. 2.3.2

[71] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemysław Pawełczak. Time-sensitive intermittent computing meets legacy software. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pages 85–99, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378476. 2.1

[72] Eric Koskinen and Junfeng Yang. Reducing crash recoverability to reachability. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 97–108, New York, NY, USA, 2016. ACM. doi: 10.1145/2837614.2837648. 8.4

[73] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, page 85–95, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581135742. doi: 10.1145/605397.605407. URL https://doi.org/10.1145/605397.605407. 8.3.2

[74] Daniel R Licata and Michael Shulman. Adjoint logic with a 2-category of modes. In *International Symposium on Logical Foundations of Computer Science*, pages 219–235. Springer, 2016. 8.2.1

[75] Daniel R Licata, Michael Shulman, and Mitchell Riley. A fibrational framework for sub-

structural and modal logics. In *2nd International Conference on Formal Structures for Computation and Deduction*, FSCD '17. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. 8.2.1

[76] Per Lindgren, Emil Fresk, Marcus Lindner, Andreas Lindner, David Pereira, and Luís Miguel Pinho. Abstract timers and their implementation onto the ARM Cortex-M family of MCUs. *SIGBED Rev.*, 13(1):48–53, March 2016. doi: 10.1145/2907972.2907979. URL `https://doi.org/10.1145/2907972.2907979`. 8.3.2

[77] Brandon Lucia and Benjamin Ransford. A simpler, safer programming and execution model for intermittent systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '15, 2015. ISBN 978-1-4503-3468-6. doi: 10.1145/2737924.2737978. 1.1, 1.2.1, 2.1, 2.3.2, 2.3.2, 4, 4.1, 5, 8.1

[78] Brandon Lucia, Brad Denby, Zachary Manchester, Harsh Desai, Emily Ruppel, and Alexei Colin. Computational nanosatellite constellations: Opportunities and challenges. *GetMobile: Mobile Comp. and Comm.*, 25(1):16–23, June 2021. ISSN 2375-0529. doi: 10.1145/3471440.3471446. 1

[79] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. Architecture exploration for ambient energy harvesting nonvolatile processors. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015. doi: 10.1109/HPCA.2015.7056060. 8.4.2

[80] Kiwan Maeng and Brandon Lucia. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 1101–1116, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367127. doi: 10.1145/3314221.3314613. 2.1, 7.2

[81] Kiwan Maeng and Brandon Lucia. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '20, pages 1005–1021, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450376136. doi: 10.1145/3385412.3385998. 2.3.2

[82] Kiwan Maeng, Alexei Colin, and Brandon Lucia. Alpaca: Intermittent execution without checkpoints. *Proc. ACM Program. Lang.*, 1(OOPSLA):96:1–96:30, October 2017. ISSN 2475-1421. doi: 10.1145/3133920. 1.1, 2.1, 2.3.2, 4, 5, 7.5, 8.1

[83] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. Formal verification of obstacle avoidance and navigation of ground robots. *I. J. Robotics Res.*, 36(12):1312–1340, 2017. doi: 10.1177/0278364917733549. 9.4.1

[84] Kenji Miyamoto and Atsushi Igarashi. A modal foundation for secure information flow. In *Workshop on Foundations of Computer Security*, pages 187–203, 2004. 8.2.2

[85] E. Moggi. *Computational lambda-calculus and monads*. 1989. doi: 10.1109/LICS.1989.39155. 8.2.4

[86] Dushyanth Narayanan and Orion Hodson. Whole-system persistence. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, 2012. ISBN 978-1-4503-0759-8. doi: 10.1145/2150976.2151018. 8.4.2

[87] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. Camaroptera: A batteryless long-range remote visual sensing system. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, ENSsys'19, pages 8–14, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-7010-3. doi: 10.1145/3362053.3363491. 1

[88] NASA. What is KickSat-2? `https://www.nasa.gov/ames/kicksat`, 2019. Visited April 15th, 2026. 1

[89] Gian Ntzik, Pedro da Rocha Pinto, and Philippa Gardner. Fault-tolerant resource reasoning. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems*, pages 169–188, Cham, 2015. Springer International Publishing. ISBN 978-3-319-26529-2. 8.4

[90] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019. doi: 10.1145/3341714. URL `https://doi.org/10.1145/3341714`. 8.2.2

[91] Elan Justice Pavlinich. The dangers of data centers. `https://www.environmentalhealthproject.org/post/the-dangers-of-data-centers`, 2026. June 28, 2026. 9.3

[92] Christian Pek and Matthias Althoff. Fail-safe motion planning for online verification of autonomous vehicles using convex optimization. *IEEE Transactions on Robotics*, PP, 12 2020. doi: 10.1109/TRO.2020.3036624. 9.4.1

[93] Steven Pelley, Peter M. Chen, and Thomas F. Wenisch. Memory persistency. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, Piscataway, NJ, USA, 2014. ISBN 978-1-4799-4394-4. doi: 10.1109/ISCA.2014.6853222. 8.4.2

[94] Steven Pelley, Peter M Chen, and Thomas F Wenisch. Memory persistency: Semantics for byte-addressable nonvolatile memory technologies. *IEEE Micro*, 35(3):125–131, 2015. 8.4.2

[95] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical structures in computer science*, 11(4):511–540, 2001. 2.4, 8.2.1

[96] Gordon Plotkin and John Power. Semantics for algebraic operations. *Electronic Notes in Theoretical Computer Science*, 45:332–345, 2001. 8.2.4

[97] Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In *Proceedings of the 19th European Symposium on Programming*, pages 80–94. Springer, 2009. 8.2.4

[98] Matija Pretnar and Gordon D Plotkin. Handling algebraic effects. *Logical methods in computer science*, 9, 2013. 8.2.4

[99] Proteus Digital Health. Proteus Digital Health. `http://www.proteus.com/`, 2015. 1

[100] Klaas Pruiksma. Adjoint Logic with Applications. 6 2024. doi: 10.1184/R1/25900861. v1. URL https://kilthub.cmu.edu/articles/thesis/Adjoint_Logic_with_Applications/25900861. 2.4

[101] Klaas Pruiksma and Frank Pfenning. A message-passing interpretation of adjoint logic. *Journal of Logical and Algebraic Methods in Programming*, 120:100637, 2021. 2.4, 8.2.1

[102] Azalea Raad and Viktor Vafeiadis. Persistence semantics for weak memory: Integrating epoch persistency with the tso memory model. *Proc. ACM Program. Lang.*, 2(OOPSLA): 137:1–137:27, October 2018. ISSN 2475-1421. doi: 10.1145/3276507. 8.4, 8.4.2

[103] Azalea Raad, John Wickerson, Gil Neiger, and Viktor Vafeiadis. Persistency semantics of the Intel-x86 architecture. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi: 10.1145/3371079. 8.4.2

[104] Azalea Raad, John Wickerson, and Viktor Vafeiadis. Weak persistency semantics from the ground up: Formalising the persistency semantics of ARMv8 and transactional models. *Proc. ACM Program. Lang.*, 3(OOPSLA):135:1–135:27, October 2019. ISSN 2475-1421. doi: 10.1145/3360561. 8.4.2

[105] Azalea Raad, Ori Lahav, John Wickerson, Piotr Balcer, and Brijesh Dongol. Intel PMDK transactions: Specification, validation and concurrency (extended version), 2023. URL https://arxiv.org/abs/2312.13828. 8.4.2

[106] Benjamin Ransford, Jacob Sorber, and Kevin Fu. Mementos: System support for long-running computation on RFID-scale devices. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, 2011. ISBN 978-1-4503-0266-1. doi: 10.1145/1950365.1950386. 1.1, 8.1

[107] Jason Reed. A judgmental deconstruction of modal logic. *Unpublished manuscript, January*, 2009. URL https://www.cs.cmu.edu/~jcreed/papers/jdml.pdf. 1.2.1, 2.4, 8.2.1

[108] Alberto Rodriguez Arreola, Domenico Balsamo, Geoff V Merrett, and Alex S Weddell. Restop: Retaining external peripheral state in intermittently-powered sensor systems. *Sensors*, 18(1):172, 2018. 6.2.1

[109] Emily Ruppel and Brandon Lucia. Transactional concurrency control for intermittent, energy-harvesting computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '19, page 1085–1100, 2019. ISBN 9781450367127. doi: 10.1145/3314221.3314583. 1.1, 2.1, 2.3.2, 7, 8.3.1

[110] Emily Ruppel, Milijana Surbatovich, Harsh Desai, Kiwan Maeng, and Brandon Lucia. An architectural charge management interface for energy-harvesting systems. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 318–335, 2022. doi: 10.1109/MICRO56248.2022.00034. 1, 9.1.4

[111] Alanson P Sample, Daniel J Yeager, Pauline S Powledge, Alexander V Mamishev, and Joshua R Smith. Design of an RFID-based battery-free programmable sensing platform. *IEEE Transactions on Instrumentation and Measurement*, 57(11), 2008. 1

[112] Gerhard Schellhorn, Gidon Ernst, Jörg Pfähler, Dominik Haneberg, and Wolfgang Reif. Development of a verified flash file system. In *Proceedings of the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z - Volume 8477*, ABZ '14, pages 9–24, Berlin, Heidelberg, 2014. Springer-Verlag. ISBN 9783662436516. doi: 10.1007/978-3-662-43652-3_2. 8.4

[113] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990. doi: 10.1109/12.57058. 7.1, 7.3.1, 8.3.2

[114] Sameer H. Shah. Four water insecurity concerns about datacenters driving the AI revolution. *PLOS Water*, 5(1):1–9, 01 2026. doi: 10.1371/journal.pwat.0000500. URL https://doi.org/10.1371/journal.pwat.0000500. 9.3

[115] Helgi Sigurbjarnarson, James Bornholt, Emina Torlak, and Xi Wang. Push-button verification of file systems via crash refinement. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 1–16, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. doi: 10.5555/3026877.3026879. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/sigurbjarnarson. 8.4

[116] Eric Squires, Pietro Pierpaoli, and Magnus Egerstedt. Constructive barrier certificates with applications to fixed-wing aircraft collision avoidance. In *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1656–1661, 2018. doi: 10.1109/CCTA.2018.8511342. 9.4.1

[117] Ryo Sugihara and Rajesh K. Gupta. Programming models for sensor networks: A survey. *ACM Trans. Sen. Netw.*, 4(2), April 2008. ISSN 1550-4859. doi: 10.1145/1340771.1340774. URL https://doi.org/10.1145/1340771.1340774. 8.3.2

[118] Milijana Surbatovich, Limin Jia, and Brandon Lucia. I/O dependent idempotence bugs in intermittent systems. *Proc. ACM Program. Lang.*, 3(OOPSLA):183:1–183:31, October 2019. ISSN 2475-1421. doi: 10.1145/3360609. 6.2.1, 6.2.1, 8.1

[119] Milijana Surbatovich, Brandon Lucia, and Limin Jia. Towards a formal foundation of intermittent computing. *Proc. ACM Program. Lang.*, 4(OOPSLA), November 2020. doi: 10.1145/3428231. 1.1, 2.3.2, 2.3.2, 4, 4.1, 5, 7, 7.2, 7.4, 7.4.3, 8.1, 8.1.1

[120] Milijana Surbatovich, Limin Jia, and Brandon Lucia. Automatically enforcing fresh and consistent inputs in intermittent systems. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, volume 7 of *PLDI '21*, New York, NY, USA, jun 2021. Association for Computing Machinery. doi: 10.1145/3453483.3454081. 1.1, 2.1, 6.2, 8.1.2

[121] Milijana Surbatovich, Naomi Spargo, Limin Jia, and Brandon Lucia. A type system for safe intermittent computing. volume 7, New York, NY, USA, jun 2023. Association for Computing Machinery. doi: 10.1145/3591250. URL https://doi.org/10.1145/3591250. (document), 1.1, 2.2, 2.3.2, 6.1, 6.2.1, 6.2.2, 6.3, 6.2.2, 6.3.1, 7, 7.2, 7.2, 7.4, 7.4.3, 7.6, 8.1.1, 10

[122] Jacob Thamsborg and Lars Birkedal. A Kripke logical relation for effect-based program

transformations. *ACM SIGPLAN Notices*, 46(9):445–456, 2011. 3.4.1, 8.2.3

[123] TI Inc. Overview for MSP430FRxx FRAM. http://ti.com/wolverine, 2014. Visited July 28, 2025. 1

[124] Bernardo Toninho. *A Logical Foundation for Session-Based Concurrent Computation*. PhD thesis, Carnegie Mellon University and Universidade Nova de Lisboa, 2015. URL http://reports-archive.adm.cs.cmu.edu/anon/2015/CMU-CS-15-109.pdf. 9.1.5

[125] Joel Van Der Woude and Matthew Hicks. Intermittent computation without hardware support or programmer intervention. In *Proceedings of OSDI'16: 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016. doi: 10.5555/3026877.3026880. 2.1, 2.3.2, 4.1, 5, 8.1

[126] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, 2011. ISBN 978-1-4503-0266-1. doi: 10.1145/1950365.1950379. 8.4.2

[127] Harrison Williams, Xun Jian, and Matthew Hicks. Forget failure: Exploiting SRAM data remanence for low-overhead intermittent computation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, pages 69–84, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378478. 2.1

[128] Yilun Wu, Byounguk Min, Mohannad Ismail, Wenjie Xiong, Changhee Jung, and Dongyoon Lee. IntOS: Persistent embedded operating system and language support for multithreaded intermittent computing. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 425–443, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/wu-yilun. 8.3.1

[129] Eren Yıldız, Lijun Chen, and Kasim Sinan Yıldırım. Immortal threads: Multithreaded event-driven intermittent computing on Ultra-Low-Power microcontrollers. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 339–355, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yildiz. 1.1, 7, 8.3.1

[130] Zac Manchester. KickSat. http://zacinaction.github.io/kicksat/, 2015. 1

[131] Hong Zhang, Jeremy Gummeson, Benjamin Ransford, and Kevin Fu. Moo: A batteryless computational RFID and sensing platform. *Department of Computer Science, University of Massachusetts Amherst., Tech. Rep*, 2011. 1

Appendix A

Modal Crash Types for Intermittent Computing

A.1 Progress and Preservation

Lemma 1 (Progress for shifted expressions) *If*

$$\texttt{Md} \mid b{:}\texttt{nat} \mid \Omega \vdash_{Rd} e : \uparrow A$$

then $\forall\ n : \texttt{nat}$ *with* $n > 0$ *and* $\forall\, \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$*, either*

- $Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ *or*
- $\exists(\gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ *such that* $\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Proof: The proof proceeds by structural induction over $\texttt{Md} \mid b : \texttt{nat} \mid \Omega \vdash_{Rd} e : \uparrow A$.

Case 1 [T-Loc-Read]**.** Suppose the last rule in the typing derivation is T-Loc-Read:

$$\frac{\Omega = x : \uparrow A @ q, \Omega'}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega \vdash_{\texttt{RD}} x : \uparrow A}\ \text{(T-Loc-Read)}$$

Then by inversion, we have $\Omega = x : \uparrow A @ q, \Omega'$. By assumption, $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$. By inversion of $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$ according to the well-formedness definition, one of the two subcases hold:

Subcase 1. $\mathsf{N} = \ell @ q \hookrightarrow v, \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-loc-Read applies:

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{N} = \ell @ q \hookrightarrow v, \mathsf{N}' \qquad \delta(q, \texttt{RD}) \neq \mathit{UN} \qquad n = n' + 1}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \rightarrow \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v}\ \text{(D-loc-Read)}$$

This yields the desired result.

Subcase 2. $\mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-LOC-READ applies:

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}' \qquad \delta(q, \mathtt{RD}) \neq \mathit{UN} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-VAR-READ)}$$

This yields exactly the desired result.

Case 2 [T-BOOL-T]**.** Suppose that the last rule in the typing derivation is T-BOOL-T:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathbf{tt} :\uparrow \mathit{bool}} \text{ (T-BOOL-T)}$$

By the value rule V-BOOL-T and the assumption $n > 0$ we get

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{tt})} \text{ (V-BOOL-T)}$$

Case 3 [T-BOOL-F]**.** Suppose that the last rule in the typing derivation is T-BOOL-F:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathbf{ff} :\uparrow \mathit{bool}} \text{ (T-BOOL-F)}$$

By the value rule V-BOOL-F and the assumption $n > 0$ we get

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{ff})} \text{ (V-BOOL-F)}$$

Case 4 [T-INT]**.**

Suppose that the last rule in the typing derivation is T-INT:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathbf{n} : \uparrow \mathit{int}} \text{ (T-INT)}$$

By the value rule V-INT and the assumption $n > 0$ we get:

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{n})} \text{ (V-INT)}$$

□

Theorem 1 (Progress for expressions) *If* $\mathtt{Md} \mid b \mathcal{R} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Rd;\mathtt{Sig}} e : \tau$*, then* $\forall\ n : nat$ *with* $n\mathcal{R}m$ *and* $\forall\ \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b \ \mathcal{R}\ m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Rd;\mathtt{Sig}} e : \tau$. We consider a specific (co-)natural number $n\mathcal{R}m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b\ =0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau \qquad \mathtt{Md} \mid b\ >0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau}\ \text{(T-enough?)}$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n (i.e. $n = 0$ or $n > 0$):

Subcase 1 [$n = 0$]**.** By the value rule V-e-Crash, we have

$$Val(\gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid e),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, we have

(1) $\mathtt{Md} \mid b\ =0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau$

(2) $\mathtt{Md} \mid b\ >0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau$

(3) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\}$

(4) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis to (2) to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

which completes the proof of this subcase.

Case 2 [T-Binary**].**

Suppose the last rule in the typing derivation is T-Binary and $e = e_1 \odot e_2$:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \ \geq\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e_1 : \mathsf{C}_{T}^{\mathtt{Md}} \\ \mathtt{Md} \mid b \ \geq\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}} \qquad \odot : \uparrow T \times \uparrow T' \rightarrow \uparrow T''\end{array}}{\mathtt{Md} \mid b \ >\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e_1 \odot e_2 : \mathsf{C}_{T''}^{\mathtt{Md}}}\ \text{(T-Binary)}$$

By assumption, we know that $n > 0$. By inversion, we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_T^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}}$

(3) $\odot : \uparrow T \times \uparrow T' \to \uparrow T''$

By the inductive hypothesis applied to (1), either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$, or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$.

From here, the proof proceeds in two subcases:

Subcase 1. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$

We apply the inductive hypothesis to (2) to get two sub-subcases.

Subcase 1a. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2)$.

Since $n > 0$, we can apply the dynamic rule D-Binary-v to get

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v,$$

where $v = e_1 \odot e_2$, and $n = n' + 1$.

Subcase 1b. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2'$. By D-Binary-2,

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid V \mid e_1 \odot e_2'$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$. Then, by D-Binary-1,

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1' \odot e_2$$

The desired result holds in all subcases.

Case 3 [T-∨-succ]**.**

Suppose the last rule in the typing derivation is T-∨-succ:

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau \vee \downarrow\uparrow A}\ \text{(T-∨-succ)}$$

By assumption, we get $n > 0$. By inversion, we have:

$$\dagger\ \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A$$

We know that the last rule for deriving † is T-R-Shift:

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A}\ \text{(T-R-Shift)}$$

By inversion, we have:

(1) $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A$

(2) $\Sigma = \downarrow\Sigma'$

(3) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

By assumption $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and (2), it follows from Lemma 27 that $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. Then the desired result follows directly by application of Lemma 26 to (1) (since $n > 0$).

□

Theorem 2 (Progress for commands) *If* $\mathtt{Md} \mid b \mathcal{R} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$*, then* $\forall\ n : \mathtt{nat}$ *with* $n\mathcal{R}m$ *and* $\forall\ \gamma, \mathsf{N}, \mathsf{V}$ *with* $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ *or*
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b \mathcal{R} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$. We consider a specific (co-)natural number $n\mathcal{R}m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau \qquad \mathtt{Md} \mid b\ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b\ \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b\ \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau}\ \text{(T-enough?)}$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n:

Subcase 1 [$n = 0$]**.** By the value rule V-c-Crash, we have

$$Val(\gamma \mid \mathsf{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid c),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, and since $n > 0$, we have

(1) $\mathtt{Md} \mid b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau$

(2) $\mathtt{Md} \mid b\ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$

(3) $\mathtt{Sig}' = \{\mathtt{Md} \mid b\ \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\}$

(4) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis on this judgment to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ or
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

which completes the proof of this subcase.

Case 2 [T-IF]**.**

Suppose the last rule in the typing derivation is T-IF and $c = \textbf{if}\ e\ \textbf{then}\ c_1\ \textbf{else}\ c_2$:

$$\frac{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \texttt{C}^{\texttt{Md}}_{\texttt{bool}} \quad \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1 : \tau \qquad \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_2 : \tau}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} \textbf{if}\ e\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau}\ \text{(T-IF)}$$

By assumption, we know that $n > 0$. By inversion, we have:

(1) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \texttt{C}^{\texttt{Md}}_{\texttt{bool}}$

(2) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1 : \tau$

(3) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_2 : \tau$

By Theorem 16 applied to (1), either

- $Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

From here, the proof proceeds in two subcases:

Subcase 1. $Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$.

Since $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \texttt{C}^{\texttt{Md}}_{bool}$, we have by inversion on T-ENOUGH? followed by inversion on T-∨-SUCC that

$$\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \downarrow\uparrow bool$$

Again, by inversion on T-R-SHIFT, we have

(1) $\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \uparrow bool$

(2) $\Omega = \Omega', \Omega''_{\texttt{ck}}$

(3) $\Sigma = \downarrow\Sigma'$

By inversion on the rules T-BOOL-T and T-BOOL-F, we have that either e is **tt** or **ff**.

Subcase 1a. If $e = \textbf{tt}$, then since $n > 0$ the rule D-IF-TT applies and yields the desired result.

$$\frac{n = n' + 1 \qquad Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \texttt{tt})}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\ \texttt{tt}\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \rightarrow\ \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1}\ \text{(D-IF-TT)}$$

Subcase 1b. Similarly, if $e = \textbf{ff}$, then since $n > 0$, the rule D-IF-FF applies and yields the desired result.

$$\frac{n = n' + 1 \qquad Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \texttt{ff})}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\ \texttt{ff}\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \rightarrow\ \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2}\ \text{(D-IF-FF)}$$

Subcase 2. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$. Then the rule D-IF applies and produces the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \; e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 \;\to \\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2\end{array}}\ \text{(D-IF)}$$

Case 3 [(T-LET)]**.**

Suppose the last rule in the typing derivation is T-LET:

$$\frac{\begin{array}{c}\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \;@\; \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau\end{array}}{\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{let}\, x = e_1\, \textbf{in}\, c : \tau}\ \text{(T-LET)}$$

By assumption, we know $n > 0$. By inversion, we have a typing derivation for:

(1) $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \;@\; \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau$

By Theorem 16 applied to (1), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$.

The proof proceeds in two subcases.

Subcase 1. Suppose that $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$. Then since $n > 0$, the rule D-LET-STEP-V applies and yields the desired result. Note that here γ' extends γ by adding a new mapping from x to ℓ.

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \; e_1) \\ \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \; \mathit{fresh} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{let}\, x = e_1\, \textbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell @ \mathtt{Ck} \hookrightarrow e_1 \mid c}\ \text{(D-LET-STEP-V)}$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$. Then since $n > 0$, the rule D-LET-STEP applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \; e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{let}\, x = e\, \textbf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{let}\, x = e'\, \textbf{in}\, c}\ \text{(D-LET-STEP)}$$

Case 4 [T-∨-SUCC].

Suppose the last rule in the typing derivation is T-$\vee$-Succ:

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \downarrow\uparrow\mathtt{unit}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \tau \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-Succ)}$$

Then since $n > 0$, the rule V-skip applies and yields the desired result.

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip})}\ \text{(V-skip)}$$

Case 5 [T-Assign]**.**

Suppose the last rule in the typing derivation is T-Assign:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} p := e : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}}}\ \text{(T-Assign)}$$

By assumption, we know that $n > 0$. By inversion on T-Assign, we have

(i) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(ii) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A$

By Theorem 16 applied to (i), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

The proof proceeds in two subcases.

Subcase 1. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$.

By inversion on T-w-Shift applied to (ii), we have:

$$\frac{\Sigma =\downarrow \Sigma' \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{Wt} p : \uparrow A \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A}\ \text{(T-w-Shift)}$$

(1) $\Sigma =\downarrow \Sigma'$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{Wt} p : \uparrow A$

(3) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

Now we proceed by inversion on T-Loc-Write applied to (2). From this inversion, we have:

$$\frac{\Omega, \Sigma' = x{:}\uparrow A @ q, \Omega'_2 \qquad q \neq \mathtt{RD}}{\kappa \mid \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{Wt} x : \uparrow A}\ \text{(T-Loc-Write)}$$

(i) $\Omega, \Sigma' = x{:}\uparrow A@q, \Omega_2'$, and

(ii) $q \neq \mathtt{RD}$

By assumption, we have $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. By Lemma 27, and $\Sigma =\downarrow \Sigma'$, we have $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. By the well-formedness definition, one of the two subcases hold:

Subcase 1a. $\mathsf{V} = \ell@q \hookrightarrow v', \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, the D-Assign-V rule applies and yields the desired result.

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q \neq \mathtt{RD} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

Subcase 1b. $\mathsf{N} = \ell@q \hookrightarrow v', \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, the D-Assign-N rule applies and yields the desired result.

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q \neq \mathtt{RD} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}} \text{ (D-Assign-N)}$$

Subcase 2. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$. The rule D-Assign-step applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid p := e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid p := e'} \text{ (D-Assign-step)}$$

Case 6 [(T-seq)]**.**

Suppose the last rule in the typing derivation is T-seq:

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau} \text{ (T-seq)}$$

Then the desired result follows by D-seq:

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 ;_{\gamma|\mathsf{V}} c_2} \text{ D-seq}$$

Case 7 [(T-seq-d)]**.** Suppose the last rule in the typing derivation is T-seq-d:

$$\frac{W = \gamma \mid V \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \qquad \Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma) \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau}\ \text{(T-SEQ-D)}$$

By inversion we have

(1) $W = \gamma \mid V$

(2) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}$

(3) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$

(4) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau$

By the inductive hypothesis applied to (1), we have that either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$ or
- $\exists(\gamma \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$.

The proof proceeds in two subcases.

Subcase 1. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$.

By (2), $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$, and the assumption $n > 0$, it follows by inversion on T-ENOUGH? that

(1) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}\}$

(2) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$

(3) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c_1 : \mathsf{C}_{\mathtt{unit}}$

(4) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}$

By definition, $\mathsf{C}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Additionally, observe that $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$ implies that $c_1 = \mathsf{skip}$. Then by inversion of T-$\vee$-SUCC

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-SUCC)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit}$.

The assumption $n > 0$ implies that $n = n' + 1$. By this fact, (1), and $\mathsf{V} = \mathsf{V} \restriction \mathit{dom}(\mathsf{V})$ (which is trivially satisfied because $\mathsf{V} = \mathsf{V}$), the rule D-SEQ-V applies

$$\frac{n = n' + 1 \qquad W = \gamma \mid \mathsf{V} \qquad \mathsf{V} = \mathsf{V} \restriction \mathit{dom}(\mathsf{V})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathbf{skip};_W c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2}\ \text{(D-SEQ-V)}$$

Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathbf{skip};_W c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2$ is the desired result.

Subcase 2. $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \;\rightarrow\; \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$.

Since $n > 0$, the rule D-CONT applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \;\rightarrow\; \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_W c_2 \;\rightarrow\; \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{(D-SEQ-STEP)}$$

□

Lemma 4 (Well-typedness of expressions under crash in jit) $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathrm{RD};\mathtt{Sig}'} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathrm{RD}} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}\}$.

Proof: Note that $\mathsf{C}^{\mathsf{jit}}_{\mathtt{A}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}) \vee \downarrow\uparrow\mathtt{A}$. Let $\Omega' = \Omega, \Omega''_{\mathrm{ck}}$ where $\Sigma = \downarrow\Omega''$. By axiom 3, we have $\varepsilon \;\#\; \mathsf{i}\c{n}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumption that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathrm{RD}} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}} \in \mathtt{Sig}'$.

$$\cfrac{\cfrac{\cfrac{\cfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash_{\mathrm{RD}} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}} \in \mathtt{Sig}' \qquad \Omega' = \Omega, \Omega''_{\mathrm{ck}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathrm{RD};\mathtt{Sig}'} \uparrow e : \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}} \text{(T-JIT-RESTORE)} \qquad \varepsilon \;\#\; \mathsf{i}\c{n}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega, \Omega''_{\mathrm{ck}} \vdash_{\mathrm{RD};\mathtt{Sig}'} \varepsilon \;\#\; \mathsf{i}\c{n}(b > 0, \uparrow e) : (\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{A}})} \text{(T-CHARGE)} \qquad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathrm{RD};\mathtt{Sig}'} \downarrow\varepsilon \;\#\; \mathsf{i}\c{n}(b > 0, \uparrow e) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}})} \text{(T-JIT-STOP)}}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathrm{RD};\mathtt{Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{A}) \vee \downarrow\uparrow\mathtt{A}} \text{(T-}\vee\text{-CRASH)}$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathrm{RD};\mathtt{Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{A}) \vee \downarrow\uparrow\mathtt{A}$ yields the desired result.

□

Lemma 4 (Well-typedness of expressions under crash in aID) *If* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathrm{RD};\mathtt{Sig}} e' : \tau$ *then* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathrm{RD};\mathtt{Sig}} e : \tau$.

Proof: Let the type $\tau = \mathsf{C}^{\mathsf{aID}}_{A}$ and note that $\mathsf{C}^{\mathsf{aID}}_{A} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{A}$. By inversion on the assumed typing judgment

$$\cfrac{\cfrac{\cfrac{\cfrac{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \qquad \Omega = \Omega', \Omega''_{\mathrm{ck}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow e' : \uparrow\mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}} \text{(T-AID-RESTORE)} \qquad \varepsilon \;\#\; \mathsf{i}\c{n}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \;\#\; \mathsf{i}\c{n}(b > 0, \uparrow e') : (\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})} \text{(T-CHARGE)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} \downarrow\varepsilon \;\#\; \mathsf{i}\c{n}(b > 0, \uparrow e') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})} \text{(T-AID-STOP)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} e' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{A}} \text{(T-}\vee\text{-CRASH)}$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig}$

(2) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

(3) $\varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow e : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ \text{(T-aID-Restore)} \quad \varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow e) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-Charge)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow e) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-aID-stop)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{A}}\ \text{(T-}\vee\text{-Crash)}$$

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{A}$ yields the desired result.

□

Lemma 5 (Well-typedness of commands under crash in `jit`) $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}\}$.

Proof: Note that $\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Let $\Omega' = \Omega, \Omega''_{\mathsf{ck}}$ where $\downarrow\Omega'' = \Sigma$. By axiom 3, we have that $\varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumptions $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}' \quad \Omega' = \Omega, \Omega''_{\mathsf{ck}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{Sig}'} \uparrow c : \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}}\ \text{(T-Jit-Restore)} \quad \varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega' \vdash_{\mathtt{Sig}'} \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow c) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ \text{(T-Charge)} \quad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} \downarrow \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow c) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ \text{(T-Jit-stop)}}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-Crash)}$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$ yields the desired result. □

Lemma 6 (Well-typedness of commands under crash in aID) *If* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c' : \tau$ *then* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$.

Proof: Let the type $\tau = \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}$ and note that $\mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \downarrow \Omega'' \vdash c_0\ : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid\ b\ > 0 : \mathtt{nat} \mid\ \Omega \vdash_{\mathtt{Sig}} \uparrow c' : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ \text{(T-aID-Restore)} \quad \varepsilon \,\#\, \mathsf{i\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid\ \Omega \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{i\eta}(b > 0, \uparrow c') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-Charge)}}{\mathsf{aID}(c_0) \mid \cdot \mid\ \Omega; \Sigma' \vdash_{\mathtt{Sig}} \downarrow \varepsilon \,\#\, \mathsf{i\eta}(b > 0, \uparrow c') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-aID-stop)}}{\mathsf{aID}(c_0) \mid b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-Crash)}$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \downarrow \Omega'' \vdash c_0\ : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig}$

(2) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

(3) $\varepsilon \,\#\, \mathsf{i\eta}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \downarrow \Omega'' \vdash c_0\ : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid\ b\ > 0 : \mathtt{nat} \mid\ \Omega \vdash_{\mathtt{Sig}} \uparrow c : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ \text{(T-aID-Restore)} \quad \varepsilon \,\#\, \mathsf{i\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid\ \Omega \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{i\eta}(b > 0, \uparrow c) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-Charge)}}{\mathsf{aID}(c_0) \mid \cdot \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow \varepsilon \,\#\, \mathsf{i\eta}(b > 0, \uparrow c) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-aID-stop)}}{\mathsf{aID}(c_0) \mid b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-Crash)}$$

The conclusion $\mathsf{aID}(c_0) \mid b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$ yields the desired result.

□

Theorem 3 (Preservation for expressions) *If*

$$(\dagger) \quad \mathtt{Md} \mid b\ \geq\ 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau$$

and for some $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ *and (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e\ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$$

then

$$\mathtt{Md} \mid b\ \geq\ 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \tau$$

with $n' \geq 0$*.*

Proof: The proof is by induction on the size of e. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e\ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Case 1. [D-Binary-1].

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1' \odot e_2} \text{ (D-Binary-1)}$$

By the first premise of D-Binary-1, we have that $n > 0$. By inversion on the assumed typing judgment (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau.$$

By inversion on $(\dagger_1)$ via T-Binary

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

By the inductive hypothesis applied to (1) and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' : \tau^s,$$

and $n' \geq 0$. By the following application of the T-Binary rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' : \tau^s \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s} \text{ (T-Binary)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$.

In both subcases, we have the typing judgments $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$ and $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$. Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau} \text{ (T-enough?)}$$

Case 2 [D-Binary-2].

$$\frac{n > 0 \qquad \gamma \mid \mathit{Val}(\mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_2'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_1 \odot e_2'} \text{ (D-Binary-2)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s.$$

By inversion on ($\dagger_1$) via T-Binary

$$\frac{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \qquad \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

By the inductive hypothesis applied to $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$ and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_2'$, it follows that

$$\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2' : \tau^s,$$

and $n' \geq 0$. By the following application of the T-Binary rule we get

$$\frac{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \qquad \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2' : \tau}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s} \text{ (T-Binary)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s \qquad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau}{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau} \text{ (T-enough?)}$$

Case 3. [D-Binary-v].

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \begin{array}{c} n = n' + 1 \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2) \end{array} \qquad v = e_1 \odot e_2}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-Binary-v)}$$

By the first premise $n > 0$ and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s.$$

By inversion on ($\dagger_1$) via T-Binary

$$\frac{\begin{array}{c} \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

From (1) and (2), we apply inversion on T-Enough?, T-∨-Succ, and T-R-Shift to get $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e_1 : \uparrow A^s$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e_2 : \uparrow A^s$ for $\Sigma =\downarrow \Sigma'$. By definition of $\odot$ operator, we have $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} v : \uparrow A^s$ for $v = e_1 \odot e_2$. By application of T-∨-Succ and T-R-Shift we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c} \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau \end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau} \text{ (T-enough?)}$$

Case 4. [D-VAR-READ].

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}' \qquad \delta(q, \mathtt{RD}) \neq \mathit{UN} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-VAR-READ)}$$

By the last premise $n > 0$ and $n' \geq 0$. By inversion on (†) via T-ENOUGH? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s.$$

By inversion on $(\dagger_1)$ via T-∨-SUCC

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau_1 \vee \downarrow\uparrow A^i} \text{ (T-∨-SUCC)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i$.

By inversion on $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i$ via T-R-SHIFT

$$\frac{\Sigma = \downarrow\Sigma' \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^i}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i} \text{ (T-R-SHIFT)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^i$ and $\Sigma = \downarrow\Sigma'$.

By Lemma 27 applied to the assumption $\vdash_\gamma^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and premise $\Sigma = \downarrow\Sigma'$, we know that $\vdash_\gamma^{Md} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. By definition of well-formedness, we know that $\gamma = \gamma', [x \mapsto \ell]$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^s$

By application of T-∨-SUCC and T-R-SHIFT we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$.

In both subcases, we have $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$. Then the desired result follows by T-ENOUGH?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau} \text{ (T-ENOUGH?)}$$

Case 5. [D-Loc-Read]**.** The proof is similar to the previous case.

□

Lemma 7 (Equality of trimmed volatile contexts) *If*

(i) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$
(ii) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,
(iii) $\vdash^{\mathtt{Md}}_{\gamma''} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma''$,
(iv) $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ *and* $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$
(v) $\gamma_0 \subseteq \gamma$ *and* $\gamma_0 \subseteq \gamma''$
then $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

Proof: We need to show that $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$. The proof proceeds by proving each direction separately:

Ad ⇒. Let $x : \downarrow\uparrow A@q \in \Sigma'$. Then by (i), we have

(1) $x : \downarrow\uparrow A@q \in \Sigma$
(2) $\gamma_0 = [x \mapsto l], \gamma_0'$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma''$. From (2) and $\gamma_0 \subseteq \gamma''$, it follows that $\exists \gamma_0'' \supseteq \gamma_0'$ such that $\gamma'' = [x \mapsto l], \gamma_0''$ $(*)$. By (iv) and (3), we have that $l \in dom(\mathsf{V}')$. So, $\exists \mathsf{V}_0'$, v such that $\mathsf{V}' = \mathsf{V}_0', l \hookrightarrow v$ $(**)$ and $\cdot \vdash v : \uparrow A$ $(***)$. Note that inverting (iii) via V-Loc yields the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma_0''} \mathsf{N}' \mid \mathsf{V}_0' : \Omega \mid \Sigma_0''$ (†) and $q = \mathtt{Ck}$ (‡). Then, it follows by V-Loc applied to $(*)$, $(**)$, $(***)$, (†), (‡) that $x : \downarrow\uparrow A@q \in \Sigma''$.

Ad ⇐. Let

(1) $x : \downarrow\uparrow A@q \in \Sigma''$
(2) $\gamma_0 = [x \mapsto l], \gamma_0'$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma'$. To this end, it suffices to show that $x : \downarrow\uparrow A@q \in \Sigma$. By (3) and $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$, we have that $l \in dom(\mathsf{V})$. Therefore, $\exists\, \mathsf{V}_0', v$ such that $\mathsf{V} = \mathsf{V}_0', l@q \hookrightarrow v$ $(*)$, $\cdot \vdash v : \uparrow A$ $(**)$, and $q = \mathtt{Ck}$. From (2) and $\gamma_0 \subseteq \gamma$, we have that $\exists\, \gamma' \supseteq \gamma_0'$ such that $\gamma = [x \mapsto l], \gamma'$. Note that inverting (ii) via V-Loc yields the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}_0' : \Omega \mid \Sigma_0$ $(***)$. By V-Loc applied to $(*)$, $(**)$, $(***)$, $q = \mathtt{Ck}$, and $\gamma = [x \mapsto l], \gamma'$, we have

$$x : \downarrow\uparrow A@q \in \Sigma$$

Then it follows by definition 28 applied to (i) that

$$x : \downarrow\uparrow A@q \in \Sigma'.$$

We have shown that $x : \downarrow\uparrow A@q \in \Sigma'$ iff $x : \downarrow\uparrow A@q \in \Sigma''$, $\gamma = [x \mapsto l], \gamma'$, and $l \in dom(\mathsf{V})$. The desired result holds by definition 28. □

Lemma 8 (Well-formedness of smaller memories) *If*

(i) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,

(ii) $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$,

(iii) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma')$, *and*

(iv) $\gamma' \subseteq \gamma$

then $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$.

Proof: By (2), observe that $\mathsf{V} \supseteq \mathsf{V}''$. The proof proceeds by induction on the size of $\mathsf{V} - \mathsf{V}''$.

Base case: $|\mathsf{V} - \mathsf{V}''| = 0$**.** If $|\mathsf{V} - \mathsf{V}''| = 0$, then $\mathsf{V} = \mathsf{V}''$ and the desired result holds by assumption (1).

Inductive case. Let $|\mathsf{V} - \mathsf{V}''| = k + 1$ where $|\mathsf{V}_0 - \mathsf{V}''| = k$ where $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$ and $\cdot \vdash v : \uparrow A$. Let $x{:}\downarrow\uparrow A@q \in \Sigma$ such that $\Sigma = \Sigma_0, (x{:}\downarrow\uparrow A@q)$. By inversion on V-Loc applied to the assumed judgment:

$$\frac{\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 : \Omega \mid \Sigma_0 \qquad \gamma = [x \mapsto l], \gamma_0 \qquad \mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v \qquad \cdot \vdash v : \uparrow A}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma_0, (x{:}\downarrow\uparrow A@q)}\ \text{V-Loc}$$

we learn that

(1) $\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 : \Omega \mid \Sigma_0$

(2) $\gamma = [x \mapsto l], \gamma_0$

(3) $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$

(4) $\cdot \vdash v{:}\uparrow A$

Now we need to show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$.

Ad $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$**.** To show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it suffices to show that $l \notin dom(\mathsf{V}')$:

By assumption, it follows that $l \in \mathsf{V} - \mathsf{V}''$, or equivalently, $l \in \mathsf{V}$ and $l \notin \mathsf{V}''$. Now suppose that $l \in dom(\mathsf{V}')$. Then it follows by (ii) that $l \in \mathsf{V}''$, a contradiction. Therefore, we have $l \notin dom(\mathsf{V}')$.

Therefore, $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ follows by $l \notin dom(\mathsf{V}')$ and (ii).

Ad $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$**.** To show $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we first need to show that

(a) $dom(\mathsf{V}') \subseteq dom(\mathsf{V}_0)$

(b) $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$

(c) $\gamma \supseteq \gamma'$

(d) $\gamma_0 \supseteq \gamma'$

(a) follows by $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$. Towards (b), note that since $\mathsf{V} \supseteq \mathsf{V}''$ by assumption, we have $dom(\mathsf{V}) \supseteq dom(\mathsf{V}'')$. By $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it follows that $dom(\mathsf{V}'') = dom(\mathsf{V}')$. Therefore, $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$ (b) holds. (c) follows by assumption (iv). By definition, $\gamma = [x \mapsto l], \gamma_0$, so γ_0 is the smallest subset of γ not

containing $x \mapsto l$. Observe that γ' is a subset of γ that does not contain $x \mapsto l$. So, $\gamma \supseteq \gamma_0 \supseteq \gamma'$, which concludes the proof of (d).

$\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$ follows by lemma 36 applied to (iii), (a), (b), (c), and (d).

By the inductive hypothesis applied to (1), $\mathsf{V}'' = \mathsf{V}_0 \restriction \mathit{dom}(\mathsf{V}')$, and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we have $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. This is the desired result.

□

Theorem 4 (preservation for commands) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \ \geq\ 0 : \mathit{nat} \mid \Omega; \Sigma \vdash c : \tau$$

and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed and* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ *and (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$$

then for some Σ'

$$\mathtt{Md} \mid b \ \geq\ 0 : \mathit{nat} \mid \Omega; \Sigma' \vdash c' : \tau$$

where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma'$ *and* $n' \geq 0$*. Moreover* $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *is well-formed.*

Proof: The proof is by induction on the size of c. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Case 1 [D-Let-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e'\, \mathbf{in}\, c}\ \text{(D-Let-step)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \ >\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \ =\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau.$$

By inversion on ($\dagger_1$) via T-Let

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{A} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \ @\ \mathtt{Ck} \vdash c : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau}\ \text{(T-Let)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash c : \tau$

By the application of Theorem 17 on (1) and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}},$$

and $n' \geq 0$. By the following application of the T-Let rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash c : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau}\ (\text{T-Let})$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau^s$ via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau}\ (\text{T-enough?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e' \,\mathbf{in}\, c$ follows by definition 26, vacuously.

Case 2. [D-Let-v].

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \mathit{\ fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e \,\mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell @ \mathtt{Ck} \hookrightarrow e \mid c}\ (\text{D-Let-v})$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau.$$

By inversion on ($\dagger_1$) via T-Let

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash c : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau}\ (\text{T-Let})$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}\downarrow\uparrow A \mathbin{@} \mathtt{Ck} \vdash c : \tau$

The typing judgment (2) completes the proof, if we can show that $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e : \Omega \mid \Sigma, x{:}\downarrow\uparrow A$.

By $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$.

This is enough to prove $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e : \Omega \mid \Sigma, x{:}\downarrow\uparrow A$.

Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c$ follows by definition 26, vacuously because c is not of the form $c';_W c''$.

Case 3 [D-Assign-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'} \text{ (D-Assign-step)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau.$$

By inversion on $(\dagger_1)$ via T-Assign we have $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ and $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} x : \downarrow\uparrow A$.

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} x : \downarrow\uparrow A\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}}} \text{ (T-Assign)}$$

By the application of Theorem 17 on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}},$$

and $n' \geq 0$. By the following application of the T-Assign rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} x : \downarrow\uparrow A\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}}} \text{ (T-Assign)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau$ via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau} \text{ (T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'$ follows by definition 26, vacuously.

Case 4 [D-Assign-V]**.**

$$\frac{\begin{array}{c}Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq UN \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau.$$

By inversion on ($\dagger_1$) via T-Assign

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} x : \downarrow\uparrow A\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}}} \text{ (T-Assign)}$$

we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} x : \downarrow\uparrow A$

By the premise $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$.

Appying V-Loc, we can show that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V}', (\ell@\mathtt{Ck} \hookrightarrow e) : \Omega \mid \Sigma, (x : \downarrow\uparrow A@\mathtt{Ck})$.

Moreover, by T-Skip, T-R-Shift, and T-$\vee$-Succ we have

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}.$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$ via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}} \text{ (T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}$ follows by definition 26, vacuously.

Case 5 [D-Assign-N]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathit{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}} \text{ (D-Assign-N)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on $(\dagger)$ via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau.$$

By inversion on $(\dagger_1)$ via T-Assign

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathit{Wt}} x : \downarrow\uparrow A\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}}} \text{ (T-Assign)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathit{Wt}} x : \downarrow\uparrow A$

By $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$. This is enough to prove $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}', \ell@\mathtt{Ck} \hookrightarrow e \mid \mathsf{V} : \Omega' \mid \Sigma$. Moreover, by T-Skip, T-R-Shift, and T-$\vee$-Succ we have

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}.$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$ via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}} \text{ (T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$ follows by definition 26, vacuously.

Case 6 [D-If]**.**

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\, e\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\, e'\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2} \text{ (D-If)}$$

By the first premise $n > 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{if}\, e\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2 : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{if}\, e\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2 : \tau.$$

By inversion on ($\dagger_1$) via T-If

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool} \quad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \tau \quad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{if}\, e\, \mathbf{then}\, c_1\, \mathbf{else}\, c_2 : \tau} \text{ (T-If)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \tau$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau$.

By the application of Theorem 17 on (1), it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_{bool},$$

and $n' \geq 0$. By the following application of the T-If rule we get

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_{bool} \quad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \tau \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau} \text{ (T-If)}$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau$ via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau^s$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau} \text{ (T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2$ follows by definition 26, vacuously.

Case 7. [D-If-tt]**.**

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \text{ (D-If-tt)}$$

By the first premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau.$$

By inversion on ($\dagger_1$) via T-If

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} \mathtt{tt} : \mathsf{C}^{\mathtt{Md}}_{bool} \quad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \tau \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau} \text{ (T-If)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} \mathtt{tt} : \mathsf{C}^{\mathtt{Md}}_{bool}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \tau$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1$ follows by definition 26, vacuously because c_1 is not of the form $c';_W c''$

The typing judgment (2) completes the proof, because $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ holds by assumption.

Case 8 [D-IF-FF]**.** Similar to the previous case.

Case 9 [D-SEQ]**.**

$$\frac{n > 0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1; c_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma|\mathsf{V}} c_2} \text{ (D-SEQ)}$$

By the premise $n > 0$. By inversion on (†) via T-ENOUGH? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1; c_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1; c_2 : \tau^s.$$

By inversion on $(\dagger_1)$ via T-SEQ

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \mathsf{C}_{\mathtt{unit}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1; c_2 : \tau} \text{ (T-SEQ)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 : \mathsf{C}_{\mathtt{unit}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_2 : \tau^s$

Put $W = \gamma \mid \mathsf{V}$. We now want to show that $\Sigma = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. By definition 28, it is straightforward to see that $\mathsf{trim}(\Sigma, \mathsf{V}, \gamma) \subseteq \Sigma$. We need to show $\Sigma \subseteq \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. Let $x : \downarrow\uparrow A@q \in \Sigma$ be arbitrary and write $\Sigma = \Sigma', (x : \downarrow\uparrow A@q)$. Then by inversion on the well-formedness definition V-LOC:

$$\frac{\mathsf{V} = \mathsf{V}', \ell \mathbin{@} q \hookrightarrow v \qquad \begin{array}{c} \vdash^{\mathsf{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}' : \Omega \mid \Sigma' \\ q = \mathsf{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \end{array} \qquad \cdot \vdash v : \uparrow A}{\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma', (x{:}\downarrow\uparrow A@q)} \text{ (V-LOC)}$$

we know that $\gamma = \gamma', [x \mapsto \ell]$ with $\ell \in dom(\mathsf{V})$. From here, definition 28 implies that $x \in \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. Because $x \in \Sigma$ was arbitrary, we have shown that $\Sigma \subseteq \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. Therefore, $\Sigma = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$.

By the following application of the T-SEQ-D rule

$$\frac{\begin{array}{c} W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1 : \mathsf{C}_{\mathtt{unit}} \\ \Sigma = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma) \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_2 : \tau \end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau} \text{ (T-SEQ-D)}$$

we learn that $\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau^s$ via lemma 30, we get $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau^s$.

In both subcases, the desired result follows by T-ENOUGH?:

$$\frac{\begin{array}{c} \mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau^s \\ \mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau^s \end{array}}{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1;_W c_2 : \tau} \text{ (T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma|\mathsf{V}} c_2$ follows by definition 26 since $\gamma \subseteq \gamma$ and $dom(\mathsf{V}) \subseteq dom(\mathsf{V})$.

Case 10 [D-SEQ-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{ (D-SEQ-STEP)}$$

The premises yield

(a) $n > 0$

(b) $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$

By assumption, note that

- $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_W c_2$ is well-formed.

Put $W = \gamma_0 \mid \mathsf{V}_0$. Then by definition 26, we have $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$. Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1$ is well-formed, which vacuously follows by definition 26 because c_1 does not have the form $c';_{W'} c''$.

By inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau$ via T-ENOUGH? rule, we have

($\dagger_1$) $\quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau.$$

By inversion on $(\dagger_1)$ via T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \\ \Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau} \text{(T-SEQ-D)}$$

we learn that

(1) $W = \gamma_0 \mid \mathsf{V}_0$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}$

(3) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$

(4) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau$

By the inductive hypothesis applied to (2), (b), the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1$, $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, and $n \geq 0$, we get $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1' : \mathsf{C}_{\mathtt{unit}}$, where

(i) $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma''$,

(ii) $n' \geq 0$, and

(iii) $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$ is well-formed.

We now need to show that $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$. Observe that $c_1 \neq c';_W c''$ because $c_1;_W c_2$. By lemma 38 $c_1 \neq c';_W c''$, it follows that $dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$ and $\gamma \subseteq \gamma'$. Then it follows by $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}_1)$ and $\gamma_0 \subseteq \gamma_1$ (as shown above), that $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}_1) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma_1 \subseteq \gamma'$. Hence, $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$.

It suffices to show $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

By lemma 36, it follows that $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

Using (1), (3), (4), and $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1' : \mathsf{C}_{\mathtt{unit}}$, we can apply T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1' : \mathsf{C}_{\mathtt{unit}} \\ \Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0) \qquad \mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \mid \; \Omega; \Sigma' \vdash c_2 : \tau \end{array}}{\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1';_W c_2 \; : \tau} \text{(T-SEQ-D)}$$

We now need to show that $\mathtt{Md} \mid \; b \;= 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1';_W c_2 : \tau$. The proof proceeds in two subcases based on $\mathtt{Md}$:

Case $\mathtt{Md} = \mathsf{jit}$**.** By $\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1';_W c_2 \; : \tau$ via lemma 30, we can see that $\mathtt{Md} \mid \; b \;= 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1';_W c_2 \; : \tau$.

Case $\mathtt{Md} = \mathsf{aID}(c_0)$**.** By $(\dagger_2)$ via lemma 31, we can see that $\mathtt{Md} \mid \; b \;= 0 : \mathtt{nat} \mid \; \Omega; \Sigma'' \vdash c_1';_W c_2 \; : \tau$.

In both cases, $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma'' \vdash c_1';_W c_2 : \tau$.

The desired result follows by T-ENOUGH?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma'' \vdash c_1';_W c_2 : \tau \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma'' \vdash c_1';_W c_2 : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma'' \vdash c_1';_W c_2 : \tau} \text{ (T-ENOUGH?)}$$

Observe that by definition 26 applied to $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$ (as shown above), $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2$ is well-formed. In the case where c_1' is of the form $c';_W c''$, the rule stepping c_1 must be D-SEQ since $c_1 \neq c';_W c''$. Thus, observe that $\gamma' \subseteq \gamma'$ and $dom(\mathsf{V}') \subseteq dom(\mathsf{V}')$, and hence it follows by definition 26 that $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2$ is well-formed.

Case 11 [D-SEQ-V]**.**

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2} \text{ (D-SEQ-V)}$$

Observe that $n = n' + 1$ (from the premise of D-SEQ-V) implies $n > 0$, and hence $b > 0$.

By assumption, we have

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip};_W c_2 : \tau$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$.

By inversion on T-SEQ-D, we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash c_2 : \tau$

(3) $W = \gamma' \mid \mathsf{V}'$ (from the premise of D-SEQ-V)

(4) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma')$

We now need to show that $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. Since $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2$ is well-formed (by assumption), it follows by definition 26 that $\gamma' \subseteq \gamma$. Therefore, the desired result follows by lemma 37 applied to $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, the premise $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$, (4), and $\gamma' \subseteq \gamma$. Observe that (2) yields the desired result where $\vdash^{\mathtt{Md}}_{\gamma''} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2$ follows by definition 26, vacuously because c_2 is not of the form $c';_W c''$.

□

Theorem 7 (Preservation for programs) *Consider* $b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}$*, a nonvolatile memory* N *and a bijective map* γ *that matches qualifiers and types from variables in* Ω *to locations in* N*. For any* n:$\mathtt{nat} \geq 0$*, if we have* $[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid p \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma' \mid n' \mid \mathsf{N}' \mid p'$*, then* $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$*, with* γ *remaining a bijective map from* Ω *to* N'*.*

Proof: By a structural induction on the typing derivation, and case distinction on the step.
Case 1.

$$\frac{n > 0 \qquad n' > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid c; p' \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'} \text{ (P-SEQ)}$$

By inversion on the typing rules, we know that $b\mathcal{R}0 : \mathtt{nat} \mid \Omega \mid \cdot \vdash c : \mathsf{C}_{\mathtt{unit}}$ and $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow\mathsf{C}_{\mathtt{unit}}$ By preservation for commands, we know that γ' is well-formed for N' and V' at the jit mode with respect to Ω and some Σ. By definition of well-formedness, we know that for some $\gamma_1 \subseteq \gamma'$, we have γ_1 is well-formed for Ω and N'. But we know that $\gamma \subseteq \gamma'$ is well-formed for Ω. This means that $\gamma = \gamma'$ and thus γ is a bijective map that matches qualifiers and types from variables in Ω to locations in N'.

Case 2.

$$\frac{\begin{array}{c} n > 0 \\ \mathsf{InitWorld}_d(\mathsf{N}; \rho) = \mathsf{N}_0, V_0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* \\ [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{Ckpt}[(\mathsf{aID}; \rho)](c_0); p \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p} \text{ (P-CK}_{\text{PT}}\text{)}$$

By inversion on the typing rules, we know that $b\mathcal{R}0 : \mathtt{nat} \mid \Omega_0 \mid \Sigma_0 \vdash c : \mathsf{C}_{\mathtt{unit}}$ and $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow\mathsf{C}_{\mathtt{unit}}$ By preservation for commands, we know that γ' is well-formed for N' and V' at the jit mode with respect to Ω and some Σ. By definition of well-formedness and FinWorld, we know that for some $\gamma_1 \subseteq \gamma'$, we have γ_1 is well-formed for Ω and N_1. But we know that $\gamma \subseteq \gamma'$ is well-formed for Ω. This means that $\gamma = \gamma'$ and thus γ is a bijective map that matches qualifiers and types from variables in Ω to locations in N_1.

□

A.2 Fundamental Theorem of Logical Relation

Theorem 5 (Fundamental theorem) *If* $b : \mathtt{nat} \mid \Omega \vdash p : \uparrow\mathsf{C}_{\mathtt{unit}}$, *then* $b : \mathtt{nat} \mid \Omega \Vdash p : \uparrow\mathsf{C}_{\mathtt{unit}}$.

Proof: The proof is by induction on the static typing derivation for p and considering the last step in the derivation.

Case 1. Suppose that $p = \mathsf{Ckpt}[\mathsf{aID}, \rho](c); p'$. Figures A.1 and A.2 explain the proof for the case where T-P-CKPT is the last step of the derivation.

$$\frac{\begin{array}{c} \Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho) \qquad \mathtt{Sig} = \{\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash c : \mathsf{C}_{\mathtt{unit}}\} \\ \mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega';\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow\mathsf{C}_{\mathtt{unit}} \end{array}}{b : \mathtt{nat} \mid \Omega \vdash \mathsf{Ckpt}[\mathsf{aID}, \rho](c); p' : \uparrow\mathsf{C}_{\mathtt{unit}}} \text{ (T-P-CKPT)}$$

By inversion, we know that

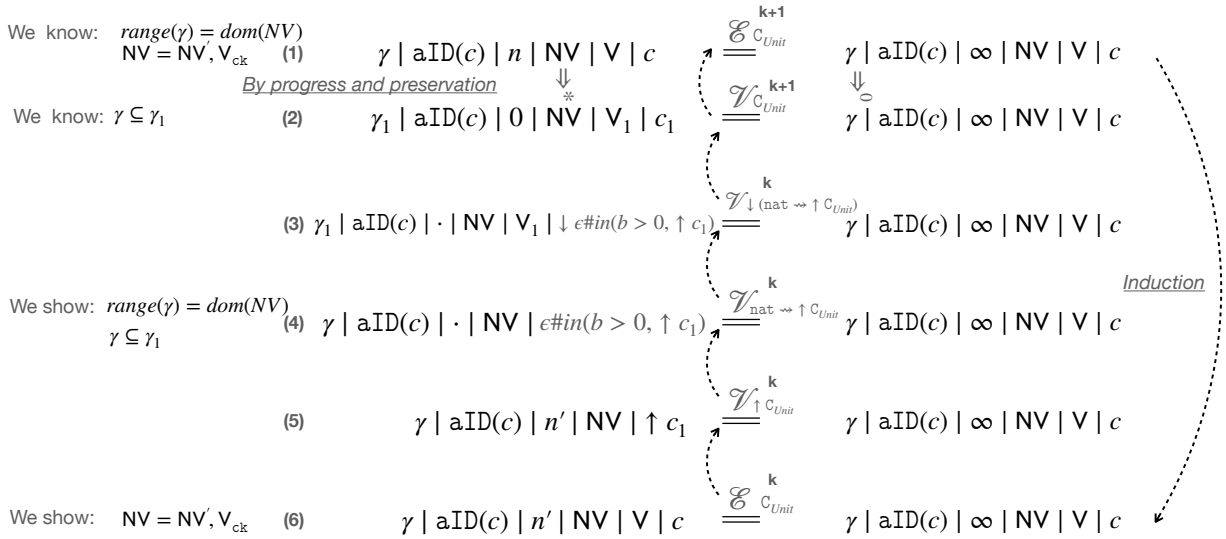

Figure A.1: Proof of the fundamental theorem for aID - inductive case

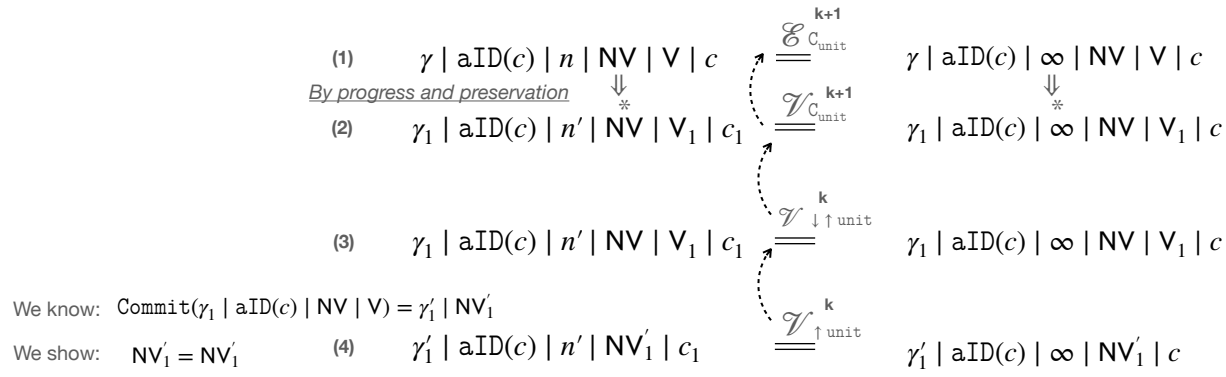

Figure A.2: Proof of the fundamental theorem for aID - base case

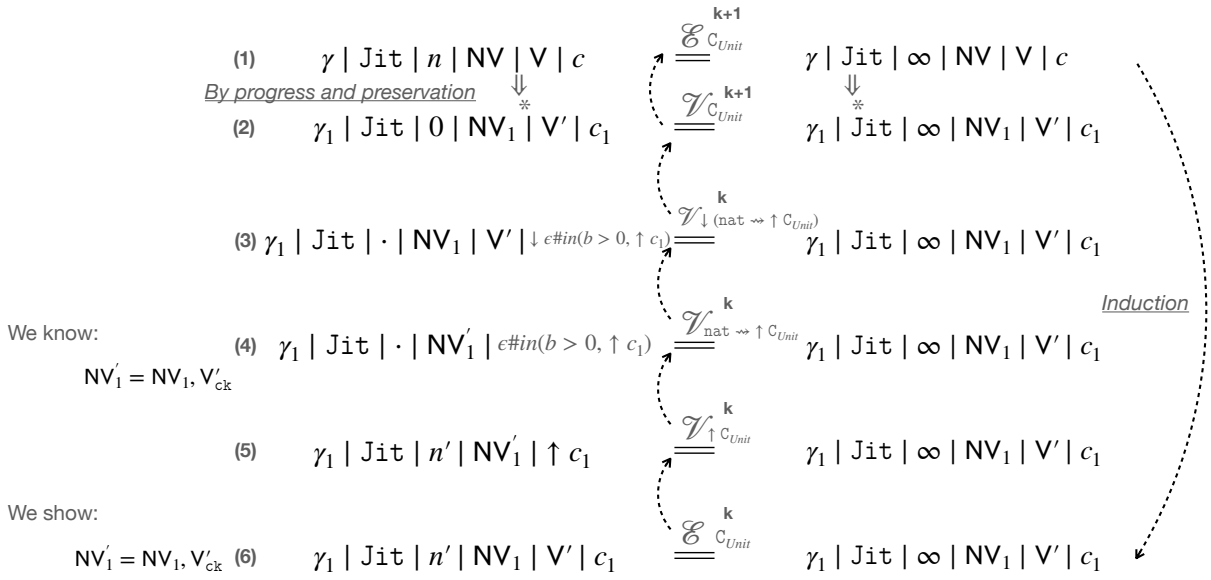

Figure A.3: Proof of the fundamental theorem for Jit - inductive case

(1) $\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho)$

(2) $\mathtt{Sig} = \{\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash c : \mathsf{C}_{\mathtt{unit}}\}$

(3) $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega';\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}}$

(4) $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

By (1) and the definition of $\mathsf{InitWorld}_t$, we have that $\Omega' = \Omega'', \Sigma_{\mathsf{ck}}$. Observe that the inductive hypothesis asserts that $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$ implies $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$. By applying the inductive hypothesis to (4), we learn that

$$b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}.$$

To complete the proof, we need to establish

$$\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}.$$

By definition of logical relation, this is equivalent to showing that c is related to itself in the term interpretation for arbitrary n_0, m_0, γ_0, N_0, and V_0 where $\mathsf{N}_0 \mid \mathsf{V}_0 \Vdash \gamma_0 :: \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$. In particular, this condition establishes that $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$, and hence, $\mathsf{N}_0 = \mathsf{N}'_0, \mathsf{V}_{0\mathsf{ck}}$ and $\mathit{range}(\gamma_0) = \mathit{dom}(\mathsf{N}_0)$. By assumption, p does not contain any worlds W, so it follows that c does not contain any worlds W. Therefore, it follows by definition 26 that $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ and $\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ are well-formed.

We need to show that $\forall n_0$:

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$$

The proof proceeds by induction on m_0:

Base case ($m_0 = 0$). When $m_0 = 0$, the proof is trivial and the desired result follows immediately by the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$:

$$(\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^0$$

Inductive case ($m_0 = k + 1$ ($\exists k$)). If $m_0 = k + 1$, we need to show that

$$(\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

such that

(i) $\exists.(\gamma_1 \mid \mathsf{aID}(\mathsf{c}) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1)$ such that $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1 \mid \mathsf{aID}(\mathsf{c}) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1$ AND

(ii) $\exists.(\gamma_2 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2)$ such that $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_2 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2$ AND

(iii) $(\gamma_1 \mid \mathsf{aID}(\mathsf{c}) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$,

By the progress and preservation for commands applied to (2) and (3), we know that the first configuration

$$\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$$

can take multiple steps until it becomes a value configuration that continues to be well-typed. Observe that in the mode $\mathsf{aID}(c)$, nonvolatile memory remains unchanged. We prove this by induction on n_0:

Base case. If $n_0 = 0$, then the configuration is a value.

Inductive case. Suppose that $n_0 = n_0' + 1$ ($\exists n_0'$). Since $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}}$ and $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$, it follows by progress for commands that either $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is a value or $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is not a value, in which case $\exists\, \gamma_1'' \mid \mathsf{aID}(\mathsf{c}) \mid n_1'' \mid \mathsf{N}_0 \mid \mathsf{V}_1'' \mid c_1''$ such that

$$\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow \gamma_1'' \mid \mathsf{aID}(\mathsf{c}) \mid n_1'' \mid \mathsf{N}_0 \mid \mathsf{V}_1'' \mid c_1''$$

where $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}}$, $\vdash_{\gamma_1''}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0'' : \Omega' \mid \Sigma'$, and $\gamma_1'' \mid \mathsf{aID}(\mathsf{c}) \mid n_1'' \mid \mathsf{N}_0 \mid \mathsf{V}_1'' \mid c_1''$ is well-formed (by theorem 5 because $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is well-formed).

By the inductive hypothesis, $\gamma_1'' \mid \mathsf{aID}(\mathsf{c}) \mid n_1'' \mid \mathsf{N}_0 \mid \mathsf{V}_1'' \mid c_1'' \rightarrow^* \gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1'$ where $\gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1'$ is well-formed and a value, $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathtt{unit}}$, and $\vdash_{\gamma_1'}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_1' : \Omega' \mid \Sigma''$. By head expansion, we establish that

$$\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1'$$

We have just shown that (i) holds.

The proof proceeds in two subcases, depending on the value of n_1':

Subcase $n_1' = 0$**.** Observe that (ii) holds vacuously because $\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ in 0 steps.

We now need to show that (iii) holds:

$$(\gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1', \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

This step corresponds to (2) in the figure where we need to show that the configurations are in the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$. By the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$, and because $n_1' = 0$, this is equivalent to showing

$$(\gamma_1' \mid \mathsf{aID(c)} \mid \cdot \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid \downarrow \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k$$

This step corresponds to (3) in the diagram. At this step we show the above relation holds by its definition:

(vi) $\mathtt{Pw0ff}(\gamma_1', \mathsf{aID}(c), \mathsf{N}_0, \mathsf{V}_1') = \gamma_1'' \mid \emptyset$ AND

(vii) $(\gamma_1'' \mid \mathsf{aID}(c) \mid \cdot \mid \mathsf{N}_0 \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$

To show (vi), we need to show that $range(\gamma_1'') = dom(\mathsf{N}_0)$ where γ_1'' is the largest restriction of γ_1' such that this condition holds. Observe that the desired result follows immediately by the assumptions $\gamma_0 \subseteq \gamma_1'$ and $range(\gamma_0) = dom(\mathsf{N}_0)$, where $\gamma_1'' = \gamma_0$. Hence, we need to show

$$(\gamma_0 \mid \mathsf{aID}(c) \mid \cdot \mid \mathsf{N}_0 \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

This corresponds to step (4) in the diagram. By definition of the value relation at the type $\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing the following:

$$\forall n_1' > 0.(\gamma_0 \mid \mathsf{aID}(c) \mid n_1' \mid \mathsf{N}_0 \mid \uparrow c_1', \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

Fix an arbitrary n_1. We need to show that

$$(\gamma_0 \mid \mathsf{aID}(c) \mid n_1 \mid \mathsf{N}_0 \mid \uparrow c_1', \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

This corresponds to step (5) in the diagram. By the definition of value relation at the type $\uparrow \mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing

(viii) $\texttt{restore}(\gamma_0 \mid \mathsf{aID}(c) \mid \mathsf{N}_0 \mid c_1') = \mathsf{N}_0 \mid \mathsf{V}_0' \mid c_0'$ (for some V_0', c_0') AND

(ix) $(\gamma_0 \mid \mathsf{aID}(c) \mid n_1 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0', \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0' \mid c) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^k$

By assumption, we have that $\mathsf{N}_0 = \mathsf{N}_0', \mathsf{V}_{0\texttt{ck}}$. By the definition of $\texttt{restore}(\gamma_0 \mid \mathsf{aID}(c) \mid \mathsf{N}_0 \mid c_1') = \mathsf{N}_0 \mid \mathsf{V}_0' \mid c_0'$ we have that $\mathsf{N}_0 = \mathsf{N}_0', \mathsf{V}_{0\texttt{ck}}'$. Therefore, $\mathsf{V}_0' = \mathsf{V}_0$. Now we need to show that

$$(\gamma_0 \mid \mathsf{aID}(c) \mid n_1 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^k$$

which follows directly by the inductive hypothesis. Propagating up the cascade, we learn that since n_1 in step (4) was arbitrary, the value relation holds for all n_1. In summary, we have just shown that

$$(\gamma_0 \mid \mathsf{aID}(c) \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+1}$$

Subcase $n_1' > 0$**.** Since $n_1' > 0$ and $\gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1'$ is a value, we step the second configuration to completion:

$$\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \mapsto^* \gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1'$$

Now we want to show that

$$(\gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1', \gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow\uparrow \mathsf{C}_{\texttt{unit}}]\!]^{k+1}$$

By the value interpretation at type $\mathsf{C}_{\texttt{unit}}$ and $n_1' > 0$, we need to show that

$$(\gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid n_1' \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1', \gamma_1' \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^k$$

By definition of the value interpretation at the type $\downarrow\uparrow \texttt{unit}$, this is equivalent to showing

(x) $\texttt{Commit}(\gamma_1' \mid \mathsf{aID}(c) \mid \mathsf{N}_0 \mid \mathsf{V}_1') = \gamma_1 \mid \mathsf{N}_1', \mathsf{V}_1''$ AND

(xi) $\texttt{Commit}(\gamma_1' \mid \mathsf{aID}(c) \mid \mathsf{N}_0 \mid \mathsf{V}_1') = \gamma_2 \mid \mathsf{N}_2', \mathsf{V}_2''$ AND

(xii) $(\gamma_1 \mid \mathsf{aID}(c) \mid n_1' \mid \mathsf{N}_1', \mathsf{V}_1'' \mid \mathsf{skip}, \gamma_2 \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N}_2', \mathsf{V}_2'' \mid \mathsf{skip})$
$\in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^k$

By (x) and (xi), we observe that $\gamma_1 = \gamma_2$ and $\mathsf{N}_1', \mathsf{V}_1'' = \mathsf{N}_2', \mathsf{V}_2''$.

Therefore, we can use the value interpretation at type $\uparrow \texttt{unit}$ to prove (xii):

$$(\gamma_1 \mid \mathsf{aID}(c) \mid n_1' \mid \mathsf{N}_1', \mathsf{V}_1'' \mid \mathsf{skip}, \gamma_2 \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N}_2', \mathsf{V}_2'' \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^k$$

which holds by definition of logical relation. This is the last piece we needed in order to prove the desired result:

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

In general, we have that $(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$ where $\mathsf{N}_0 \mid \mathsf{V}_0 \Vdash \gamma_0 :: \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$. Since $n_0, m_0 \geq 0$, γ_0, N_0, and V_0 were arbitrarily chosen, this result holds for all $n, m \geq 0, \gamma, \mathsf{N}, \mathsf{V}$. Therefore, it follows by definition of logical relation that $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$. Finally, the desired result follows by application of P-Ckpt-semantic.

$$\frac{\begin{array}{c}\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho) \\ \mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}\end{array}}{b : \mathtt{nat} \mid \Omega \Vdash \mathsf{Ckpt}[\mathsf{aID}, \rho](c); p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-Ckpt-semantic)}$$

Case 2. Suppose that $p = c; p'$. Figure A.3 explains the proof for the case where T-P-seq is the last step of the derivation.

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_\emptyset c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-seq)}$$

By inversion, we know that

(1) $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_\emptyset c : \mathsf{C}_{\mathtt{unit}}$

(2) $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

From (1), we know that c is well-typed for static context Ω and $\Sigma = \cdot$. By applying the inductive hypothesis to (2), we learn that $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$. To complete the proof, we need to show that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$. By the definition of logical relation, this is equivalent to establishing that c is related to itself in the term interpretation for arbitrary n_0, m_0, γ_0, N_0, and V_0 where $\mathsf{N}_0 \mid \mathsf{V}_0 \Vdash \gamma_0 :: \Omega \mid \Sigma$. In particular, this condition establishes the condition that $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega \mid \Sigma$. By assumption, the program p, and by extension the command c, does not contain any worlds W, so it follows by definition 26 that $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ and $\gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ are well-formed configurations. We need to show that $\forall c.\, \mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_\emptyset c : \mathsf{C}_{\mathtt{unit}}$ and $\forall \mathsf{N}_0, \mathsf{V}_0, \gamma_0, n_0. \mathsf{N}_0 \mid \mathsf{V}_0 \Vdash \gamma_0 :: \Omega \mid \Sigma$:

$$(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$$

The proof proceeds by induction on m_0:

Base case ($m_0 = 0$). When $m_0 = 0$, the proof is trivial and the desired result follows immediately by the definition of logical relation.

Inductive case ($m_0 = k + 1$ ($\exists k$)) Consider the case where $m_0 = k + 1$.

We need to show that

$$(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

By the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing

(i) $\exists . \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1$ such that $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1$

(ii) $\exists . \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2$ such that $\gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2$

(iii) $(\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$

By the progress and preservation theorems, we know that the first configuration can take multiple steps until it becomes a value configuration that continues to be well-typed. We proceed by induction on n_0:

Base case. If $n_0 = 0$, then the configuration is a value.

Inductive case. Suppose that $n_0 = n_0' + 1$ ($\exists n_0'$). Since $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega \mid \cdot$ and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_\emptyset c : \mathsf{C}_{\mathtt{unit}}$, it follows by theorem 2 (Progress for commands) that either $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is a value or $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is not a value, in which case $\exists\ \gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$ such that

$$\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow \gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$$

where $\gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$ is well-formed, $\vdash^{\mathsf{jit}}_{\gamma_1''} \mathsf{N}_1'' \mid \mathsf{V}_1'' : \Omega'' \mid \Sigma''$ and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega''; \Sigma'' \vdash_\emptyset c_1'' : \mathsf{C}_{\mathtt{unit}}$ (by theorem 5 because $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is well-formed and $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega \mid \cdot$).

By the inductive hypothesis, $\gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1'' \rightarrow^* \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ where $\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ is well-formed and a value, $\vdash^{\mathsf{jit}}_{\gamma_1'} \mathsf{N}_1' \mid \mathsf{V}_1' : \Omega''' \mid \Sigma'''$, and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega'''; \Sigma''' \vdash_\emptyset c_1' : \mathsf{C}_{\mathtt{unit}}$. By head expansion, we establish that

$$\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

Analogously, we step the second configuration until c_1' with the exact same steps as in the first configuration by theorems 2 (Progress for commands) and 5:

$$\gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

We have just shown (i) and (ii). The proof of (iii) corresponds to step (2) in the diagram. Proving that the configurations are in the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$ is equivalent to showing

(iv) $n_1' = 0 \wedge (\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\downarrow \varepsilon \# \mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1')$
$\in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k$ OR

(v) $n_1' > 0 \ \wedge \ (\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]^k$

The proof proceeds in two subcases depending on the value of n_0':

Case $n_1' = 0$**.** If $n_1' = 0$, then we need to show that

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\downarrow \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow\uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k$$

This proof corresponds to point (3) in the diagram. To show that the two configurations are in the value interpretation at type $\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})$, we need to show:

(vi) $\mathtt{Pw0ff}(\gamma_1', \mathsf{jit}, \mathsf{N}_1', \mathsf{V}_1') = \gamma_1' \mid \mathsf{V}_1'$ AND

(vii) $(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$

To show (vii), we need to show that the configurations are in the value interpretation of $\mathtt{nat} \rightsquigarrow\uparrow \mathsf{C}_{\mathtt{unit}}$, which corresponds to step (4) in the diagram:

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

This is equivalent to proving that

$$\forall n > 0.(\gamma_1' \mid \mathsf{jit} \mid n \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

This step corresponds to point (5) in the diagram. Fix an arbitrary n'. We need to show that

$$(\gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

According to the value interpretation at type $\uparrow \mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing:

(viii) $\mathtt{restore}(\gamma_1', \mathsf{jit}, (\mathsf{V}_1', \mathsf{N}_1'), c_1') = \mathsf{N}' \mid \mathsf{N}'' \mid c_1'$ where $\mathsf{V}_1', \mathsf{N}_1' = \mathsf{N}', \mathsf{N}_{\mathsf{ck}}''$ AND

(ix) $(\gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{N}_{\mathsf{ck}}'' \mid c_1', \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^k$

From (viii), we have that $\mathsf{V}_1' = \mathsf{N}_{\mathsf{ck}}''$ and $\mathsf{N}' = \mathsf{N}_1'$. Recalling from above that $\vdash^{\mathsf{jit}}_{\gamma_1'} \mathsf{N}_1' \mid \mathsf{V}_1' : \Omega''' \mid \Sigma'''$ (which is equivalent to $\mathsf{N}_1' \mid \mathsf{V}_1' \Vdash \gamma_1'::\Omega''' \mid \Sigma'''$ for jit) and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega'''; \Sigma''' \vdash_\emptyset c_1' : \mathsf{C}_{\mathtt{unit}}$, we can apply the inductive hypothesis to prove (ix). Therefore, we have established that

$$(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

Case $n_0' > 0$. If $n_0' > 0$, then we need to show that

$$(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]^k$$

From the value interpretation at type $\downarrow\uparrow \mathtt{unit}$, we have

(x) $\mathtt{Commit}(\gamma_1' \mid \mathsf{jit} \mid \mathsf{N}_1' \mid \mathsf{V}_1') = \gamma_1 \mid \mathsf{N}_1$

(xi) $\mathtt{Commit}(\gamma_1' \mid \mathsf{jit} \mid \mathsf{N}_1' \mid \mathsf{V}_1') = \gamma_2 \mid \mathsf{N}_2$

(xii) $(\gamma_1 \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1 \mid \mathsf{skip}, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]^k$

From (x) and (xi), we have that $\gamma_1 = \gamma_2$ and $\mathsf{N}_1 = \mathsf{N}_2$. Therefore, the desired result follows by the value interpretation at type $\downarrow\uparrow \mathtt{unit}$:

$$(\gamma_1 \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1 \mid \mathsf{skip}, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]^k$$

Therefore, we have shown that $(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$ where $\mathsf{N}_0 \mid \mathsf{V}_0 \Vdash \gamma_0 :: \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$. Since $n_0, m_0 \geq 0$, γ_0, N_0, V_0 were arbitrary, this result holds for all $n, m \geq 0$, γ, N, V. Therefore, it follows by the definition of logical relation that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$. Finally, the desired result follows by application of P-SEQ-SEMANTIC.

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-SEQ-SEMANTIC)}$$

□

A.3 Adequacy

Theorem 6 (Adequacy) *Consider $b : \mathtt{nat} \mid \Omega \Vdash p : \mathsf{C}_{\mathtt{unit}}$, a nonvolatile memory N and a bijective map γ that matches qualifiers and types from variables in Ω to locations in N. The triple of p, N, and γ is idempotent.*

Proof: The proof is by cases according to the execution mode.

Stepping a JIT block. Consider a program of form $[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid c; p'$ that can take a step using the D-P-SEQ rule to $[\chi'' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'$. By inversion on the D-P-SEQ rule,

$$\frac{\begin{array}{c} n > 0 \\ n' > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid c; p' \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'} \text{ (D-P-SEQ)}$$

Suppose that the command c is successfully executed to completion with possibly m power failures and $m + 1$ tries along the way:

- $n > 0$
- $n' > 0$
- $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$

Our goal is to show that we can run the configuration with ∞, an infinite level of energy, as:

$[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2 \mid \mathsf{skip}$.

Figure B.2 shows the main idea of the proof. Here we provide more details. We first need to establish that the configuration with n level of energy is related to itself when provided with an infinite energy level ∞ for every index, including $m+1$ (point (1) in Figure 3.23).

By inversion on T-P-SEQ-SEMANTIC and the assumption $b : \mathtt{nat} \mid \Omega \Vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}$,

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}}\ \text{(P-SEQ-SEMANTIC)}$$

we learn that

(i) $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$

(ii) $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

By the definition of logical relation applied to (i), we have that $\forall n_0, m_0 \geq 0. \forall \gamma_0, \mathsf{N}_0$ s.t. $\mathsf{N}_0 \mid \cdot \Vdash \gamma_0 :: \Omega \mid \cdot$. $(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \cdot \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \cdot \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$.

Instantiating m_0 to be the number of tries $m+1$ (i.e. $m_0 = m+1$), we have that

$$(\gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c, \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$$

To prove the desired result, we use induction to prove the following generalized statement:

- $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in m crashes and
- $(\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$

then for any energy stream χ^0, we have $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2'' \mid \mathsf{V}_2'' \mid \mathsf{skip}$ and $\mathsf{N}_2'' = \mathsf{N}'$.

The proof proceeds by induction on the number of crashes (m):

Base case: $m = 0$ **(*# tries* = 1).** It follows by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$ that

1. $\exists(\gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1')$ s.t. $\gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ AND
2. $\exists(\gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2')$ s.t. $\gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$ AND
3. $(\gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^1$

Since $m = 0$, there are no crashes in (1). So, $n' > 0$ and the first configuration steps to completion via D-STEP where $\mathsf{N}_1' = \mathsf{N}'$ and $\mathsf{V}_1' = \mathsf{V}'$ and $\gamma_1'' = \gamma'$:

$$[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \mathsf{skip}$$

By (2) we know that:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (3) and $n' > 0$, we get that the post steps are related by the value interpretation at type $\downarrow\uparrow \texttt{unit}$. This means that $c_2' = \mathsf{skip}$ and we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}$$

and

$$(\gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \mathsf{skip}, \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^0$$

It then follows by the value relation at type $\downarrow\uparrow \texttt{unit}$ that

1. $\texttt{Commit}(\gamma_1'' \mid \mathsf{jit} \mid \mathsf{N}_1' \mid \mathsf{V}_1') = \gamma^1 \mid \mathsf{N}^1$ where $\mathsf{range}(\gamma^1) = \mathsf{dom}(\mathsf{N}^1)$ and $\gamma^1 \subseteq \gamma_1''$
2. $\texttt{Commit}(\gamma_2'' \mid \mathsf{jit} \mid \mathsf{N}_2' \mid \mathsf{V}_2') = \gamma^2 \mid \mathsf{N}^2$ where $\mathsf{range}(\gamma^2) = \mathsf{dom}(\mathsf{N}^2)$ and $\gamma^2 \subseteq \gamma_2''$
3. $(\gamma^1 \mid \mathsf{jit} \mid n' \mid \mathsf{N}^1 \mid \mathsf{skip}, \gamma^2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^0$

By the definition of commit in the jit mode and (1) and (2), we have $\mathsf{N}^1 = \mathsf{N}_1'$ and $\mathsf{N}^2 = \mathsf{N}_2'$. Thus, it follows by the value interpretation at type $\uparrow \texttt{unit}$:

$$\mathsf{N}' = \mathsf{N}_1' = \mathsf{N}_2'$$

Therefore, we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2' \mid \mathsf{skip}$$

which completes the proof for this subcase.

Inductive case: $m = k + 1 (\exists\, k)$ **(*# tries* $= k + 2$).** It follows by the term interpretation at type $\mathsf{C}_{\texttt{unit}}$ that

(iii) $\exists(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1')$ s.t. $\gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$

(iv) $\exists(\gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2')$ s.t. $\gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$

(v) $(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+2}$

From (iii), we step the first configuration until it becomes a value. It follows by the rule D-STEP that

$$[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

We step the second configuration via D-STEP applied to (iv):

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (v) we have:

$$(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+2}$$

Since there are $m > 0$ crashes, we know that $n_1' = 0$. By application of D-CRASH, we have

$$[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid 0 \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \downarrow \varepsilon \#\mathsf{in}(b > 0; \uparrow c_1')$$

By the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$, observe that the stepped configuration continues to be related to the second configuration:

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^{k+1}$$

By D-S-Jit, we have

$$[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1') \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1', \mathsf{V}_{1\mathsf{Ck}}' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1')$$

By the value interpretation at type $\downarrow (\mathtt{nat} \rightsquigarrow \mathsf{C}_{\mathtt{unit}})$, the post step configuration remains related to the second configuration:

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1', \mathsf{V}_{1\mathsf{Ck}}' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

as by definition of `PwOff` in the jit mode, we have $\mathtt{PwOff}(\gamma_1', \mathsf{jit}, \mathsf{N}_1', \mathsf{V}_1') = \gamma_1' \mid \mathsf{V}_1'$.
By D-charge, for some $n'' > 0$, we have $\chi = n'' :: \chi'$:

$$[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1', \mathsf{V}_{1\mathsf{Ck}}' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1') \\ \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1', \mathsf{V}_{1\mathsf{Ck}}' \mid\!\uparrow c_1'$$

It follows by the value interpretation at type $\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}$ that the stepped configuration and the second configuration remain related for n'' (by $\forall$ elimination):

$$(\gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1', \mathsf{V}_{1\mathsf{Ck}}' \mid\!\uparrow c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

By D-restore-Jit, we have

$$[\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1', \mathsf{V}_{1\mathsf{Ck}}' \mid\!\uparrow c_1' \\ \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

From above, we get $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ and $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$. By the value interpretation at type $\uparrow \mathsf{C}_{\mathtt{unit}}$, these configurations continue to be related:

$$(\gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

as $\mathtt{restore}(\gamma_1', \mathsf{jit}, (\mathsf{N}_1', \mathsf{V}_{1\,\mathsf{Ck}}'), c_1') = \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$.

Since $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in k crashes we know that $[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in $k-1$ crashes.

By the application of the induction hypothesis we get: $[\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2' \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \mathsf{V}_2'' \mid \mathsf{skip}$, which combined with $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$ gives us the desired result of this subcase:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \mathsf{V}_2'' \mid \mathsf{skip}$$

With that established, we can apply the generalized statement on assumptions

$$(\gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c, \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$$

and $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ to get

$$[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2 \mid \mathsf{skip}$$

and apply D-P-Seq rule to complete the proof of this case:

$$\frac{\infty > 0 \qquad \begin{array}{c} \infty > 0 \\ [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2 \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N} \mid c; p' \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N}' \mid p'} \text{(D-P-SEQ)}$$

Stepping an atomic region. Consider a program of form $[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{Ckpt}[(\mathsf{aID}; \rho)](c_0); p'$ that can take a step using the D-P-Seq rule to $[\chi'' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p'$. By inversion on the D-P-Ckpt rule,

$$\frac{\begin{array}{c} n > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; \rho; \gamma) = \mathsf{N}_0, V_0 \\ [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{Ckpt}[(\mathsf{aID}; \rho)](c_0); p' \Rightarrow [\chi'' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p'} \text{(D-P-CKPT)}$$

we learn that

- $n > 0$
- $\mathsf{InitWorld}_d(\mathsf{N}; \rho; \gamma) = \mathsf{N}_0, V_0$

- $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$
- $n' > 0$
- $\mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}')$

Our goal is to simulate this execution in a continuous setting. In particular, we need to find a continuous execution such that $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid \mathsf{skip}$, where $\mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'_2; \mathsf{V}'_2)$. To this end, we invert the assumption $b : \mathtt{nat} \mid \Omega \Vdash \mathsf{Ckpt}[(\mathsf{aID}; \rho)](c_0); p' :\uparrow \mathsf{C}_{\mathtt{unit}}$ via P-Ckpt-semantic,

$$\frac{\begin{array}{c}\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho) \\ \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c_0 \leq c_0 : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}\end{array}}{b : \mathtt{nat} \mid \Omega \Vdash \mathsf{Ckpt}[\mathsf{aID}, \rho](c_0); p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-Ckpt-semantic)}$$

we learn that

(i) $\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho)$

(ii) $\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c_0 \leq c_0 : \mathsf{C}_{\mathtt{unit}}$

(iii) $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

By definition of logical relation applied to (ii), we have that $\forall n_1, m_1 \geq 0. \forall \gamma_1, \mathsf{N}_1, \mathsf{V}_1$ s.t. $\mathsf{N}_1 \mid \mathsf{V}_1 \Vdash \gamma_1 :: \Omega \mid \Sigma\ (\gamma_1 \mid \mathsf{aID}(c_0) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_0, \gamma_1 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_0) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_1}$.

By instantiating the memories accordingly, and the index m_1 with the number of tries $m + 1$ (where m is the number of crashes), we have

$$(\gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$$

To get our result, we first prove the following generalized statement: if

- $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID}(c_0) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n'_1 \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in m crashes and
- $(\gamma_1 \mid \mathsf{aID}(c_0) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$

then for all energy streams χ^0, we have $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}''_2 \mid \mathsf{V}''_2 \mid \mathsf{skip}$, where $\mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') = \mathsf{FinWorld}_d(\mathsf{N}''_2; \mathsf{V}''_2)$

The proof proceeds by induction on the number of crashes:

Base case: $m = 0$ **(*# tries* = 1).** If $m = 0$, then it follows by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$,

(1) $\exists(\gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1)$ s.t. $\gamma_1 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1$ AND

(2) $\exists(\gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2)$ s.t. $\gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2$ AND

(3) $(\gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1, \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^1$

The number of crashes is 0, so $n' > 0$ and the first configuration steps to completion via D-step where $\mathsf{N}'_1 = \mathsf{N}'$ and $\mathsf{V}'_1 = \mathsf{V}'$ and $\gamma''_1 = \gamma'$:

$$[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid \mathsf{skip}$$

Applying D-Step to (2), we know that:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (3) and $n' > 0$, we get that the post steps are related by the value interpretation at type $\downarrow\uparrow \texttt{unit}$. This means that $c_2' = \mathsf{skip}$ and we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}$$

and

$$(\gamma_1'' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \mathsf{skip}, \gamma_2'' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^0$$

It then follows that

(1) $\texttt{Commit}(\gamma_1'' \mid \mathsf{aID}(c_0) \mid \mathsf{N}_1' \mid \mathsf{V}_1') = \gamma' \mid \mathsf{N}', \mathsf{V}'$ where $\gamma' \subseteq \gamma_1''$, $\mathsf{N}_1' = \mathsf{N}', \mathsf{N}_{0\texttt{Ck}}'$, $\mathsf{V}_1' = \mathsf{V}_0', \mathsf{V}'$, $dom(\mathsf{V}') = dom(\mathsf{N}_0')$, $range(\gamma') = dom(\mathsf{N}_1') \cup dom(\mathsf{V}')$.

(2) $\texttt{Commit}(\gamma_2'' \mid \mathsf{aID}(c_0) \mid \mathsf{N}_2' \mid \mathsf{V}_2') = \gamma^2 \mid \mathsf{N}^2, \mathsf{V}^2$ where $\gamma^2 \subseteq \gamma_2''$, $\mathsf{N}_2' = \mathsf{N}^2, \mathsf{N}_{0\texttt{Ck}}^2$, $\mathsf{V}_2' = \mathsf{V}_0^2, \mathsf{V}^2$, $dom(\mathsf{V}^2) = dom(\mathsf{N}_0^2)$, $range(\gamma^2) = dom(\mathsf{N}_2') \cup dom(\mathsf{V}^2)$.

(3) $(\gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}', \mathsf{V}' \mid \mathsf{skip}, \gamma^2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}^2, \mathsf{V}^2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^0$

It follows by the value interpretation at type $\uparrow \texttt{unit}$ that $\mathsf{N}', \mathsf{V}' = \mathsf{N}^2, \mathsf{V}^2$. We observe that by definition, $\mathsf{FinWorld}_d(\mathsf{N}_1'; \mathsf{V}_1') = \mathsf{N}', \mathsf{V}'$ and $\mathsf{FinWorld}_d(\mathsf{N}_2'; \mathsf{V}_2') = \mathsf{N}^2, \mathsf{V}^2$.

Therefore, we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}$$

where $\mathsf{FinWorld}_d(\mathsf{N}_1'; \mathsf{V}_1') = \mathsf{FinWorld}_d(\mathsf{N}_2'; \mathsf{V}_2')$, and the proof of this subcase is complete.

Inductive case: $m = k + 1 (\exists k)$ **(*# tries* $= k + 2$).** By the term interpretation at type $\mathsf{C}_{\texttt{unit}}$, we have

(i) $\exists \gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ s.t. $\gamma_1 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$

(ii) $\exists \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$ s.t. $\gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$.

(iii) $(\gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+1}$

From (i), we step the first configuration until it becomes a value. It follows by the rule D-step that

$$[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

Since there are $m > 0$ crashes, we know that $n_1' = 0$. By (ii), we step the second configuration via D-step:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (iii), we have

$$(\gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

By application of D-Crash, we have

$$\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid 0 \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1')\end{array}$$

By the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$, observe that the stepped configuration continues to be related to the second configuration:

$$\begin{array}{r}(\gamma_1' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k\end{array}$$

By D-S-aID, for $\gamma^1 \subseteq \gamma_1'$ such that $\mathit{range}(\gamma^1) = \mathit{dom}(\mathsf{N}_1')$, we have:

$$\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1') \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1')\end{array}$$

By the value interpretation at type $\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})$, and the definition of `Pw0ff` in the atomic case, we have $\mathtt{Pw0ff}(\gamma_1', \mathsf{aID}(c_0), \mathsf{N}_1', \mathsf{V}_1') = \gamma^1 \mid \emptyset$, and thus the stepped configuration continues to be related to the second configuration:

$$\begin{array}{r}(\gamma^1 \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k\end{array}$$

By stepping the first configuration according to D-charge, we have for some $n'' > 0$ such that $\chi = n'' :: \chi'$:

$$\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1') \\ \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid\!\uparrow c_1'\end{array}$$

By the value interpretation at type $\mathtt{nat} \rightsquigarrow\!\uparrow \mathsf{C}_{\mathtt{unit}}$, observe that the stepped configuration remains related to the second configuration for $n'' > 0$:

$$(\gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid\!\uparrow c_1', \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

Stepping the first configuration via D-restore-aID, we have

$$[\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid\!\uparrow c_1' \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid \cdot \mid c_0$$

By the value interpretation at type $\uparrow\ \mathsf{C}_{\mathtt{unit}}$, the first and second configurations remain related:

$$(\gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid \cdot \mid c_0, \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^k$$

since $\texttt{restore}(\gamma^1, \mathsf{aID}(c_0), \mathsf{N}'_1, c'_1) = \mathsf{N}'_1 \mid \cdot \mid c_0$.

By assumption, $[\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}'_1 \mid \cdot \mid c_0 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in $k-1$ crashes.

By induction hypothesis, we get $[\chi^0 \rhd \varepsilon] \otimes \gamma'_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}''_2 \mid \mathsf{V}''_2 \mid \mathsf{skip}$ such that $\mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') = \mathsf{FinWorld}_d(\mathsf{N}''_2; \mathsf{V}''_2)$. This combined with $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma'_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2$ gives us $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}''_2 \mid \mathsf{V}''_2 \mid \mathsf{skip}$, and completes the proof of this subcase.

With that established, we can apply the generalized statement on assumptions

$$(\gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{m+1}$$

and $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ to get $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma'' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'' \mid \mathsf{V}'' \mid \mathsf{skip}$, where $\mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') = \mathsf{FinWorld}_d(\mathsf{N}''; \mathsf{V}'')$. and apply D-P-Ckpt rule to complete the proof of this case.

□

Appendix B

More Efficient Checkpointing Policies Appendix

B.1 Syntax and Operational Semantics

B.1.1 Syntax

We present the syntax for our calculus below, which extends the syntax from Appendix A with supports for must-first-write variables. These include extending atomic region policies with a programmer-defined set of must-first-write variables μ and qualifiers `MFstWt` and `Wtn` that are assigned to these variables. In particular, variables in μ are initially assigned a `MFstWt` qualifier which evolves to `Wtn` on its first write for a given atomic region (re-)execution.

Commands and expressions

values	v	::=	$\mathsf{n} \mid \mathsf{tt} \mid \mathsf{ff} \mid x$
exprs	e	::=	$v \mid e \odot e$
cmds	c	::=	$\mathsf{skip} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \mid \mathsf{if}\, e\, \mathsf{then}\, c\, \mathsf{else}\, c \mid x := e \mid c; c \mid c;_W c$
atom. reg. pols.	$aPol$	::=	(ρ, μ, ω)
progs	p	::=	$\mathsf{skip} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}; p \mid c; p$

Access qualifiers and memories

access qualifier	q	::=	$\mathtt{CK} \mid \mathtt{RD} \mid \mathtt{MFstWt} \mid \mathtt{Wtn}$
nonvolatile mem	N	::=	$\cdot \mid \ell \mathbin{@} q \hookrightarrow v, \mathsf{N} \mid \ell_{\mathtt{ck}} \mathbin{@} \mathtt{CK} \hookrightarrow v, \mathsf{N}$
volatile mem	V	::=	$\cdot \mid \ell \mathbin{@} \mathtt{CK} \hookrightarrow v, \mathsf{V}$
var loc map	γ	::=	$\cdot \mid \gamma, x \mapsto \ell$

Instructions, statements, and configurations

continuations	κ	::=	$c \mid e$
statements	s	::=	$\kappa \mid i \mid p$
crash instrs	i	::=	$\downarrow\varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa) \mid \varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa) \mid\uparrow \kappa$
energy level	g	::=	$\cdot \mid n$
charge stream	χ	::=	$n :: \chi$
open config	C_o	::=	$(\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{V} \mid s) \mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid s)$
closed config	C_c	::=	$[\chi \rhd \varepsilon] \otimes C_o$
exec. mode	$\mathtt{Md}$	::=	$\mathsf{aID}(c) \mid \mathsf{jit}$

B.1.2 Access Transition Function (δ)

We define a helper function δ to aid in checking memory accesses and evolving access qualifiers. The function indicates that reading and writing to checkpointed (CK) variables is allowed and results in the same qualifier. Writing to a must-first-write (MFstWt) variable evolves the qualifier to Wtn, whereas reading from it is undefined, indicating a potential WAR variable that is not checkpointed. Reading from a read-only (RD) variable results in the same qualifier, whereas written on a read-only variable is undefined, pointing to another WAR variable that is not checkpointed. Lastly, writing and reading an already written variable (Wtn) results in the same qualifier.

$$\delta(\texttt{CK}, Rd) = \delta(\texttt{CK}, Wt) = \texttt{CK} \qquad \delta(\texttt{MFstWt}, Wt) = \texttt{Wtn} \qquad \delta(\texttt{MFstWt}, Rd) = UN$$
$$\delta(\texttt{RD}, Rd) = \texttt{RD} \qquad \delta(\texttt{RD}, Wt) = UN \qquad \delta(\texttt{Wtn}, Wt) = \delta(\texttt{Wtn}, Rd) = \texttt{Wtn}$$

B.1.3 Value Configurations and Expression Execution

The value relations remain the same as before. We repeat them below.

$$\frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip})}\ \text{(V-SKIP)} \qquad \frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{n})}\ \text{(V-NAT)}$$

$$\frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{tt})}\ \text{(V-BOOL-T)} \qquad \frac{n > 0}{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{ff})}\ \text{(V-BOOL-F)}$$

$$\frac{}{Val(\gamma \mid \texttt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid \kappa)}\ \text{(V-CRASH)}$$

$$\frac{}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \texttt{Md} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \downarrow\varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa))}\ \text{(V-}\downarrow\text{)}$$

$$\frac{}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \texttt{Md} \mid \cdot \mid \mathsf{N} \mid \varepsilon \mathbin{\#} \mathsf{in}(b > 0, \uparrow\kappa))}\ \text{(V-\# in)}$$

$$\frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \uparrow\kappa)}\ \text{(V-}\uparrow\text{)} \qquad \frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{skip})}\ \text{(V-P-DONE)}$$

$$\frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip})}\ \text{(V-C-DONE)}$$

B.1.4 Expression Execution

The operational semantic rules for evaluating binary expressions are also the same as before, except for D-N-READ for reading a nonvolatile memory location is extended with a check that the location is readable ($\delta(q, Rd) \neq \texttt{UN}$).

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell @ q \hookrightarrow v, \mathsf{V}' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-V-READ)}$$

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \delta(q, Rd) \neq \mathtt{UN} \qquad \mathsf{N} = \ell @ q \hookrightarrow v, \mathsf{N}' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-N-READ)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1' \odot e_2} \text{ (D-BINARY-1)}$$

$$\frac{\begin{array}{c} n > 0 \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2' \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2'} \text{ (D-BINARY-2)}$$

$$\frac{\begin{array}{c} n = n' + 1 \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2) \qquad v = e_1 \odot e_2 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-BINARY-V)}$$

B.1.5 Command Execution (Open Config)

The rules for command execution of open configurations remain the same as before, except for assignment to a nonvolatile variable. The rule D-Assign-N is extended with a check that the location corresponding to the variable x is in fact writable according to δ, and if it is, updates the qualifier accordingly.

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e'\, \mathsf{in}\, c} \text{ (D-Let-step)}$$

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\ \mathit{fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{let}\, x = e_1\, \mathsf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{CK} \hookrightarrow e_1 \mid c} \text{ (D-Let-v)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid p := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid p := e'} \text{ (D-Assign-step)}$$

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q \hookrightarrow e \mid \mathsf{skip}} \text{ (D-Assign-V)}$$

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \qquad q' = \delta(q, \mathit{Wt}) \neq \mathtt{UN} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathsf{skip}} \text{ (D-Assign-N)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, e'\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2} \text{ (D-If)}$$

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, \mathsf{tt}\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \text{ (D-If-tt)}$$

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{if}\, \mathsf{ff}\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2} \text{ (D-If-ff)}$$

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2} \text{ (D-seq)}$$

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{ (D-seq-step)}$$

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction \mathsf{dom}(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{skip};_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2} \text{ (D-seq-v)}$$

Definition 11 (Well-formedness for configurations) *A configuration* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed iff when* $c = c_1;_{W_1} \cdots ;_{W_{n-1}} c_n;_{W_n} c_{n+1}$ *where* $n > 1$*, all of the following hold*

- $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}_j) \subseteq dom(\mathsf{V}_i) \subseteq dom(\mathsf{V})$
- $\gamma_j \subseteq \gamma_i \subseteq \gamma$

where $1 \leq i < j \leq n$ *and* $W_k = \gamma_k \mid \mathsf{V}_k$ *for all* $k \in [1, n]$.

B.1.6 Command Execution (Closed Config)

Since only the restore operation must be modified to reset the `Wtn` qualifiers to `MFstWt` prior to each atomic region re-execution, the only closed configuration command execution rule that changes is D-RESTORE-AID. To restore state, as detected by the instruction $\uparrow\kappa$, the rule replaces all `Wtn` qualifiers in nonvolatile memory with `MFstWt` and regenerates the volatile log for checkpointed variables.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'} \text{ (D-STEP)}$$

$$\frac{}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid \kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa)} \text{ (D-CRASH)}$$

$$\frac{}{\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \mathsf{N}, \mathsf{V}_{\mathsf{ck}} \mid \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa)\end{array}} \text{ (D-S-JIT)}$$

$$\frac{\gamma' \subseteq \gamma \qquad range(\gamma') = dom(\mathsf{NV})}{\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N} \mid \mathsf{V} \mid \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N} \mid \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa)\end{array}} \text{ (D-S-AID)}$$

$$\frac{}{[n :: \chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{N} \mid \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \uparrow\kappa} \text{ (D-CHARGE)}$$

$$\frac{\mathsf{N} = \mathsf{N}'_{\mathtt{RD},\mathtt{MFstWt}}, \mathsf{N}''_{\mathsf{ck}}, \mathsf{N}'''_{\mathtt{Wtn}}}{\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid n \mid \mathsf{N} \mid \uparrow\kappa \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID} \mid n \mid \mathsf{N}'_{\mathtt{RD},\mathtt{MFstWt}}, \mathsf{N}''_{\mathsf{ck}}, \mathsf{N}'''_{\mathtt{MFstWt}} \mid \mathsf{N}'' \mid c_0\end{array}} \text{ (D-RESTORE-AID)}$$

$$\frac{\mathsf{N} = \mathsf{N}', \mathsf{N}''_{\mathsf{ck}}}{[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \uparrow\kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N}' \mid \mathsf{N}'' \mid \kappa} \text{ (D-RESTORE-JIT)}$$

B.1.7 Top-Level Program Execution

The top-level program execution rules are given below. The rules for JIT and atomic regions remain the same as before, with a modified definition of $\mathsf{InitWorld}_d$ and $\mathsf{FinWorld}_d$. From before, $\mathsf{InitWorld}_d$ is additionally defined to set the qualifiers of all variables in the set μ to `MFstWt`.

By the successful end of an atomic region execution, all `MFstWt` qualifiers will become `Wtn`. Therefore, the $\mathsf{FinWorld}_d$ function replaces the `Wtn` qualifiers with `CK` and commits the volatile logs to nonvolatile memory.

$$\frac{\begin{array}{c} n > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{V}_0 \\ [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\varepsilon : l'] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p \;\Rightarrow_p\; [\varepsilon : l'] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p} \text{ (D-P-CKPT)}$$

$$\frac{\begin{array}{c} n > 0 \\ n' > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid c; p \;\Rightarrow_p\; [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p} \text{ (D-P-SEQ)}$$

Definitions of $\mathsf{InitWorld}_d$ and $\mathsf{FinWorld}_d$

Definition 12 $\mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{V}_0 \mid \gamma_0$ *where* $aPol = (\rho, \mu, \omega)$ *iff*

- $\mathsf{N} = \mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_3$,
- $\mathsf{dom}(\mathsf{N}_1) = \mathsf{range}(\gamma\restriction\rho)$,
- $\mathsf{dom}(\mathsf{N}_2) = \mathsf{range}(\gamma\restriction\omega)$,
- $\mathsf{dom}(\mathsf{N}_3) = \mathsf{range}(\gamma\restriction\mu)$,
- $\mathsf{V}_0, \gamma_v = \mathtt{unzip}(\{(\ell^*@\mathtt{CK} \hookrightarrow v,\ x \mapsto \ell^*) \mid \mathsf{N}_2(\ell) = v,\ \ell^* \textit{ fresh}, x \in \omega \textit{ s.t. } \gamma(x) = \ell\})$,
- $\mathsf{N}_0 = \{\ell@\mathtt{RD} \hookrightarrow v \mid \mathsf{N}_1(\ell) = v\} \cup \{\ell_{\mathtt{ck}}@\mathtt{CK} \hookrightarrow v \mid \mathsf{N}_2(\ell) = v\} \cup \{\ell@\mathtt{MFstWt} \hookrightarrow v \mid \mathsf{N}_3(\ell) = v\}$,
- $\gamma_0 = \{x_{\mathtt{ck}} \mapsto \gamma(x)_{\mathtt{ck}} \mid x \in \omega\} \cup \{x \mapsto \gamma(x) \mid x \notin \omega\} \cup \gamma_v$

Definition 13 $\mathsf{FinWorld}_d(\mathsf{N}; \mathsf{V}) = \mathsf{N}_f$ *iff*

- $\mathsf{N} = \mathsf{N}_{1\,\mathtt{ck}}, \mathsf{N}_2$,
- $\mathsf{dom}(\mathsf{N}_1) = \mathsf{dom}(\mathsf{V})$,
- $\mathsf{N}_f = \mathsf{V} \cup \{\ell@\mathtt{CK} \hookrightarrow v \mid \ell@q \hookrightarrow v \in \mathsf{N}_2, q \in \{\mathtt{RD}, \mathtt{Wtn}\}\}$

B.2 Type system

B.2.1 Types

We present the grammar for types below. The structure of crash types and typing contexts remains the same as before, but are extended with access qualifiers `MFstWt` and `Wtn` for type checking atomic regions.

$$
\begin{array}{lllll}
\textit{store types} & A & ::= & \texttt{int} \mid \texttt{bool} \\
\textit{basic types} & T & ::= & \texttt{unit} \mid A \\
\textit{unstable types} & \tau^u & ::= & T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid \mathsf{C}_T^{\texttt{Md}} \\
\textit{stable types} & \tau^s & ::= & \texttt{nat} \rightsquigarrow \tau^s \mid \uparrow \tau^u \\
\textit{type variables} & \mathsf{C}_T^{\texttt{Md}} & ::= & \mathsf{C}_{\texttt{unit}} \mid \mathsf{C}_A^{\texttt{Md}} \\
\textit{access qualifier} & q & ::= & \texttt{CK} \mid \texttt{RD} \mid \texttt{MFstWt} \mid \texttt{Wtn} \\
\textit{unstable store typing context} & \Sigma & ::= & \cdot \mid x : \downarrow_u^s \uparrow_u^s A @ \texttt{CK}, \Sigma \\
\textit{stable store typing context} & \Omega & := & \cdot \mid x : \uparrow_u^s A @ q,\ \Omega \mid x_{\texttt{ck}} : \uparrow_u^s A @ \texttt{CK}, \Omega
\end{array}
$$

$$
\begin{array}{rl}
\textit{command crash type} & \mathsf{C}_{\texttt{unit}} = \downarrow_u^s (\texttt{nat} \rightsquigarrow \uparrow_u^s \mathsf{C}_{\texttt{unit}}) \vee \downarrow_u^s \uparrow_u^s \texttt{unit} \\
\textit{atomic crash type} & \mathsf{C}_A^{\texttt{Md}} = \downarrow_u^s (\texttt{nat} \rightsquigarrow \uparrow_u^s \mathsf{C}_{\texttt{unit}}) \vee \downarrow_u^s \uparrow_u^s A \\
\textit{jit crash type} & \mathsf{C}_A^{\texttt{jit}} = \downarrow_u^s (\texttt{nat} \rightsquigarrow \uparrow_u^s \mathsf{C}_A^{\texttt{jit}}) \vee \downarrow_u^s \uparrow_u^s A
\end{array}
$$

B.2.2 Expression Typing

Since `MFstWt` qualifiers could evolve to `Wtn` over the course of type checking, `WT` expression typing judgments are extended with a stable post-context whose domain matches the initial stable typing context, but may differ in its qualifiers to track changes in qualifiers.

For example, to check a variable that is being written, the rule T-Loc-Write checks that the variable is writable according to δ. If the variable is writable, the rule assigns the updated qualifier to the typing of x. The rule T-W-Shift, restricts the post-typing context from premise to conclusion to match the domain of the initial stable typing context. The remainder of the expression typing rules are all for the `RD` judgments. Since qualifiers do not change on read, These rules remain the same as before.

$$\frac{\Omega, \Sigma' = x{:}\uparrow A@q, \Omega_2' \qquad q' = \delta(q, Wt) \neq UN}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega, \Sigma' \vdash_{\texttt{WT}} x : \uparrow A \dashv x{:}\uparrow A@q', \Omega_2'} \text{ (T-Loc-Write)}$$

$$\frac{\Sigma =\downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\text{ck}} \qquad \texttt{Md} \mid b : \texttt{nat} \mid \Omega', \Sigma' \vdash_{\texttt{WT}} x : \uparrow A \dashv \Omega_1}{\texttt{Md} \mid \ b : \texttt{nat} \ \mid \ \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega_1 \restriction dom(\Omega)} \text{ (T-W-Shift)}$$

$$\frac{\Omega(x) = \uparrow A@q}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega \vdash_{\texttt{RD}} x : \uparrow A} \text{ (T-Loc-Read)} \qquad \frac{}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega \vdash_{\texttt{RD}} \texttt{tt} :\uparrow \texttt{bool}} \text{ (T-Bool-t)}$$

$$\frac{}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega \vdash_{\texttt{RD}} \texttt{ff} : \uparrow\texttt{bool}} \text{ (T-Bool-f)} \qquad \frac{}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega \vdash_{\texttt{RD}} \texttt{n} : \uparrow\texttt{int}} \text{ (T-Int)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\text{ck}} \qquad \texttt{Md} \mid b : \texttt{nat} \mid \Omega', \Sigma' \vdash_{\texttt{RD}} v : \uparrow A}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} v : \downarrow\uparrow A} \text{ (T-R-Shift)}$$

$$\frac{\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} x : \downarrow\uparrow A}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} x : \tau_1 \vee \downarrow\uparrow A} \text{ (T-}\vee\text{-Succ)}$$

$$\frac{\begin{array}{c}\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e_1 : \texttt{C}_T^{\texttt{Md}} \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e_2 : \texttt{C}_{T'}^{\texttt{Md}} \qquad \odot : \uparrow T \times \uparrow T' \to \uparrow T''\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e_1 \odot e_2 : \texttt{C}_{T''}^{\texttt{Md}}} \text{ (T-Binary)}$$

$$\frac{\begin{array}{c}\texttt{Sig}' = \{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD}} e : \tau\} \\ \texttt{Sig}'' = \textit{if } \texttt{Md} = \texttt{jit}, \textit{ then } \texttt{Sig}', \textit{else } \texttt{Sig} \\ \texttt{Md} \mid b = 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}''} e : \tau \qquad \texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \tau\end{array}}{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \tau} \text{ (T-enough?)}$$

B.2.3 Command Typing

Checking variable writes propagates to command typing in the form of checking assignments, which propagates through sequencing to account for updated variable accesses in checking later commands. In particular, T-Assign returns a post-context that matches its WT judgment to reflect the write to x that occurs during the assignment. The sequencing rules (T-seq and T-seq-d) pass along information about qualifier accesses from checking a command c_1 to checking the next command c_2. The typing rule for checking conditionals (T-If) assumes that checking c_1 and c_2 result in the same post-typing context, rejecting programs that write to a MFstWt variable in one branch but not the other. T-C-Shift is similar to T-W-Shift, restricting the post-context of the conclusion from the premise. T-Skip returns the same post-context as the initial stable typing context. The rules T-$\vee$-Succ and T-Let pass the same post-context from premise to conclusion. Lastly, T-enough? populates the type signature with the post-context Ω' to remember that a typing derivation that already solved for Ω' already exists.

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega} \text{ (T-Skip)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathtt{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \lceil dom(\Omega)} \text{ (T-C-Shift)}$$

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \vee \downarrow\uparrow\mathtt{unit} \dashv \Omega'} \text{ (T-}\vee\text{-Succ)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}\downarrow\uparrow A @ \mathtt{CK} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{let}\, x = e_1 \,\mathsf{in}\, c : \tau \dashv \Omega'} \text{ (T-Let)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'} \text{ (T-Assign)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_{\mathtt{bool}}^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{if}\, e \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2 : \tau \dashv \Omega'} \text{ (T-If)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''} \text{ (T-seq)}$$

$$\frac{\begin{array}{c}W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega' \\ \Sigma' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''} \text{ (T-seq-d)}$$

$$\frac{\begin{array}{c}\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau \dashv \Omega'\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'} \text{ (T-enough?)}$$

B.2.4 Statement Typing

The rules for statement typing remain similar to before, excpet for T-aID-Restore which is extended to revert all `Wtn` qualifiers to `MFstWt` which matches the dynamic rule.

$$\frac{\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T'})}{\mathtt{Md} \mid b \ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \kappa' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-}\vee\text{-Crash)}$$

$$\frac{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}})}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}})} \text{ (T-aID-stop)}$$

$$\frac{\Sigma = \downarrow\uparrow\Sigma' \qquad \mathsf{jit} \mid \cdot \mid \Omega, \uparrow\Sigma'_{\mathsf{ck}} \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})} \text{ (T-Jit-stop)}$$

$$\frac{\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathtt{Md} \mid \ b \ > 0 : \mathtt{nat} \mid \ \Omega \vdash_{\mathtt{Sig}} \uparrow \kappa' : \uparrow \mathsf{C}^{\mathtt{Md}}_{T}}{\mathtt{Md} \mid \ \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow \kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})} \text{ (T-Charge)}$$

$$\frac{\begin{array}{c}\Omega = \Omega'_{\mathtt{RD},\mathtt{MFstWt},\mathtt{Wtn}}, \Omega''_{\mathsf{ck}} \\ \Omega_1 = \Omega'_{\mathtt{RD},\mathtt{Wtn},\mathtt{Wtn}}, \Omega''_{\mathsf{ck}} \qquad \mathsf{aID}(c_0) \mid \ b \ \geq 0 : \mathtt{nat} \mid \ \Omega_1; \downarrow\Omega'' \vdash c_0 \ : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig}\end{array}}{\mathsf{aID}(c_0) \mid \ b \ > 0 : \mathtt{nat} \mid \ \Omega \vdash_{\mathtt{Sig}} \uparrow\kappa' :\uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-aID-Restore)}$$

$$\frac{\Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathsf{jit} \mid \ b \ \geq 0 : \mathtt{nat} \mid \ \Omega'; \downarrow\Omega'' \vdash \kappa' \ : \mathsf{C}^{\mathtt{Md}}_{T} \in \mathtt{Sig}}{\mathsf{jit} \mid \ b \ > 0 : \mathtt{nat} \mid \ \Omega \vdash_{\mathtt{Sig}} \uparrow\kappa' :\uparrow \mathsf{C}^{\mathtt{Md}}_{T}} \text{ (T-Jit-Restore)}$$

B.2.5 Program Typing

The rule for checking an atomic region is extended from before. The modified $\mathsf{InitWorld}_t$ definition checks that the variable sets in *aPol* are all pairwise disjoint and cover the domain of nonvolatile memory. For every variable declared in μ, it sets the qualifier to `MFstWt`. At the end of checking an atomic region, all `MFstWt` variables should become `Wtn`. This is checked by $\Omega' \restriction \{\mathtt{MFstWt}\} = \emptyset$, where Ω' is the post-context from checking the atomic region command c_0.

$$\frac{\begin{array}{c}\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; \mathit{aPol}) \\ \mathtt{Sig} = \{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'\} \\ \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \ \Sigma_0 \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \\ \Omega' \restriction \{\mathtt{MFstWt}\} = \emptyset \qquad b \ : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}\end{array}}{b : \mathtt{nat} \mid \Omega \vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \mathit{aPol}); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-Ckpt)}$$

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash c; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-seq)}$$

Definition 14 $\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; \mathit{aPol})$ *where* $\mathit{aPol} = (\rho, \mu, \omega)$ *iff*

- $\mathsf{dom}(\Omega) = \rho \cup \mu \cup \omega$,
- $\rho \cap \mu = \emptyset, \mu \cap \omega = \emptyset, \rho \cap \omega = \emptyset$,

- $\Omega_0 = \Omega^r, \Omega^m, \Omega^c_{\mathsf{ck}}$ *and*
- $\Sigma_0 = \downarrow\Omega^c$

where $\Omega = \Omega^r, \Omega^m, \Omega^c$ *and* $\Omega^r = \Omega\restriction\rho, \Omega^m = \Omega\restriction\mu$, *and* $\Omega^c = \Omega\restriction\omega$.

B.2.6 Store Typing

The store typing definition remains the same as before. We repeat the definition below.

$$\frac{}{\vdash^{\mathsf{aID}}_{\gamma} \cdot \mid \cdot : \cdot \mid \cdot} \text{(Empty)}$$

$$\frac{\mathsf{V} = \mathsf{V}', \ell \mathbin{@} q \hookrightarrow v \qquad q = \mathtt{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A \qquad \vdash^{\mathsf{jit}}_{\gamma'} \mathsf{N} \mid \mathsf{V}' : \Omega \mid \Sigma}{\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma, (x{:}\downarrow\uparrow A @ q)} \text{(V-loc)}$$

$$\frac{q \neq \mathtt{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A \qquad \mathsf{N} = \mathsf{N}', \ell \mathbin{@} q \hookrightarrow v \qquad \vdash^{\mathsf{aID}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} : \Omega \mid \Sigma}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, (x{:}\uparrow A @ q) \mid \Sigma} \text{(N-loc-aID-1)}$$

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v' : \uparrow A \qquad q = \mathtt{Ck} \qquad \mathsf{V} = \mathsf{V}', \ell \mathbin{@} q \hookrightarrow v' \qquad \mathsf{N} = \mathsf{N}', \ell_{\mathsf{ck}} \mathbin{@} q \hookrightarrow v \qquad \vdash^{\mathsf{aID}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, (x_{\mathsf{ck}}{:}\uparrow A @ q) \mid \Sigma, (x{:}\downarrow\uparrow A @ q)} \text{(N-loc-aID-2)}$$

$$\frac{q = \mathtt{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A \qquad \mathsf{N} = \mathsf{N}', \ell \mathbin{@} q \hookrightarrow v \qquad \vdash^{\mathsf{jit}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} : \Omega \mid \Sigma}{\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, (x{:}\uparrow A @ q) \mid \Sigma} \text{(N-loc-Jit)}$$

B.3 Logical Relation

The logical relation is the same as before, but includes a condition that makes it more general. We state the definition below.

The key difference is that now there may be variables written on the intermittent execution that are not written on the continuously-powered execution at intermediary stages of the simulation. After restore, the qualifiers of these variables are reverted to `MFstWt`. This check is included in the definition of $\mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^m$.

Binary relation for commands

$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega \mid \Sigma \Vdash c_1 \leq c_2 : \mathsf{C}_{\mathtt{unit}}$
iff $\forall n, m \geq 0.\ \forall \gamma, \mathsf{N}, \mathsf{V}. s.t.\ \vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma.$
$(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1, \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m$

Term relation

$$\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1} = \{(C_{o1}, C^{\infty}_{o2}) \mid \exists C'_{o1}\ s.t. C_{o1} \rightarrow^*_{irred} C'_{o1} \wedge \exists C^{\infty'}_{o2}\ s.t. C^{\infty}_{o2} \rightarrow^* C^{\infty'}_{o2} \wedge (C'_{o1}, C^{\infty'}_{o2}) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}\}$$

$$\mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{0} = \{(C_{o1}, C^{\infty}_{o2})\}$$

Value relation

$$\mathcal{V}[\![\uparrow \mathtt{unit}]\!] = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma \mid \mathtt{Md} \mid n_1 \mid \mathsf{N} \mid \mathsf{skip}) \wedge C^{\infty}_{o2} = (\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N} \mid \mathsf{skip})\}$$

$$\mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!] = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \mathsf{skip}) \wedge C^{\infty}_{o2} = (\gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid \mathsf{skip}) \wedge \mathtt{Commit}(\gamma_i, \mathtt{Md}, \mathsf{N}_i, \mathsf{V}_i) = \gamma'_i \mid \mathsf{N}'_i \wedge (\gamma'_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}'_1 \mid \mathsf{skip}, \gamma'_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \mathtt{unit}]\!]\}$$

$$\mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^m = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \uparrow \kappa) \wedge C^{\infty}_{o2} = (\gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \wedge \mathtt{Restore}(\gamma_1, \mathtt{Md}, \mathsf{N}_1, \kappa) = \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \wedge \mathsf{N}_0 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\} \wedge (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma_2 \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^m\}$$

$$\mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^m = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa)) \wedge \forall n{>}0. (\gamma_1 \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \uparrow \kappa, C^{\infty}_{o2}) \in \mathcal{V}[\![\uparrow\ \mathsf{C}_{\mathtt{unit}}]\!]^m\}$$

$$\mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^m = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa)) \wedge \mathtt{PwOff}(\gamma_1, \mathtt{Md}, \mathsf{N}_1, \mathsf{V}_1) = \gamma'_1 \mid \mathsf{V}' \wedge (\gamma'_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{V}'_{\mathsf{ck}}, \mathsf{N}_1 \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa), C^{\infty}_{o2}) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^m\}$$

$$\mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1} = \{(C_{o1}, C^{\infty}_{o2}) \mid C_{o1} = (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1) \wedge \textit{either}\ n_1 = 0 \wedge (\gamma_1 \mid \mathtt{Md} \mid \cdot \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow c_1), C^{\infty}_{o2}) \in \mathcal{V}[\![\downarrow(\mathtt{nat} \rightsquigarrow \uparrow\ \mathsf{C}_{\mathtt{unit}})]\!]^m, \textit{or}\quad n_1 > 0 \wedge (\gamma_1 \mid \mathtt{Md} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, C^{\infty}_{o2}) \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]\}$$

Figure B.1: Step-indexed logical relation for crash types

B.3.1 `Commit`, `PwOff`, and `Restore` Policies

The policies for `Commit` and `PwOff` are the same as before, whereas `Restore` is modified. Like the typing dynamic rules for restore, the policy reverts all `Wtn` qualifiers to `MFstWt`.

$$\frac{\mathsf{N}_2 = \mathsf{N}'_{1\,\mathtt{ck}}, \mathsf{V}'' \qquad \mathsf{V}_1 = \mathsf{V}'_1, \mathsf{V}'' \qquad \mathsf{N}_1 = \mathsf{N}'_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}''_{\mathtt{ck}} \qquad \mathsf{dom}(\mathsf{V}'') = \mathsf{dom}(\mathsf{N}'') \qquad \gamma' \subseteq \gamma \; s.t. \; \mathit{range}(\gamma') = \mathsf{dom}(\mathsf{N}_1)}{\mathtt{Commit}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{N}_2}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{ck}}, \mathsf{N}^3_{\mathtt{Wtn}} \qquad \mathsf{N}_2 = \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} \qquad \mathsf{V}_2 = \mathsf{N}''}{\mathtt{Restore}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \kappa) = \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_0}$$

$$\frac{\gamma' \subseteq \gamma \; s.t. \; \mathit{range}(\gamma') = \mathsf{dom}(\mathsf{N}_1) \qquad \mathsf{V}_2 = \emptyset}{\mathtt{PwOff}(\gamma, \mathsf{aID}(c_0), \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2} \qquad \frac{\gamma' \subseteq \gamma \; s.t. \; \mathsf{range}(\gamma') = \mathsf{dom}(\mathsf{N})}{\mathtt{Commit}(\gamma, \mathsf{jit}, \mathsf{N}, \mathsf{V}_1) = \gamma' \mid \mathsf{N}}$$

$$\frac{\mathsf{N}_1 = \mathsf{N}', \mathsf{N}''_{\mathtt{ck}} \qquad \mathsf{N}_2 = \mathsf{N}' \qquad \mathsf{V}_2 = \mathsf{N}''}{\mathtt{Restore}(\gamma, \mathsf{jit}, \mathsf{N}_1, \kappa) = \mathsf{N}_2 \mid \mathsf{V}_2 \mid c} \qquad \frac{\mathsf{V}_2 = \mathsf{V}_1 \qquad \gamma' = \gamma}{\mathtt{PwOff}(\gamma, \mathsf{jit}, \mathsf{N}_1, \mathsf{V}_1) = \gamma' \mid \mathsf{V}_2}$$

B.3.2 Semantic Typing

The rules for semantic typing remain the same as before. We repeat them below.

$$\frac{\Omega_0 \mid \Sigma_0 = \mathsf{InitWorld}_t(\Omega; \mathit{aPol}) \qquad \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0 \mid \Sigma_0 \Vdash c_0 \leq c_0 : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \mathit{aPol}); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-CKPT-SEMANTIC)}$$

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega \mid \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash c; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-SEQ-SEMANTIC)}$$

B.4 Progress and Preservation

Lemma 2 (Progress for shifted expressions) *If*

$$\mathtt{Md} \mid b{:}\mathtt{nat} \mid \Omega \vdash_{Rd} e : \uparrow A$$

then $\forall\ n : \mathtt{nat}$ *with* $n > 0$ *and* $\forall\, \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Proof: The proof proceeds by structural induction over $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{Rd} e : \uparrow A$.

Case 1 [T-Loc-Read]**.** Suppose the last rule in the typing derivation is T-Loc-Read:

$$\frac{\Omega = x : \uparrow A @ q, \Omega'}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} x : \uparrow A}\ \text{(T-Loc-Read)}$$

Then by inversion, we have $\Omega = x : \uparrow A @ q, \Omega'$. By assumption, $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$. By inversion of $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$ according to the well-formedness definition, one of the two subcases hold:

Subcase 1. $\mathsf{N} = \ell @ q \hookrightarrow v, \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-loc-Read applies:

$$\frac{\begin{array}{c}\gamma = \gamma', [x \mapsto \ell] \\ \mathsf{N} = \ell @ q \hookrightarrow v, \mathsf{N}' \qquad \delta(q, \mathtt{RD}) \neq \mathtt{UN} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v}\ \text{(D-loc-Read)}$$

This yields the desired result.

Subcase 2. $\mathsf{V} = \ell @ q \hookrightarrow v, \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-loc-Read applies:

$$\frac{\begin{array}{c}\gamma = \gamma', [x \mapsto \ell] \\ \mathsf{V} = \ell @ q \hookrightarrow v, \mathsf{V}' \qquad \delta(q, \mathtt{RD}) \neq \mathtt{UN} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v}\ \text{(D-var-Read)}$$

This yields exactly the desired result.

Case 2 [T-Bool-t]**.** Suppose that the last rule in the typing derivation is T-Bool-t:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} \mathbf{tt} :\uparrow \mathit{bool}}\ (\text{T-Bool-t})$$

By the value rule V-Bool-t and the assumption $n > 0$ we get

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{tt})}\ (\text{V-Bool-t})$$

Case 3 [T-Bool-f]**.** Suppose that the last rule in the typing derivation is T-Bool-f:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} \mathbf{ff} :\uparrow \mathit{bool}}\ (\text{T-Bool-f})$$

By the value rule V-Bool-f and the assumption $n > 0$ we get

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{ff})}\ (\text{V-Bool-f})$$

Case 4 [T-Int]**.**

Suppose that the last rule in the typing derivation is T-Int:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} \mathbf{n} : \uparrow \mathit{int}}\ (\text{T-Int})$$

By the value rule V-Int and the assumption $n > 0$ we get:

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{n})}\ (\text{V-Int})$$

□

Theorem 11 (Progress for expressions) *If* $\mathtt{Md} \mid b\ \mathcal{R}\ m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Rd;\mathtt{Sig}} e : \tau$*, then* $\forall\ n : \mathit{nat}$ *with* $n\mathcal{R}m$ *and* $\forall\ \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*, either*

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e\ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b\ \mathcal{R}\ m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Rd;\mathtt{Sig}} e : \tau$. We consider a specific (co-)natural number $n\mathcal{R}m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau}\ \text{(T-enough?)}$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n (i.e. $n = 0$ or $n > 0$):

Subcase 1 [$n = 0$]**.** By the value rule V-e-Crash, we have

$$\mathit{Val}(\gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid e),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, we have

(1) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau$

(3) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\}$

(4) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis to (2) to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

which completes the proof of this subcase.

Case 2 [T-Binary]**.**

Suppose the last rule in the typing derivation is T-Binary and $e = e_1 \odot e_2$:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_T^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}} \qquad \odot : \uparrow T \times \uparrow T' \rightarrow \uparrow T''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}_{T''}^{\mathtt{Md}}}\ \text{(T-Binary)}$$

By assumption, we know that $n > 0$. By inversion, we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_T^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}}$

(3) $\odot : \uparrow T \times \uparrow T' \rightarrow \uparrow T''$

By the inductive hypothesis applied to (1), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$, or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$.

From here, the proof proceeds in two subcases:

Subcase 1. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$

We apply the inductive hypothesis to (2) to get two sub-subcases.

Subcase 1a. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2)$.

Since $n > 0$, we can apply the dynamic rule D-Binary-v to get

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v,$$

where $v = e_1 \odot e_2$, and $n = n' + 1$.

Subcase 1b. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2'$. By D-Binary-2,

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid V \mid e_1 \odot e_2'$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$. Then, by D-Binary-1,

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1' \odot e_2$$

The desired result holds in all subcases.

Case 3 [T-∨-succ]**.**

Suppose the last rule in the typing derivation is T-∨-succ:

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau \vee \downarrow\uparrow A}\ \text{(T-∨-succ)}$$

By assumption, we get $n > 0$. By inversion, we have:

$$\dagger\ \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A$$

We know that the last rule for deriving † is T-R-Shift:

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathtt{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A}\ \text{(T-R-Shift)}$$

By inversion, we have:

(1) $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A$

(2) $\Sigma = \downarrow\Sigma'$

(3) $\Omega = \Omega', \Omega''_{\mathtt{ck}}$

By assumption $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and (2), it follows from Lemma 27 that $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. Then the desired result follows directly by application of Lemma 26 to (1) (since $n > 0$).

□

Lemma 3 *If* $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ *and* $\Sigma = \downarrow\Sigma'$*, then* $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$.

Proof: The proof is by induction on the structure of $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. For each step in the derivation, we build the corresponding step of a derivation for $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$ according to the well-formedness definition. □

Theorem 6.1 (Progress for Commands) *If* $\mathtt{Md} \mid b \mathrel{\mathcal{R}} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$, *then* $\forall n : \mathtt{nat}$ *with* $n \mathcal{R} m$ *and* $\forall \gamma, \mathsf{N}, \mathsf{V}$ *with* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, *either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ *or*
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b \mathrel{\mathcal{R}} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$. We consider a specific (co-)natural number $n \mathcal{R} m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega' \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'}\ (\text{T-ENOUGH?})$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n:

Subcase 1 [$n = 0$]**.** By the value rule V-c-Crash, we have

$$Val(\gamma \mid \mathsf{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid c),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, and since $n > 0$, we have

(1) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$

(3) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\}$

(4) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis on this judgment to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ or
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

which completes the proof of this subcase.

Case 2 [T-If]**.**

Suppose the last rule in the typing derivation is T-if and $c = \textbf{if}\ e\ \textbf{then}\ c_1\ \textbf{else}\ c_2$:

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} \\ \mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1\ : \tau \dashv \Omega' \\ \mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2\ : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid\ b\ > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau \dashv \Omega'}\ (\text{T-If})$$

By assumption, we know that $n > 0$. By inversion, we have:

(1) $\mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}}$

(2) $\mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1\ : \tau \dashv \Omega'$

(3) $\mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2\ : \tau \dashv \Omega'$

By Theorem 16 applied to (1), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

From here, the proof proceeds in two subcases:

Subcase 1. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$.

Since $\mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathit{bool}}$, we have by inversion on T-Enough? followed by inversion on T-∨-succ that

$$\mathtt{Md} \mid\ b : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \downarrow\uparrow \mathit{bool}$$

Again, by inversion on T-R-Shift, we have

(1) $\mathtt{Md} \mid\ b : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \uparrow \mathit{bool}$

(2) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

(3) $\Sigma = \downarrow\Sigma'$

By inversion on the rules T-bool-t and T-bool-f, we have that either e is **tt** or **ff**.

Subcase 1a. If $e = \mathbf{tt}$, then since $n > 0$ the rule D-If-tt applies and yields the desired result.

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1}\ (\text{D-If-tt})$$

Subcase 1b. Similarly, if $e = \mathbf{ff}$, then since $n > 0$, the rule D-If-ff applies and yields the desired result.

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{ff}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2}\ (\text{D-If-ff})$$

Subcase 2. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \;\to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$. Then the rule D-IF applies and produces the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \; e \;\to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 \to \\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2\end{array}} \text{(D-IF)}$$

Case 3 [(T-LET)]**.**

Suppose the last rule in the typing derivation is T-LET:

$$\frac{\begin{array}{c}\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \;@\; \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{let}\, x = e_1\, \textbf{in}\, c : \tau \dashv \Omega'} \text{(T-LET)}$$

By assumption, we know $n > 0$. By inversion, we have a typing derivation for:

(1) $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid\; \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \;@\; \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$

By Theorem 16 applied to (1), either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$.

The proof proceeds in two subcases.

Subcase 1. Suppose that $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$. Then since $n > 0$, the rule D-LET-STEP-V applies and yields the desired result. Note that here γ' extends γ by adding a new mapping from x to ℓ.

$$\frac{\begin{array}{c}Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \; e_1) \\ \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \; \mathit{fresh} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{let}\, x = e_1\, \textbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell @ \mathtt{Ck} \hookrightarrow e_1 \mid c} \text{(D-LET-STEP-V)}$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \;\to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$. Then since $n > 0$, the rule D-LET-STEP applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \; e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{let}\, x = e\, \textbf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{let}\, x = e'\, \textbf{in}\, c} \text{(D-LET-STEP)}$$

Case 4 [T-∨-SUCC].

Suppose the last rule in the typing derivation is T-$\vee$-Succ:

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \tau \vee \downarrow\uparrow\mathtt{unit} \dashv \Omega'} \text{ (T-}\vee\text{-Succ)}$$

Then since $n > 0$, the rule V-skip applies and yields the desired result.

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip})} \text{ (V-skip)}$$

Case 5 [T-Assign]**.**

Suppose the last rule in the typing derivation is T-Assign:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A \dashv \Omega_1\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} p := e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega_1} \text{ (T-Assign)}$$

By assumption, we know that $n > 0$. By inversion on T-Assign, we have

(i) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(ii) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A \dashv \Omega_1$

By Theorem 16 applied to (i), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

The proof proceeds in two subcases.

Subcase 1. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$.

By inversion on T-w-Shift applied to (ii), we have:

$$\frac{\begin{array}{c}\Sigma =\downarrow \Sigma' \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{Wt} p : \uparrow A \dashv \Omega_2 \qquad \Omega_1 = \Omega_2 \setminus \mathit{dom}(\Sigma')\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A \dashv \Omega_1} \text{ (T-w-Shift)}$$

(1) $\Sigma =\downarrow \Sigma'$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{Wt} p : \uparrow A \dashv \Omega_2$

(3) $\Omega_1 = \Omega_2 / \mathit{dom}(\Sigma')$

Now we proceed by inversion on T-Loc-Write applied to (2). From this inversion,

$$\frac{\Omega, \Sigma' = x{:}\uparrow A@q, \Omega_2' \qquad q' = \delta(q, Wt) \neq \texttt{UN}}{\kappa \mid \texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega, \Sigma' \vdash_{Wt} p : \uparrow A \dashv \Omega_2} \text{ (T-Loc-Write)}$$

we have:

(i) $\Omega, \Sigma' = x{:}\uparrow A@q, \Omega_2'$, and

(ii) $q' = \delta(q, Wt) \neq \texttt{UN}$

where $\Omega_2 = x{:}\uparrow A@q', \Omega_2'$

By assumption, we have $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. By Lemma 27, and $\Sigma =\downarrow \Sigma'$, we have $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. By the well-formedness definition, one of the two subcases hold:

Subcase 1a. $\mathsf{V} = \ell@q \hookrightarrow v', \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, the D-Assign-V rule applies and yields the desired result.

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \texttt{wt}) \neq \texttt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \to \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \textbf{skip}} \text{ (D-Assign-V)}$$

Subcase 1b. $\mathsf{N} = \ell@q \hookrightarrow v', \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, the D-Assign-N rule applies and yields the desired result. By definition of δ, it follows that $q \neq \texttt{Rd}$, for we cannot have $\delta(\texttt{Rd}, Wt) \neq \texttt{UN}$.

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \texttt{wt}) \neq \texttt{UN} \qquad q \neq \texttt{RD} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \to \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \textbf{skip}} \text{ (D-Assign-N)}$$

Subcase 2. $\exists(\gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \to \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$. The rule D-Assign-step applies and yields the desired result.

$$\frac{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \to \ \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid p := e \ \to \gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid p := e'} \text{ (D-Assign-step)}$$

Case 6 [(T-seq)]**.**

Suppose the last rule in the typing derivation is T-seq:

$$\frac{\begin{array}{c} \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega' \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega'; \Sigma \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1; c_2 \ : \tau \dashv \Omega''} \text{ (T-seq)}$$

Then the desired result follows by D-SEQ:

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2}\ \text{D-SEQ}$$

Case 7 [(T-SEQ-D)]**.** Suppose the last rule in the typing derivation is T-SEQ-D:

$$\frac{\begin{array}{c} W = \gamma \mid V \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \\ \Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''}\ \text{(T-SEQ-D)}$$

By inversion we have

(1) $W = \gamma \mid V$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$

(3) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$

(4) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

By the inductive hypothesis applied to (1), we have that either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$ or
- $\exists(\gamma \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \to \gamma \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$.

The proof proceeds in two subcases.

Subcase 1. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$.

By (2), $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$, and the assumption $n > 0$, it follows by inversion on T-ENOUGH? that

(1) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}\}$

(2) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$

(3) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c_1 : \mathsf{C}_{\mathtt{unit}}$

(4) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$

By definition, $\mathsf{C}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Additionally, observe that $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$ implies that $c_1 = \mathsf{skip}$. Then by inversion of T-$\vee$-SUCC

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit} \dashv \Omega'}\ \text{(T-}\vee\text{-SUCC)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'$.

The assumption $n > 0$ implies that $n = n' + 1$. By this fact, (1), and $\mathsf{V} = \mathsf{V} \restriction \mathit{dom}(\mathsf{V})$ (which is trivially satisfied because $\mathsf{V} = \mathsf{V}$), the rule D-SEQ-V applies

$$\frac{n = n' + 1 \qquad W = \gamma \mid \mathsf{V} \qquad \mathsf{V} = \mathsf{V} \restriction dom(\mathsf{V})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2} \text{ (D-SEQ-V)}$$

Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2$ is the desired result.

Subcase 2. $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$.

Since $n > 0$, the rule D-CONT applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_W c_2 \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{ (D-SEQ-STEP)}$$

□

Axiom 1 (positive input to generation channel) $\varepsilon \# \mathsf{i\!n}() : \mathtt{nat} > 0$.

Lemma 4 (well-typedness of expressions under crash in jit**)** $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}\}$.

Proof: Note that $\mathsf{C}_{\mathtt{A}}^{\mathsf{jit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}$. Let $\Omega' = \Omega, \Omega''_{\mathsf{ck}}$ where $\Sigma = \downarrow\Omega''$. By axiom 3, we have $\varepsilon \# \mathsf{i\!n}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumption that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}} \in \mathtt{Sig}' \qquad \Omega' = \Omega, \Omega''_{\mathsf{ck}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{RD;Sig}'} \uparrow e : \uparrow \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}} \text{ (T-JIT-RESTORE)} \qquad \varepsilon \# \mathsf{i\!n}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega, \Omega''_{\mathsf{ck}} \vdash_{\mathtt{RD;Sig}'} \varepsilon \# \mathsf{i\!n}(b > 0, \uparrow e) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}})} \text{ (T-CHARGE)} \qquad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} \downarrow\varepsilon \# \mathsf{i\!n}(b > 0, \uparrow e) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}})} \text{ (T-JIT-STOP)}}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{A}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}} \text{ (T-}\vee\text{-CRASH)}$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{A}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}$ yields the desired result.

□

Lemma 5 (well-typedness of expressions under crash in alD**)** *If* $\mathsf{alD}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{RD;Sig}} e' : \tau$ *then* $\mathsf{alD}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau$.

Proof: Let the type $\tau = \mathsf{C}_{A}^{\mathsf{alD}}$ and note that $\mathsf{C}_{A}^{\mathsf{alD}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}^{\mathsf{alD}}) \vee \downarrow\uparrow\mathtt{A}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow e' : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ \text{(T-aID-Restore)} \quad \varepsilon \,\#\, \mathsf{i\!\!\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{i\!\!\eta}(b > 0, \uparrow e') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-Charge)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} \downarrow \varepsilon \,\#\, \mathsf{i\!\!\eta}(b > 0, \uparrow e') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-aID-stop)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} e' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{A}}\ \text{(T-}\vee\text{-Crash)}$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig}$

(2) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

(3) $\varepsilon \,\#\, \mathsf{i\!\!\eta}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow e : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ \text{(T-aID-Restore)} \quad \varepsilon \,\#\, \mathsf{i\!\!\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{i\!\!\eta}(b > 0, \uparrow e) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-Charge)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow \varepsilon \,\#\, \mathsf{i\!\!\eta}(b > 0, \uparrow e) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ \text{(T-aID-stop)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{A}}\ \text{(T-}\vee\text{-Crash)}$$

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{A}$ yields the desired result.

□

Lemma 6 (well-typedness of commands under crash in `jit`) $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}\}$.

Proof: Note that $\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Let $\Omega' = \Omega, \Omega''_{\mathsf{ck}}$ where $\downarrow\Omega'' = \Sigma$. By axiom 3, we have that $\varepsilon \,\#\, \mathsf{i\!\!\eta}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumptions $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}' \quad \Omega' = \Omega, \Omega''_{\mathsf{ck}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{Sig}'} \uparrow c : \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}}\ \text{(T-Jit-Restore)} \quad \varepsilon \,\#\, \mathsf{i\!\!\eta}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega' \vdash_{\mathtt{Sig}'} \varepsilon \,\#\, \mathsf{i\!\!\eta}(b > 0, \uparrow c) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ \text{(T-Charge)} \quad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} \downarrow \varepsilon \,\#\, \mathsf{i\!\!\eta}(b > 0, \uparrow c) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ \text{(T-Jit-stop)}}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-Crash)}$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$ yields the desired result. □

Lemma 7 (well-typedness of commands under crash in aID) *If* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c' : \tau$ *then* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$.

Proof: Let the type $\tau = \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}$ and note that $\mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow c' : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ (\text{T-aID-Restore}) \quad \varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow c') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ (\text{T-Charge})}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} \downarrow \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow c') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ (\text{T-aID-stop})}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ (\text{T-}\vee\text{-Crash})$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig}$

(2) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

(3) $\varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow \Omega'' \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \in \mathtt{Sig} \quad \Omega = \Omega', \Omega''_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \uparrow c : \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}}\ (\text{T-aID-Restore}) \quad \varepsilon \# \mathsf{i\eta}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow c) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ (\text{T-Charge})}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow \varepsilon \# \mathsf{i\eta}(b > 0, \uparrow c) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}})}\ (\text{T-aID-stop})}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ (\text{T-}\vee\text{-Crash})$$

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$ yields the desired result.

□

Theorem 12 (preservation for expressions) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \geq 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau$$

and for some $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ *and natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$$

then

$$\mathtt{Md} \mid b \geq 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e' : \tau$$

with $n' \geq 0$.

Proof: The proof is by induction on the size of e. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Case 1. [D-Binary-1]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1' \odot e_2} \text{ (D-Binary-1)}$$

By the first premise of D-Binary-1, we have that $n > 0$. By inversion on the assumed typing judgment (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau.$$

By inversion on $(\dagger_1)$ via T-Binary

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

By the inductive hypothesis applied to (1) and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' : \tau^s,$$

and $n' \geq 0$. By the following application of the T-Binary rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' : \tau^s \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s} \text{ (T-Binary)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$.

In both subcases, we have the typing judgments $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$ and $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$. Then the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s \quad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau}{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau}\ (\text{T-ENOUGH?})$$

Case 2 [D-BINARY-2].

$$\frac{n > 0 \qquad \gamma \mid \mathit{Val}(\mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_2'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_1 \odot e_2'}\ (\text{D-BINARY-2})$$

By the first premise $n > 0$. By inversion on (†) via T-ENOUGH? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s.$$

By inversion on $(\dagger_1)$ via T-BINARY

$$\frac{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \quad \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s}\ (\text{T-BINARY})$$

we learn that

(1) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

By the inductive hypothesis applied to $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$ and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_2'$, it follows that

$$\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2' : \tau^s,$$

and $n' \geq 0$. By the following application of the T-BINARY rule we get

$$\frac{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \quad \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2' : \tau}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s}\ (\text{T-BINARY})$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. By $\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau} \text{ (T-ENOUGH?)}$$

Case 3. [D-Binary-v].

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2) \qquad v = e_1 \odot e_2}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-Binary-v)}$$

By the first premise $n > 0$ and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s.$$

By inversion on ($\dagger_1$) via T-Binary

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \quad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

From (1) and (2), we apply inversion on T-Enough?, T-$\vee$-Succ, and T-R-Shift to get $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e_1 : \uparrow A^s$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e_2 : \uparrow A^s$ for $\Sigma =\downarrow \Sigma'$. By definition of $\odot$ operator, we have $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} v : \uparrow A^s$ for $v = e_1 \odot e_2$. By application of T-$\vee$-Succ and T-R-Shift we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau} \text{ (T-ENOUGH?)}$$

Case 4. [D-Var-Read].

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell @ q \hookrightarrow v, \mathsf{V}' \qquad \delta(q, \mathtt{RD}) \neq \mathtt{UN} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-VAR-READ)}$$

By the last premise $n > 0$ and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s.$$

By inversion on ($\dagger_1$) via T-$\vee$-Succ

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau_1 \vee \downarrow\uparrow A^i} \text{ (T-}\vee\text{-SUCC)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i$.

By inversion on $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i$ via T-R-Shift

$$\frac{\Sigma = \downarrow\Sigma' \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^i}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i} \text{ (T-R-SHIFT)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^i$ and $\Sigma = \downarrow\Sigma'$.

By Lemma 27 applied to the assumption $\vdash_\gamma^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and premise $\Sigma = \downarrow\Sigma'$, we know that $\vdash_\gamma^{Md} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. By definition of well-formedness, we know that $\gamma = \gamma', [x \mapsto \ell]$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^s$

By application of T-$\vee$-Succ and T-R-Shift we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$.

In both subcases, we have $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$. Then the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau}\ \text{(T-ENOUGH?)}$$

Case 5. [D-N-Read].

The proof is analogous to the previous case.

□

Definition 15 *We write* $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$ *where* $x{:}\tau@q \in \Sigma'$ *iff* $\gamma = [x \mapsto \ell], \gamma'$ *and* $x{:}\tau@q \in \Sigma$ *and* $\ell \in dom(\mathsf{V})$.

Lemma 8 (equality of trimmed volatile contexts) *If*

(i) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$
(ii) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,
(iii) $\vdash^{\mathtt{Md}}_{\gamma''} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma''$,
(iv) $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ *and* $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$
(v) $\gamma_0 \subseteq \gamma$ *and* $\gamma_0 \subseteq \gamma''$
then $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

Proof: We need to show that $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$. The proof proceeds by proving each direction separately:

Ad ⇒. Let $x : \downarrow\uparrow A@q \in \Sigma'$. Then by (i), we have

(1) $x : \downarrow\uparrow A@q \in \Sigma$
(2) $\gamma_0 = [x \mapsto l], \gamma_0'$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma''$. From (2) and $\gamma_0 \subseteq \gamma''$, it follows that $\exists \gamma_0'' \supseteq \gamma_0'$ such that $\gamma'' = [x \mapsto l], \gamma_0''$ ($*$). By (iv) and (3), we have that $l \in dom(\mathsf{V}')$. So, $\exists \mathsf{V}_0'$, v such that $\mathsf{V}' = \mathsf{V}_0', l \hookrightarrow v$ ($**$) and $\cdot \vdash v : \uparrow A$ ($***$). Note that inverting (iii) via V-Loc yields the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma_0''} \mathsf{N}' \mid \mathsf{V}_0' : \Omega \mid \Sigma_0''$ ($\dagger$) and $q = \mathtt{Ck}$ ($\ddagger$). Then, it follows by V-Loc applied to ($*$), ($**$), ($***$), ($\dagger$), ($\ddagger$) that $x : \downarrow\uparrow A@q \in \Sigma''$.

Ad ⇐. Let

(1) $x : \downarrow\uparrow A@q \in \Sigma''$
(2) $\gamma_0 = [x \mapsto l], \gamma_0'$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma'$. To this end, it suffices to show that $x : \downarrow\uparrow A@q \in \Sigma$. By (3) and $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$, we have that $l \in dom(\mathsf{V})$. Therefore, $\exists\, \mathsf{V}'_0, v$ such that $\mathsf{V} = \mathsf{V}'_0, l@q \hookrightarrow v$ $(*)$, $\cdot \vdash v : \uparrow A$ $(**)$, and $q = \mathtt{Ck}$. From (2) and $\gamma_0 \subseteq \gamma$, we have that $\exists\, \gamma' \supseteq \gamma'_0$ such that $\gamma = [x \mapsto l], \gamma'$. Note that inverting (ii) via V-Loc yields the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'_0 : \Omega \mid \Sigma_0$ $(***)$. By V-Loc applied to $(*)$, $(**)$, $(***)$, $q = \mathtt{Ck}$, and $\gamma = [x \mapsto l], \gamma'$, we have

$$x : \downarrow\uparrow A@q \in \Sigma$$

Then it follows by definition 28 applied to (i) that

$$x : \downarrow\uparrow A@q \in \Sigma'.$$

We have shown that $x : \downarrow\uparrow A@q \in \Sigma'$ iff $x : \downarrow\uparrow A@q \in \Sigma''$, $\gamma = [x \mapsto l], \gamma'$, and $l \in dom(\mathsf{V})$. The desired result holds by definition 28. □

Lemma 9 (well-formedness of smaller memories) *If*

(i) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,

(ii) $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$,

(iii) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma')$, *and*

(iv) $\gamma' \subseteq \gamma$

then $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$.

Proof: By (2), observe that $\mathsf{V} \supseteq \mathsf{V}''$. The proof proceeds by induction on the size of $\mathsf{V} - \mathsf{V}''$.

Base case: $|\mathsf{V} - \mathsf{V}''| = 0$**.** If $|\mathsf{V} - \mathsf{V}''| = 0$, then $\mathsf{V} = \mathsf{V}''$ and the desired result holds by assumption (1).

Inductive case. Let $|\mathsf{V} - \mathsf{V}''| = k + 1$ where $|\mathsf{V}_0 - \mathsf{V}''| = k$ where $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$ and $\cdot \vdash v : \uparrow A$. Let $x{:}\downarrow\uparrow A@q \in \Sigma$ such that $\Sigma = \Sigma_0, (x{:}\downarrow\uparrow A@q)$. By inversion on V-Loc applied to the assumed judgment:

$$\frac{\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 : \Omega \mid \Sigma_0 \qquad \gamma = [x \mapsto l], \gamma_0 \qquad \mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v \qquad \cdot \vdash v : \uparrow A}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma_0, (x{:}\downarrow\uparrow A@q)}\ \text{V-Loc}$$

we learn that

(1) $\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 : \Omega \mid \Sigma_0$

(2) $\gamma = [x \mapsto l], \gamma_0$

(3) $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$

(4) $\cdot \vdash v{:}\uparrow A$

Now we need to show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$.

Ad $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$**.** To show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it suffices to show that $l \notin dom(\mathsf{V}')$:

By assumption, it follows that $l \in \mathsf{V} - \mathsf{V}''$, or equivalently, $l \in \mathsf{V}$ and $l \notin \mathsf{V}''$. Now suppose that $l \in dom(\mathsf{V}')$. Then it follows by (ii) that $l \in \mathsf{V}''$, a contradiction. Therefore, we have $l \notin dom(\mathsf{V}')$.

Therefore, $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ follows by $l \notin dom(\mathsf{V}')$ and (ii).

Ad $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$**.** To show $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we first need to show that

(a) $dom(\mathsf{V}') \subseteq dom(\mathsf{V}_0)$

(b) $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$

(c) $\gamma \supseteq \gamma'$

(d) $\gamma_0 \supseteq \gamma'$

(a) follows by $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$. Towards (b), note that since $\mathsf{V} \supseteq \mathsf{V}''$ by assumption, we have $dom(\mathsf{V}) \supseteq dom(\mathsf{V}'')$. By $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it follows that $dom(\mathsf{V}'') = dom(\mathsf{V}')$. Therefore, $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$ (b) holds. (c) follows by assumption (iv). By definition, $\gamma = [x \mapsto l], \gamma_0$, so γ_0 is the smallest subset of γ not containing $x \mapsto l$. Observe that γ' is a subset of γ that does not contain $x \mapsto l$. So, $\gamma \supseteq \gamma_0 \supseteq \gamma'$, which concludes the proof of (d).

$\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$ follows by lemma 36 applied to (iii), (a), (b), (c), and (d).

By the inductive hypothesis applied to (1), $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we have $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. This is the desired result.

□

Lemma 10 (Monotonicity of volatile memories) *If* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *where* $c \neq c_1;_W c_2$*, then* $dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$*,* $\gamma \subseteq \gamma'$*.*

Proof: The proof is straightforward, proceeding in cases on the dynamic rules. □

Definition 16 $\Omega > \Omega'$ *iff* $\forall x{:}\tau@q \in \Omega$*, we have* $x{:}\tau@q' \in \Omega'$ *where* $q' \neq UN$ *and either* $q = q'$ *or* $\delta(q, Wt) = q'$*.*

Lemma 11 *If* $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$*, then* $dom(\Omega) \subseteq dom(\Omega')$*.*

Proof: The proof is by induction on the size of c. We consider possible cases for $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$.

Case [T-C-Shift]**.**

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \restriction dom(\Omega)} \text{ (T-C-Shift)}$$

By definition of $\restriction$, we obtain the desired result $dom(\Omega) \subseteq dom(\Omega_1 \restriction dom(\Omega))$.

Case [T-SEQ]**.**

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \quad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''} \text{ (T-SEQ)}$$

By inversion of T-SEQ, we learn that

- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

By applying the inductive hypothesis to each of these judgments, we learn that $dom(\Omega) \subseteq dom(\Omega')$ and $dom(\Omega') \subseteq dom(\Omega'')$. By transitivity, we establish the desired result $dom(\Omega) \subseteq dom(\Omega'')$.

Case [T-SEQ-D]**.** This case is similar to T-SEQ.

Case [T-$\vee$-SUCC, T-LET, T-ASSIGN, T-IF, T-ENOUGH?]**.** In each case, we apply the inductive hypothesis to learn that $dom(\Omega) \subseteq dom(\Omega')$.

□

Theorem 6.2 (Preservation for Commands) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$$

and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed,* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*,* $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V})$*, and for some (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$$

then for some Σ'*, and* Ω_0

$$\mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega_0; \Sigma' \vdash_{\mathtt{Sig}} c' : \tau \dashv \Omega'$$

where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega_0 \mid \Sigma'$*,* $\Omega > \Omega_0$*, and* $n' \geq 0$*. Moreover* $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *is well-formed. Moreover,* $dom(\mathsf{N}) = dom(\mathsf{N}')$*, and if* $\mathtt{Md} = \mathsf{adv}(c_0)$*, then* $\mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$ *and* $dom(\mathsf{N}'_{\mathsf{ck}}) \subseteq dom(\mathsf{V}')$*.*

Proof: The proof is by induction on the size of c. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \ \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Case 1 [D-LET-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\ x = e\ \mathbf{in}\ c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\ x = e'\ \mathbf{in}\ c} \text{ (D-LET-STEP)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}'' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau\}$.

By inversion on ($\dagger_1$) via T-Let

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau \dashv \Omega'}\ (\text{T-Let})$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$

By the application of Theorem 17 to (1) and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}},$$

and $n' \geq 0$. By the following application of the T-Let rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau \dashv \Omega'}\ (\text{T-Let})$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau^s\}$. It follows via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau^s\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1, \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e'\, \mathbf{in}\, c : \tau \dashv \Omega'}\ (\text{T-enough?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e' \,\mathbf{in}\, c$ follows by definition 26, and $\Omega > \Omega$ follows by definition 7, both vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. The detail $dom(\mathsf{N}_{\mathtt{ck}}) \subseteq dom(\mathsf{V})$ follows by assumption.

Case 2. [D-Let-v]**.**

$$\frac{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \; fresh \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e \,\mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c} \text{ (D-Let-v)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau.$$

By inversion of $(\dagger_1)$ via T-Let

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \;@\; \mathtt{Ck} \vdash c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau \dashv \Omega'} \text{ (T-Let)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \;@\; \mathtt{Ck} \vdash c : \tau \dashv \Omega'$

The typing judgment (2) completes the proof, if we can show that $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e : \Omega \mid \Sigma, x{:}{\downarrow}{\uparrow}A@\mathtt{Ck}$.

By $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-∨-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : {\uparrow}A,$$

where $\Sigma = {\downarrow}\Sigma'$.

This is enough to prove $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e : \Omega \mid \Sigma, x{:}{\downarrow}{\uparrow}A@\mathtt{Ck}$.

Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c$ follows by definition 26, vacuously because c is not of the form $c';_W c''$. Observe that $\Omega > \Omega$ follows by definition 7 vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. The detail $dom(\mathsf{N}_{\mathtt{ck}}) \subseteq dom(\mathsf{V}, \ell@\mathtt{CK} \hookrightarrow e)$ follows by assumption and the observation that $dom(\mathsf{V}) \subseteq dom(\mathsf{V}, \ell@\mathtt{CK} \hookrightarrow e)$.

Case 3 [D-Assign-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'} \text{ (D-Assign-step)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \ = \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \ \geq \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau\}$.

By inversion on $(\dagger_1)$ via T-Assign

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'} \text{ (T-Assign)}$$

we have

- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ and
- $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'$.

By the application of Theorem 17 on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \ \geq \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}},$$

and $n' \geq 0$. By the following application of the T-Assign rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'} \text{ (T-Assign)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau\}$. It follows by lemma 30 that $\mathtt{Md} \mid b \ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_1} x := e' : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 31, we get $\mathtt{Md} \mid b \ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b \ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} x := e' : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega'} \text{ (T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'$ follows by definition 26, vacuously. The detail $\Omega > \Omega$ follows by definition 7, vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. The detail $dom(\mathsf{N}_{\mathtt{ck}}) \subseteq dom(\mathsf{V})$ follows by assumption.

Case [D-Assign-V]**.**

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau\}$.

By inversion on $(\dagger_1)$ via T-Assign

$$\frac{\begin{array}{c} \mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega' \end{array}}{\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e \;: \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'} \text{ (T-Assign)}$$

we have

(1) $\mathtt{Md} \mid \; b \;\geq 0 : \mathtt{nat} \;\mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'$

By V-Loc and the definition of V, we have that $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}', (\ell@\mathtt{Ck} \hookrightarrow v') : \Omega \mid \Sigma_0, (x :\downarrow\uparrow A@\mathtt{Ck})$ where $\Sigma = \Sigma_0, (x :\downarrow\uparrow A@\mathtt{CK})$. It follows by inversion on T-w-Shift and T-Loc-Write

$$\frac{\dfrac{\begin{array}{c} \Sigma = \downarrow\Sigma' \qquad \Omega = \Omega_0, \Omega_{1\,\mathtt{ck}} \\ \Omega_0, \Sigma' = x : \uparrow A@\mathtt{CK}, \Omega_2' = \Omega'' \qquad \mathtt{CK} = \delta(\mathtt{CK}, Wt) \neq \mathit{UN} \end{array}}{\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid \Omega_0, \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A \dashv \Omega''} \text{ (T-Loc-Write)}}{\mathtt{Md} \mid \; b \;> 0 : \mathtt{nat} \;\mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'' \restriction dom(\Omega)} \text{ (T-w-Shift)}$$

that

- $\Omega' = \Omega'' \restriction dom(\Omega)\ (\exists \Omega'')$
- $\Omega'' = \Omega_0, \Sigma'$
- $\Sigma = \downarrow\Sigma'$

- $\Omega = \Omega_0, \Omega_{1\,\mathsf{ck}}$

It follows by assumption $\mathsf{N}_{\mathsf{ck}} \subseteq \mathsf{V}$ and the well-formedness condition $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ that $\Omega_{1\,\mathsf{ck}} \subseteq \Sigma'$. Hence $\Omega \subseteq \Omega_0, \Sigma'$. Since $\Omega'' = \Omega_0, \Sigma'$, it follows that $\Omega \subseteq \Omega''$, and hence $\Omega = \Omega'' \restriction dom(\Omega)$. Since $\Omega' = \Omega'' \restriction dom(\Omega)$, we have that $\Omega = \Omega'$.

By the premise $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$.

Appying V-Loc, we can show that $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}', (\ell@\mathtt{Ck} \hookrightarrow e) : \Omega \mid \Sigma_0, (x :\downarrow\uparrow A@\mathtt{Ck})$ where $\Sigma' = \Sigma_0, (x :\downarrow\uparrow A@\mathtt{Ck})$.

We want to show that

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega).$$

Noting that $\Omega = \Omega'' \restriction dom(\Omega)$, it follows by T-Skip, T-C-Shift, and T-$\vee$-Succ that

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega.$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\}$. From lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [Md = aID(c_0)]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathtt{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega}\ \text{(T-Enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}$ follows by definition 26, vacuously, and $\Omega > \Omega$ follows by definition 7, both vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. Lastly, note that $dom(\mathsf{V}) = dom(\mathsf{V}', \ell@q' \hookrightarrow e)$. Therefore, it follows by assumption that $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}', \ell@q' \hookrightarrow e)$.

Case [D-Assign-N]**.**

$$\frac{\begin{array}{c}Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, Wt) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}}\ \text{(D-Assign-N)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \texttt{Md} \mid b \ > \ 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} x := e : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \texttt{Md} \mid b \ = \ 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}''} x := e : \tau$$

where $\texttt{Sig}'' = \mathit{if}\ \texttt{Md} = \mathsf{jit}\ \mathit{then}\ \texttt{Sig}'\ \mathit{else}\ \texttt{Sig}$ and $\texttt{Sig}' = \{\texttt{Md} \mid b \ \geq \ 0 : \texttt{nat} \mid \Omega; \Sigma \vdash x := e : \tau\}$.
By inversion on ($\dagger_1$) via T-Assign

$$\frac{\begin{array}{c}\texttt{Md} \mid \ b \ \geq 0 : \texttt{nat} \ \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \mathsf{C}_A^{\texttt{Md}} \\ \texttt{Md} \mid \ b \ > 0 : \texttt{nat} \ \mid \ \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega'\end{array}}{\texttt{Md} \mid \ b \ > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} x := e \ : \mathsf{C}^{\texttt{Md}}_{\texttt{unit}} \dashv \Omega'} \ \text{(T-Assign)}$$

we learn that

(1) $\texttt{Md} \mid \ b \ \geq 0 : \texttt{nat} \ \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \mathsf{C}_A^{\texttt{Md}}$

(2) $\texttt{Md} \mid \ b \ > 0 : \texttt{nat} \ \mid \ \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega'$

By inversion on (2) via T-w-Shift and T-Loc-Write:

$$\frac{\dfrac{\begin{array}{c}\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega_0, \Omega_{1\,\mathsf{ck}} \\ \Omega_0, \Sigma' = x : \uparrow A @ q, \Omega_2'\end{array}}{\dfrac{\Omega'' = x : \uparrow A @ q', \Omega_2' \qquad q' = \delta(q, Wt) \neq UN}{\texttt{Md} \mid \ b \ > 0 : \texttt{nat} \ \mid \ \Omega_0, \Sigma' \vdash_{\texttt{WT}} x : \uparrow A \dashv \Omega''}\ \text{(T-Loc-Write)}}{\texttt{Md} \mid \ b \ > 0 : \texttt{nat} \ \mid \ \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega'' \restriction dom(\Omega)}\ \text{(T-w-Shift)}$$

we have

(i) $\Omega' = \Omega'' \restriction dom(\Omega)$ $(\exists \Omega'')$

(ii) $\Sigma = \downarrow\Sigma'$

(iii) $\Omega = \Omega_0, \Omega_{1\,\mathsf{ck}}$

(iv) $\Omega_0, \Sigma' = x : \uparrow A @ q, \Omega_2'$

(v) $\Omega'' = x : \uparrow A @ q', \Omega_2'$

(vi) $q' = \delta(q, Wt) \neq UN$

By $\mathit{Val}(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\texttt{Md} \mid \ b \ \geq 0 : \texttt{nat} \ \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \mathsf{C}_A^{\texttt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\texttt{Md} \mid b : \texttt{nat} \mid \Omega, \Sigma' \vdash_{\texttt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$. This is enough to prove $\vdash^{\mathtt{Md}}_{\gamma} (\ell@\mathtt{Ck} \hookrightarrow e, \mathsf{N}'') \mid \mathsf{V} : (x : \uparrow A@q', \Omega'_2) \mid \Sigma$. From here, we note that since $x : \uparrow A@q$, we have $x : \uparrow A@q \in \Omega_0$. Therefore, $\Omega'' = \Omega^0, \Sigma'$ for some $\Omega^0 < \Omega_0$ (i.e., $\Omega^0 = \Omega''_0, x : \uparrow A@q'$ where $\Omega_0 = \Omega''_0, x : \uparrow A@q$).

We now need to show that:

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega).$$

Let $\Omega' = \Omega^0, \Omega^1_{\mathsf{ck}}$. By T-Skip, T-C-Shift, and T-$\vee$-Succ, and recalling that $\Omega' = \Omega'' \restriction dom(\Omega)$ (as defined above), we have

$$\dfrac{\dfrac{\dfrac{\Sigma = \downarrow\Sigma' \qquad \Omega' = \Omega^0, \Omega^1_{\mathsf{ck}}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^0, \Sigma' \vdash_{\mathtt{Sig}} \mathbf{skip} : \uparrow\mathtt{unit} \dashv \Omega''}\ \text{(T-Skip)}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'' \restriction dom(\Omega')}\ \text{(T-C-Shift)}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega')}\ \text{(T-}\vee\text{-Succ)}$$

We now need to show that $dom(\Omega') = dom(\Omega)$. From (i), we have that $dom(\Omega') \subseteq dom(\Omega)$. By Lemma 39, we have that $dom(\Omega) \subseteq dom(\Omega')$.

We consider two subcases based on $\mathtt{Md}$.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\}$. By lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\dfrac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}_2} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega')\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega')}\ \text{(T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$ follows by definition 26 vacuously. The detail $\Omega > \Omega'$ follows by definition 7, (iv) and (v) from above, and the premise $q' = \delta(q, Wt) \neq \mathtt{UN}$. If $\mathtt{Md} = \mathtt{aID}(c_0)$, we want to show that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = (\mathsf{N}', \ell@q' \hookrightarrow e) \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$. By definition, $\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v'$. Since ℓ is written to, $q' = \mathtt{Wtn}$, and either $q = \mathtt{Wtn}$ or $q = \mathtt{MFstWt}$. Now we need to show that $\mathsf{N}', \ell@q \hookrightarrow v' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$. From above, this reduces to showing $\mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$ which holds trivially. Lastly, note that $dom(\mathsf{N}) = dom(\mathsf{N}', \ell@q' \hookrightarrow e)$. Therefore, it follows by assumption, and the definition of δ (i.e. $q' = \mathtt{Ck}$ if $q = \mathtt{Ck}$), that $dom(\mathsf{N}^0_{\mathsf{ck}}) \subseteq dom(\mathsf{V})$ where $\mathsf{N}', \ell@q' \hookrightarrow e = \mathsf{N}^0_{\mathsf{ck}}, \mathsf{N}^1$.

Case 6 [D-If]**.**

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2}\ \text{(D-If)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b\ >\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b\ =\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau\}$.
By inversion on ($\dagger_1$) via T-If

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool} \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1\ : \tau \dashv \Omega' \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2\ : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid\ b\ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if}\, e\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau \dashv \Omega'}\ \text{(T-If)}$$

we learn that

(1) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool}$

(2) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1\ : \tau \dashv \Omega'$

(3) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2\ : \tau \dashv \Omega'$.

By the application of Theorem 17 on (1) and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b\ \geq\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_{bool},$$

and $n' \geq 0$. By the following application of the T-If rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_{bool} \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1\ : \tau \dashv \Omega' \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2\ : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid\ b\ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau \dashv \Omega'}\ \text{(T-If)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau\}$. By lemma 30, we get $\mathtt{Md} \mid\ b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_1} \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid\ b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if}\, e'\, \textbf{then}\, c_1\, \textbf{else}\, c_2 : \tau^s$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'}\ \text{(T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2$ follows by definition 26, and $\Omega > \Omega$ follows by definition 7, both vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. The detail $\mathit{dom}(\mathsf{N}_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V})$ follows by assumption.

Case 7. [D-If-tt]**.**

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1}\ \text{(D-If-tt)}$$

By the first premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau\}$.

By inversion on $(\dagger_1)$ via T-If

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} \mathtt{tt} : \mathsf{C}^{\mathtt{Md}}_{\mathit{bool}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'}\ \text{(T-If)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} \mathtt{tt} : \mathsf{C}^{\mathtt{Md}}_{\mathit{bool}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega'$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1$ follows by definition 26, vacuously because c_1 is not of the form $c';_W c''$. The detail $\Omega > \Omega$ follows vacuously by definition 7.

The typing judgment (2) completes the proof, because $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ holds by assumption. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. The detail $\mathit{dom}(\mathsf{N}_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V})$ follows by assumption.

Case 8 [D-IF-FF]**.** Similar to the previous case.

Case 9 [D-SEQ]**.**

$$\frac{n > 0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1; c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2}\ \text{(D-SEQ)}$$

By the premise $n > 0$. By inversion on (†) via T-ENOUGH? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b\ >\ 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau^s \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b\ =\ 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}''} c_1; c_2 : \tau^s$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash c_1; c_2 : \tau^s\}$.

By inversion on $(\dagger_1)$ via T-SEQ

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega' \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid\ b\ >0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2\ : \tau \dashv \Omega''}\ \text{(T-SEQ)}$$

we learn that

(1) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega'$

(2) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau^s \dashv \Omega''$

Put $W = \gamma \mid \mathsf{V}$. We now want to show that $\Sigma = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. By definition 28, it is straightforward to see that $\mathsf{trim}(\Sigma, \mathsf{V}, \gamma) \subseteq \Sigma$. We need to show $\Sigma \subseteq \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. Let $x : \downarrow\uparrow A@q \in \Sigma$ be arbitrary and write $\Sigma = \Sigma', (x : \downarrow\uparrow A@q)$. Then by inversion on the well-formedness definition V-LOC:

$$\frac{\mathsf{V} = \mathsf{V}', \ell\ @\ q \hookrightarrow v \qquad \begin{array}{c}\vdash^{\mathsf{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}' : \Omega \mid \Sigma' \\ q = \mathsf{Ck} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \cdot \vdash v : \uparrow A\end{array}}{\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma', (x{:}\downarrow\uparrow A@q)}\ \text{(V-LOC)}$$

we know that $\gamma = \gamma', [x \mapsto \ell]$ with $\ell \in \mathit{dom}(\mathsf{V})$. From here, definition 28 implies that $x \in \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. Because $x \in \Sigma$ was arbitrary, we have shown that $\Sigma \subseteq \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$. Therefore, $\Sigma = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$.

By the following application of the T-SEQ-D rule

$$\frac{\begin{array}{cc}W = \gamma \mid \mathsf{V} & \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega' \\ \Sigma = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma) & \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid\ b\ >0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2\ : \tau \dashv \Omega''}\ \text{(T-SEQ-D)}$$

we learn that $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''$

We consider two subcases based on $\mathtt{Md}$.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1;_W c_2 : \tau\}$. By lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1;_W c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1;_W c_2 : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1;_W c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} c_1;_W c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''}\ (\text{T-ENOUGH?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma|\mathsf{V}} c_2$ follows by definition 26 since $\gamma \subseteq \gamma$ and $\mathit{dom}(\mathsf{N}_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V}) \subseteq \mathit{dom}(\mathsf{V})$. The detail $\Omega > \Omega$ follows by definition 7, vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired.

Case [D-SEQ-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2}\ (\text{D-SEQ-STEP})$$

The premises yield

(a) $n > 0$

(b) $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$

By assumption, note that

- $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega'$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_W c_2$ is well-formed.

Put $W = \gamma_0 \mid \mathsf{V}_0$. Then by definition 26, we have $\mathit{dom}(\mathsf{V}_0) \subseteq \mathit{dom}(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$. Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1$ is well-formed, which vacuously follows by definition 26 because c_1 does not have the form $c';_{W'} c''$ and we always run the command c_1 before c_2.

By inversion of $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega'$ via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c_1;_W c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}'' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 ;_W c_2 : \tau\}$.

By inversion of $(\dagger_1)$ via T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \\ \Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''}\ \text{(T-SEQ-D)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$

(2) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

By the inductive hypothesis applied to (1), (b), the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1$, $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, $n \geq 0$, and $\mathit{dom}(\mathsf{N}_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V})$, we get $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$, where

(i) $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N}' \mid \mathsf{V}' : \Omega_0 \mid \Sigma''$,

(ii) $n' \geq 0$,

(iii) $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$ is well-formed,

(iv) $\Omega > \Omega_0$,

(v) if $\mathtt{Md} = \mathsf{aID}(c_0)$, $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, and

(vi) $\mathit{dom}(\mathsf{N}'_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V}')$

We now need to show that $\mathit{dom}(\mathsf{N}'_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V}_0) \subseteq \mathit{dom}(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$. Observe that $c_1 \neq c';_W c''$ because $c_1 ;_W c_2$ and we always run c_1 first. By lemma 38 and $c_1 \neq c';_W c''$, it follows that $\mathit{dom}(\mathsf{V}) \subseteq \mathit{dom}(\mathsf{V}')$ and $\gamma \subseteq \gamma'$. Then it follows by $\mathit{dom}(\mathsf{V}_0) \subseteq \mathit{dom}(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$ (as shown above), that $\mathit{dom}(\mathsf{V}_0) \subseteq \mathit{dom}(\mathsf{V}) \subseteq \mathit{dom}(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma \subseteq \gamma'$. Additionally, it follows by assumption that $\mathit{dom}(\mathsf{N}) = \mathit{dom}(\mathsf{N}')$, and so it follows by the assumption $\mathit{dom}(\mathsf{N}_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V}_0)$ that $\mathit{dom}(\mathsf{N}'_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V}_0)$. Hence, $\mathit{dom}(\mathsf{N}'_{\mathsf{ck}}) \subseteq \mathit{dom}(\mathsf{V}_0) \subseteq \mathit{dom}(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$.

It suffices to show $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

By lemma 36, it follows that $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

Using $W = \gamma_0 \mid \mathsf{V}_0$, (2), (3), and $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$, we can apply T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \\ \Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0) \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash_{\mathtt{Sig}} c_1' ;_W c_2 : \tau \dashv \Omega''}\ \text{(T-SEQ-D)}$$

We now need to show that $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash_{\mathtt{Sig}} c_1';_W c_2 : \tau$. The proof proceeds in two subcases based on Md:

Case $\mathtt{Md} = \mathsf{jit}$**.** Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau\}$. It follows by lemma 30 that $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau$.

Case $\mathtt{Md} = \mathsf{alD}(c_0)$**.** By ($\dagger_2$) via lemma 31, we can see that $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau$.

In both cases, $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau$.

The desired result follows by T-ENOUGH?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma'' \vdash c_1';_W c_2 : \tau \dashv \Omega''}\ (\text{T-ENOUGH?})$$

and (iv), which asserts that $\Omega > \Omega_0$.

Observe that by definition 26 applied to $dom(\mathsf{N}'_{\mathsf{ck}}) \subseteq dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$ (as shown above), $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2$ is well-formed. In the case where c_1' is of the form $c';_W c''$, the rule stepping c_1 must be D-SEQ since $c_1 \neq c';_W c''$. Thus, observe that $\gamma' \subseteq \gamma'$ and $dom(\mathsf{N}'_{\mathsf{ck}}) \subseteq dom(\mathsf{V}') \subseteq dom(\mathsf{V}')$, and hence it follows by definition 26 that $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2$ is well-formed.

Case 11 [D-SEQ-V]**.**

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2}\ (\text{D-SEQ-V})$$

Observe that $n = n' + 1$ (from the premise of D-SEQ-V) implies $n > 0$, and hence $b > 0$.

By assumption, we have

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip};_W c_2 : \tau \dashv \Omega'$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$.

By inversion on T-SEQ-D, we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega''; \Sigma' \vdash c_2 : \tau \dashv \Omega'$

(3) $W = \gamma' \mid \mathsf{V}'$ (from the premise of D-SEQ-V)

(4) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma')$

We now need to show that $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. Since $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2$ is well-formed (by assumption), it follows by definition 26 that $\gamma' \subseteq \gamma$. Therefore, the desired

result follows by lemma 37 applied to $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, the premise $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$, (4), and $\gamma' \subseteq \gamma$. Observe that (2) yields the desired result where $\vdash^{\mathtt{Md}}_{\gamma''} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2$ follows by definition 26, vacuously because c_2 is not of the form $c';_W c''$. We now need to show that $\Omega > \Omega''$. By inversion of (1) on T-C-Shift, it follows that $\Omega = \Omega''$. Then the desired result follows vacuously by definition 7. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired.

Lastly, we need to show that $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}'')$. That is to say that trimming the volatile context does not throw out any checkpointed locations.

Since $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip};_W c_2$ is well-formed, it follows that $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}')$. It then follows by $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$ that $dom(\mathsf{V}') = dom(\mathsf{V}'')$. Therefore, $dom(\mathsf{N}_{\mathsf{ck}}) \subseteq dom(\mathsf{V}'')$.

Note that $dom(\mathsf{N}) = dom(\mathsf{N}')$ holds by observation, and by the inductive hypothesis in case 10 (D-seq-step). □

Lemma 12 *If* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \mathsf{V} \mid c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}' \mid c'$ *and* $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$*, then* $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}' \mid c'$ *and* $\mathsf{N}'_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\mathtt{MFstWt}\}$.

Proof: The proof is by induction on the size of c. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \mathsf{V} \mid c \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}'_1 \mid \mathsf{V}' \mid c'$.

Case 1 [D-Let-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e'\, \mathbf{in}\, c} \text{ (D-Let-step)}$$

By inversion on D-Let-step, we have:

- $n > 0$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By applying the inductive hypothesis, we have

- $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e'$
- $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$

Observing that $\infty > 0$, we apply D-Let-step to learn that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{let}\, x = e'\, \mathbf{in}\, c$$

Case 2. [D-Let-v]**.**

$$\frac{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \text{ fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c} \text{ (D-Let-v)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By inversion on D-Let-v, we have

(1) $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$

(2) $\gamma' = \gamma, [x \mapsto \ell]$

(3) ℓ *fresh*

(4) $n = n' + 1$

From $n = n' + 1$, we know that $n > 0$. Thus, $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ is not a crash value (i.e. $n \neq 0$). Therefore, $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$ since $\infty > 0$. By application of D-Let-v to $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$, (2), (3), and $\infty > 0$, we have

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c$$

Case 3 [D-Assign-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'} \text{ (D-Assign-step)}$$

By inversion on D-Assign-step, we have

- $n > 0$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By applying the inductive hypothesis, we have

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e'$$

where $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$.

Observing that $\infty > 0$, we apply D-Assign-step to learn that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e'$$

Case 4 [D-Assign-V]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

The proof is analogous to the argument of Case 2 (D-Let-v).

Case 5 [D-Assign-N]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}} \text{ (D-Assign-N)}$$

By inversion of D-Assign-N, we learn that

(1) $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$

(2) $\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v'$

(3) $q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN}$

(4) $\gamma = \gamma', [x \to \ell]$

(5) $n = n' + 1$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. We want to show that $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e \ \to\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2, \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$ and $\mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2, \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\}$ where $\mathsf{N}_2 = \mathsf{N}'_2, \ell@q \hookrightarrow v'$.

From (5), we know that (1) is not a crash value (i.e. $n > 0$, and hence $n \neq 0$). Since $\infty > 0$, we have $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$. By application of D-Assign-NV to $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$, $\mathsf{N}_2 = \mathsf{N}'_2, \ell@q \hookrightarrow v'$, (3), (4), and $\infty > 0$, we have

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e \ \to\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2, \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$$

Towards $\mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2, \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\}$, observe that the assumption $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$ implies $\mathsf{N}' \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\mathtt{MFstWt}\}$. Thus, it follows that $\mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2, \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\}$, as desired.

Case 6 [D-If]**.**

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ e \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e'\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2} \ \text{(D-If)}$$

The proof is similar to the proof of Case 3 (D-Assign-step).

Case 7. [D-If-tt]**.**

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \ \text{(D-If-tt)}$$

By inversion on D-If-tt, we have

(1) $n = n' + 1$

(2) $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})$

From (1), we know that $n > 0$, and hence (2) is not a crash value. Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. Observing that $\infty > 0$, we have that $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathtt{tt})$. By applying D-If-tt, we have that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \to\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c_1$$

Case 8. [D-IF-FF]**.**

$$\frac{n = n' + 1 \qquad Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\ \mathtt{ff}\ \textbf{then}\ c_1\ \textbf{else}\ c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \text{ (D-IF-FF)}$$

The proof is similar to the previous case (D-IF-TT).

Case 9 [D-SEQ]**.**

$$\frac{n > 0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1; c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2} \text{ (D-SEQ)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. Observing that $\infty > 0$, it follows by application of D-SEQ that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid\ c_1; c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2$$

Case 10 [D-SEQ-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{ (D-SEQ-STEP)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By inversion on D-SEQ-STEP, we have

- $n > 0$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$

By the inductive hypothesis, we have

(1) $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}' \mid c_1'$

(2) $\mathsf{N}' \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2' \setminus \{\mathtt{MFstWt}\}$

From (1) and the observation $\infty > 0$, we apply D-SEQ-STEP to learn that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid\ c_1;_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}' \mid c_1';_W c_2$$

Case 11 [D-SEQ-V]**.**

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \textbf{skip};_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2} \text{ (D-SEQ-V)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By inversion on D-SEQ-V, we have

- $W = \gamma' \mid \mathsf{V}'$
- $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$

Observe that $\infty > 0$. By applying D-SEQ-V, we have

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{skip};_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}_2 \mid \mathsf{V}'' \mid c_2$$

□

B.5 Preservation for Closed Configurations

Theorem 13 (Preservation for programs) *Consider* $b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}$*, a nonvolatile memory* N *and a bijective map* γ *that matches qualifiers and types from variables in* Ω *to locations in* N*. For any* n:$\mathtt{nat} \geq 0$*, if we have* $[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid p \Rightarrow_p [\chi' \rhd \varepsilon] \otimes \gamma' \mid n' \mid \mathsf{N}' \mid p'$*, then* $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$*, with* γ *remaining a bijective map from* Ω *to* N'*.*

Proof: By a structural induction on the typing derivation, and case distinction on the step.
Case 1.

$$\frac{\begin{array}{c} n > 0 \\ n' > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid c; p' \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'} \text{ (P-SEQ)}$$

By inversion on the typing rules, we know that $b\mathcal{R}0 : \mathtt{nat} \mid \Omega \mid \cdot \vdash c : \mathsf{C}_{\mathtt{unit}}$ and $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$ By preservation for commands, we know that γ' is well-formed for N' and V' at the jit mode with respect to Ω and some Σ. By definition of well-formedness, we know that for some $\gamma_1 \subseteq \gamma'$, we have γ_1 is well-formed for Ω and N'. But we know that $\gamma \subseteq \gamma'$ is well-formed for Ω. This means that $\gamma = \gamma'$ and thus γ is a bijective map that matches qualifiers and types from variables in Ω to locations in N'.

Case 2.

$$\frac{\begin{array}{c} n > 0 \\ \mathsf{InitWorld}_d(\mathsf{N}; \rho; \mu; \omega) = \mathsf{N}_0, V_0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* \\ [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{Ckpt}[\mathsf{aID}; \rho; \mu; \omega](c_0); p \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p} \text{ (P-CKPT)}$$

By inversion on the typing rules, we know that $b\mathcal{R}0 : \mathtt{nat} \mid \Omega_0 \mid \Sigma_0 \vdash c : \mathsf{C}_{\mathtt{unit}}$ and $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$ By preservation for commands, we know that γ' is well-formed for N' and V' at the jit mode with respect to Ω and some Σ. By definition of well-formedness and $\mathsf{FinWorld}$, we know that for some $\gamma_1 \subseteq \gamma'$, we have γ_1 is well-formed for Ω and N_1. But we know that $\gamma \subseteq \gamma'$ is well-formed for Ω. This means that $\gamma = \gamma'$ and thus γ is a bijective map that matches qualifiers and types from variables in Ω to locations in N_1.

□

B.6 Fundamental Theorem of Logical Relation

Lemma 13 *If* $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$ *and* $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_1 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$*, then* $\exists.(\gamma_1 \mid \mathsf{aID}(\mathsf{c}) \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_1 \mid c_1)$ *such that*

- $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1 \mid \mathsf{aID}(\mathsf{c}) \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_1 \mid c_1$,
- $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}_1' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$*, and*
- $dom(\mathsf{N}_1) = dom(\mathsf{N}_1')$.

Proof: We prove this by induction on n_0:

Base case. If $n_0 = 0$, then the configuration is a value.

Inductive case. Suppose that $n_0 = n_0' + 1$ ($\exists n_0'$). Since $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$ and $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_1 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$, it follows by the progress theorem that either $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c$ is a value or $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c$ is not a value, in which case $\exists\, \gamma_1'' \mid \mathsf{aID(c)} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$ such that

$$\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow \gamma_1'' \mid \mathsf{aID(c)} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$$

where

- $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0''; \Sigma' \vdash_{\mathtt{Sig}} c_1'' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$,
- $\Omega' > \Omega_0''$,
- $\vdash_{\gamma_1''}^{\mathsf{aID}(c)} \mathsf{N}_1'' \mid \mathsf{V}_1'' : \Omega_0'' \mid \Sigma'$,
- $\gamma_1'' \mid \mathsf{aID(c)} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$ is well-formed,
- if $\mathtt{Md} = \mathsf{aID}(c)$, $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}_1'' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, and
- $dom(\mathsf{N}_1) = dom(\mathsf{N}_1'')$.

We get these conditions by applying the preservation theorem (theorem 5) because $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ is well-formed.

By the inductive hypothesis,

$$\gamma_1'' \mid \mathsf{aID(c)} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1'' \rightarrow^* \gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'.$$

where

- $\gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ is well-formed and a value,
- $\Omega_0'' > \Omega_0'$
- $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0'; \Sigma'' \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$,
- $\vdash_{\gamma_1'}^{\mathsf{aID}(c)} \mathsf{N}_1' \mid \mathsf{V}_1' : \Omega_0' \mid \Sigma''$,
- if $\mathtt{Md} = \mathsf{aID}(c)$, $\mathsf{N}_1'' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}_1' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, and
- $dom(\mathsf{N}_1'') = dom(\mathsf{N}_1')$.

By head expansion, we establish that

$$\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

where $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}_1' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$ and $dom(\mathsf{N}_1) = dom(\mathsf{N}_1')$.

□

Lemma 14 *If* $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow^m \gamma_1 \mid \mathsf{aID(c)} \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_1 \mid c_1$ *and* $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$*, then* $\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c \rightarrow^m \gamma_1' \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_1' \mid c_1'$ *where* $\mathsf{N}_1' \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2' \setminus \{\mathtt{MFstWt}\}$.

Proof: The proof proceeds by induction on the number of steps m such that $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow^m \gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$.

Base case ($m = 0$). By assumption, $\mathsf{N}_1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2 \setminus \{\texttt{MFstWt}\}$. Trivially,

$$\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c \rightarrow^0 \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c$$

where $\mathsf{N}_1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2 \setminus \{\texttt{MFstWt}\}$.

Inductive case ($m = k + 1$). Suppose that $m = k + 1$ such that

$$\begin{aligned} &\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \\ \rightarrow &\gamma_1'' \mid \mathsf{aID(c)} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1'' \\ \rightarrow^k &\gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \end{aligned}$$

By assumption, $\mathsf{N}_1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2 \setminus \{\texttt{MFstWt}\}$. Hence, it follows by lemma 25

$$\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c \rightarrow \gamma_1'' \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2'' \mid \mathsf{V}_1'' \mid c_1''$$

where $\mathsf{N}_1'' \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2'' \setminus \{\texttt{MFstWt}\}$. By the inductive hypothesis, it follows that

$$\gamma_1'' \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2'' \mid \mathsf{V}_1'' \mid c_1'' \rightarrow^k \gamma_1' \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_1' \mid c_1'$$

where $\mathsf{N}_1' \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2' \setminus \{\texttt{MFstWt}\}$.

□

Definition 17 (Subset Property) *A judgment* $\mathsf{aID}(c) \mid b \geq 0 : \texttt{nat} \mid \Omega;\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$ *has the subset property iff* $\Omega^0_{\mathsf{ck}} \subseteq \Sigma'$ *where* $\Sigma = \downarrow\Sigma'$ *and* $\Omega = \Omega^0_{\mathsf{ck}}, \Omega^1$.

Theorem 6.3 (Fundamental theorem) *If* $b : \texttt{nat} \mid \Omega \vdash p : \uparrow\mathsf{C}_{\mathtt{unit}}$, *then* $b : \texttt{nat} \mid \Omega \Vdash p : \uparrow\mathsf{C}_{\mathtt{unit}}$.

Proof: The proof is by induction on the static typing derivation for p and considering the last step in the derivation.

Case 1. Suppose that $p = \mathsf{Ckpt}[\mathsf{aID}, \rho, \mu, \omega](c); p'$ such that T-P-Ckpt is the last step of the derivation.

$$\frac{\begin{array}{c}\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho; \mu; \omega) \\ \mathtt{Sig} = \{\mathsf{aID}(c) \mid b \geq 0 : \texttt{nat} \mid \Omega'; \Sigma \vdash c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''\} \\ \mathsf{aID}(c) \mid b \geq 0 : \texttt{nat} \mid \Omega';\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \\ b : \texttt{nat} \mid \Omega \vdash p' : \uparrow\mathsf{C}_{\mathtt{unit}} \qquad \Omega'' \restriction \{\texttt{MFstWt}\} = \emptyset\end{array}}{b : \texttt{nat} \mid \Omega \vdash \mathsf{Ckpt}[\mathsf{aID}, \rho, \mu, \omega](c); p' : \uparrow\mathsf{C}_{\mathtt{unit}}}\ \text{(T-P-Ckpt)}$$

By inversion, we know that

(1) $\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho; \mu; \omega)$

(2) $\mathtt{Sig} = \{\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''\}$

(3) $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega';\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$

(4) $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

(5) $\Omega'' \restriction \{\mathtt{MFstWt}\} = \emptyset$

By (1) and the definition of $\mathsf{InitWorld}_t$, we have that $\Omega' = \Omega'_1, \Sigma_{\mathsf{ck}}$. Observe that the inductive hypothesis asserts that $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$ implies $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$. By applying the inductive hypothesis to (4), we learn that

$$b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}.$$

To complete the proof, we need to establish

$$\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}.$$

By definition of logical relation, this is equivalent to showing that c is related to itself in the term interpretation for arbitrary n_0, m_0, γ_0, N_0, and V_0 where $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$, and hence, $\mathsf{N}_0 = \mathsf{N}'_0, \mathsf{V}_{0\mathsf{ck}}$ and $\mathit{range}(\gamma_0) = \mathit{dom}(\mathsf{N}_0)$. By assumption, p does not contain any worlds W, so it follows that c does not contain any worlds W. Therefore, it follows by definition 26 that $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ and $\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ are well-formed.

Additionally, by (1), we know that $\Omega' = \Sigma$ and hence $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega';\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$ has the subset property (by definition 17). It then follows by $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$ that $\mathsf{N}_0^0 \subseteq \mathsf{V}_0$ where $\mathsf{N}_0 = \mathsf{N}_{0\,\mathsf{ck}}^0, \mathsf{N}_0^1$.

We need to show that $\forall n_0$:

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$$

Observe that $\mathsf{N}_0 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_0 \setminus \{\mathtt{MFstWt}\}$ vacuously. Additionally, observe that $\mathsf{N}_0 = \mathsf{N}_0 \setminus \{\mathtt{Wtn}\}$ by the definition of $\mathsf{InitWorld}_t$ which states that Ω' contains no $\mathtt{Wtn}$ variables, and $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$ establishes that N_0 contains no $\mathtt{Wtn}$ variables due to well-formedness.

To this end, we instead show a generalized version holds. That is, $\forall n_0$:

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$$

where $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$, $\mathit{range}(\gamma_0) = \mathsf{N}_1$, and $\mathsf{N}_1 = \mathsf{N}_1 \setminus \{\mathtt{Wtn}\}$.

The proof proceeds by induction on m_0:

Base case ($m_0 = 0$). When $m_0 = 0$, the proof is trivial and the desired result follows immediately by the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$:

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{0}$$

Inductive case ($m_0 = k + 1$ ($\exists k$)). If $m_0 = k + 1$, we need to show that

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

such that

(i) $\exists.(\gamma_1 \mid \mathsf{aID(c)} \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_1 \mid c_1)$ such that $\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_1 \mid \mathsf{aID(c)} \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_1 \mid c_1$ AND

(ii) $\exists.(\gamma_2 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2 \mid c_2)$ such that $\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_2 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2 \mid c_2$ AND

(iii) $(\gamma_1 \mid \mathsf{aID(c)} \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2 \mid c_2) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$,

By the progress and preservation for commands applied to (2) and (3), we know that the first configuration

$$\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c$$

can take multiple steps until it becomes a value configuration that continues to be well-typed. Observe that in the mode $\mathsf{aID}(c)$, $\mathsf{N}_1 \backslash \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}_1' \backslash \{\mathtt{MFstWt}, \mathtt{Wtn}\}$. By application of Lemma 13 to $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'$; $\Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$ and $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_1 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$, observe that (i) holds. Additionally, $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}_1' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$ and $dom(\mathsf{N}_1) = dom(\mathsf{N}_1')$.

The proof proceeds in two subcases, depending on the value of n_1':

Subcase $n_1' = 0$. To show (ii), we pick the post-step such that it holds vacuously, i.e., $\gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c$ in 0 steps. We then show (iii) for the post step:

$$(\gamma_1' \mid \mathsf{aID(c)} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

By the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$, and because $n_1' = 0$, this is equivalent to showing

$$(\gamma_1' \mid \mathsf{aID(c)} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \downarrow \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k$$

At this step we show the above relation holds by its definition:

(vi) $\mathtt{Pw0ff}(\gamma_1', \mathsf{aID}(c), \mathsf{N}_1', \mathsf{V}_1') = \gamma_1'' \mid \emptyset$ AND

(vii) $(\gamma_1'' \mid \mathsf{aID}(c) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$

To show (vi), we need to show that $range(\gamma_1'') = dom(\mathsf{N}_1')$ where γ_1'' is the largest restriction of γ_1' such that this condition holds. Obsesrve that the desired result

follows immediately by the assumptions $\gamma_0 \subseteq \gamma_1'$ and $range(\gamma_0) = dom(\mathsf{N}_1)$, where $\gamma_1'' = \gamma_0$, and also $dom(\mathsf{N}_1) = dom(\mathsf{N}_1')$ (from above).

Hence, we need to show

$$(\gamma_0 \mid \mathsf{aID}(c) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

By definition of the value relation at the type $\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing the following:

$$\forall n_1' > 0.(\gamma_0 \mid \mathsf{aID}(c) \mid n_1' \mid \mathsf{N}_1' \mid\uparrow c_1', \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

Fix an arbitrary n_1. We need to show that

$$(\gamma_0 \mid \mathsf{aID}(c) \mid n_1 \mid \mathsf{N}_1' \mid\uparrow c_1', \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

By the definition of value relation at the type $\uparrow \mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing

(viii) $\mathtt{Restore}(\gamma_0 \mid \mathsf{aID}(c) \mid \mathsf{N}_1' \mid c_1') = \mathsf{N}_1' \mid \mathsf{V}_0' \mid c_0'$ (for some V_0', c_0') AND

(ix) $(\gamma_0 \mid \mathsf{aID}(c) \mid n_1 \mid \mathsf{N}_1' \mid \mathsf{V}_0' \mid c_0', \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^k$

To show (ix), note that by the definition of `Restore`, we have that $\mathtt{Restore}(\gamma_0 \mid \mathsf{aID}(c) \mid \mathsf{N}_1' \mid c_1') = \mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} \mid \mathsf{V}_0' \mid c$ where $\mathsf{N}_1' = \mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{Wtn}}$. In particular, we need to show that

$$(\gamma_0 \mid \mathsf{aID}(c) \mid n_1 \mid \mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} \mid \mathsf{V}_0' \mid c, \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^k.$$

Additionally, observe that the nonvolatile memory $\mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}}$ contains no `Wtn`s (i.e. $\mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} = \mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} \setminus \{\mathtt{Wtn}\}$). Therefore, it follows that:

$$\begin{aligned}
&\mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{MFstWt}} \setminus \{\mathtt{MFstWt}\} \\
&= \mathsf{N}^2_{\mathtt{RD,MFstWt}}, \mathsf{V}'_{0\mathtt{ck}}, \mathsf{N}^3_{\mathtt{Wtn}} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} \\
&= \mathsf{N}_1' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} \\
&= \mathsf{N}_1 \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} \\
&= \mathsf{N}_1 \setminus \{\mathtt{MFstWt}\} \\
&= \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}
\end{aligned}$$

Each step holds by an established identity, and the second to last step holds by the assumption $\mathsf{N}_1 = \mathsf{N}_1 \setminus \{\texttt{Wtn}\}$, and the last step holds by the assumption $\mathsf{N}_1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2 \setminus \{\texttt{MFstWt}\}$. To set up the inductive step, note that $\mathsf{N}^2_{\texttt{RD,MFstWt}}, \mathsf{V}'_{0\,\texttt{ck}}, \mathsf{N}^3_{\texttt{MFstWt}}$ contains no $\texttt{Wtn}s$. Additionally, observe that

$$\begin{aligned} dom(\mathsf{N}^2_{\texttt{RD,MFstWt}}, \mathsf{V}'_{0\,\texttt{ck}}, \mathsf{N}^3_{\texttt{MFstWt}}) &= dom(\mathsf{N}^2_{\texttt{RD,MFstWt}}, \mathsf{V}'_{0\,\texttt{ck}}, \mathsf{N}^3_{\texttt{Wtn}}) \\ &= dom(\mathsf{N}'_1) \\ &= dom(\mathsf{N}_1) \\ &= range(\gamma_0) \end{aligned}$$

The desired result follows directly by the inductive hypothesis. Propagating up the cascade, we learn that since n_1 was arbitrary, the value relation holds for all n_1. In summary, we have just shown that

$$(\gamma_0 \mid \mathsf{aID}(c) \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+1}$$

where $\mathsf{N}_1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}_2 \setminus \{\texttt{MFstWt}\}$, $range(\gamma_0) = \mathsf{N}_1$, and $\mathsf{N}_1 = \mathsf{N}_1 \setminus \{\texttt{Wtn}\}$.

Subcase $n'_1 > 0$**.** Since $n'_1 > 0$ and $\gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1$ is a value, observe that $c'_1 = \mathsf{skip}$. It follows by Lemma 14 that

$$\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_1 \mid c'_1$$

where $\mathsf{N}'_1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\texttt{MFstWt}\}$.
Now we want to show that

$$(\gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1, \gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_1 \mid c'_1) \in \mathcal{V}[\![\downarrow\uparrow \mathsf{C}_{\texttt{unit}}]\!]^{k+1}$$

By the value interpretation at type $\mathsf{C}_{\texttt{unit}}$ and $n'_1 > 0$, we need to show that

$$(\gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1, \gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_1 \mid c'_1) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^{k}$$

By definition of the value interpretation at the type $\downarrow\uparrow \texttt{unit}$, this is equivalent to showing

(x) $\texttt{Commit}(\gamma'_1, \mathsf{aID}(c), \mathsf{N}'_1, \mathsf{V}'_1) = \gamma_1 \mid \mathsf{N}''_{1\,\texttt{ck}}, \mathsf{V}''_1$ AND

(xi) $\texttt{Commit}(\gamma'_1, \mathsf{aID}(c), \mathsf{N}'_2, \mathsf{V}'_1) = \gamma_2 \mid \mathsf{N}''_{2\,\texttt{ck}}, \mathsf{V}''_2$ AND

(xii) $(\gamma_1 \mid \mathsf{aID}(c) \mid n'_1 \mid \mathsf{N}''_1, \mathsf{V}''_1 \mid \mathsf{skip}, \gamma_2 \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N}''_2, \mathsf{V}''_2 \mid \mathsf{skip})$
$\in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^k$

By (x) and (xi), we observe that $\gamma_1 = \gamma_2$. Additionally, it follows by $\mathsf{N}'_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\mathtt{MFstWt}\}$ and the definition of `Commit` (which asserts that $\mathsf{N}'_1 = \mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}^3_{\mathsf{ck}}$, $\mathsf{N}'_2 = \mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}^4_{\mathsf{ck}}$, $dom(\mathsf{N}^3_{\mathsf{ck}}) = dom(\mathsf{V}''_1)$, $dom(\mathsf{N}^4_{\mathsf{ck}}) = dom(\mathsf{V}''_2)$) that

$$\begin{aligned}
&\mathsf{N}'_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\mathtt{MFstWt}\} \\
&\Rightarrow \mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}^3_{\mathsf{ck}} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}^4_{\mathsf{ck}} \setminus \{\mathtt{MFstWt}\} \\
&\Rightarrow \mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_2 \setminus \{\mathtt{MFstWt}\}
\end{aligned}$$

To complete the proof, we need to show that $\mathsf{N}''_1, \mathsf{V}''_1 = \mathsf{N}''_2, \mathsf{V}''_2$. To this end, we show that $\mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_1$ and $\mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_2 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_2$, i.e., that the final memories have no `MFstWt`.
It follows by lemma 5 (preservation for commands) applied to

- $\gamma_0 \mid \mathsf{aID}(\mathsf{c}) \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c \rightarrow^* \gamma'_1 \mid \mathsf{aID}(\mathsf{c}) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1$,
- $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega';\ \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega''$, and
- $\vdash^{\mathsf{aID}(c)}_{\gamma_0} \mathsf{N}_1 \mid \mathsf{V}_0 : \Omega' \mid \Sigma$

that

$$\mathtt{Md} \mid b : nat \mid \Omega''; \Sigma^1 \vdash \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega''$$

where $\vdash \mathsf{N}'_1 \mid \mathsf{V}'_1 : \Omega'' \mid \Sigma^1$. Then, it follows by $\mathsf{N}'_1 = \mathsf{N}''_1, \mathsf{N}^3_{\mathsf{ck}}$, $\mathsf{N}'_2 = \mathsf{N}''_2, \mathsf{N}^4_{\mathsf{ck}}$, $dom(\mathsf{N}^3_{\mathsf{ck}}) = dom(\mathsf{V}''_1)$, $dom(\mathsf{N}^4_{\mathsf{ck}}) = dom(\mathsf{V}''_2)$, and the unicity of typing that

$$\vdash \mathsf{N}''_1, \mathsf{V}''_1 \mid \mathsf{V}'_1 : \Omega'' \mid \Sigma^1$$

Since $\Omega'' \restriction \{\mathtt{MFstWt}\} = \emptyset$ (i.e. Ω'' has no `MFstWt`s), it follows that

$$\mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_1.$$

By similar reasoning, we can show that

$$\mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_2 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_2.$$

Hence, $\mathsf{N}''_{1\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_1 = \mathsf{N}''_{2\,\mathtt{RD,Wtn,MFstWt}}, \mathsf{V}''_2$. Therefore, $\mathsf{N}''_{1\,\mathsf{ck}}, \mathsf{V}''_1 = \mathsf{N}''_{2\,\mathsf{ck}}, \mathsf{V}''_2$. Therefore, we can use the value interpretation at type $\uparrow \mathtt{unit}$ to prove (xii):

$$(\gamma_1 \mid \mathsf{aID}(c) \mid n'_1 \mid \mathsf{N}''_{1\,\mathsf{ck}}, \mathsf{V}''_1 \mid \mathsf{skip}, \gamma_2 \mid \mathsf{aID}(c) \mid \infty \mid \mathsf{N}''_{2\,\mathsf{ck}}, \mathsf{V}''_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \mathtt{unit}]\!]^k$$

which holds by definition of logical relation. This is the last piece we needed in order to prove the desired result:

$$(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

In general, we have that $(\gamma_0 \mid \mathsf{aID(c)} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{aID(c)} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$ where $\vdash_{\gamma_0}^{\mathsf{aID}(c)} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$. Since $n_0, m_0 \geq 0$, γ_0, N_0, and V_0 were arbitrarily chosen, this result holds for all $n, m \geq 0, \gamma, \mathsf{N}, \mathsf{V}$. Therefore, it follows by definition of logical relation that $\mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$. Finally, the desired result follows by application of P-CKPT-SEMANTIC.

$$\frac{\begin{array}{c}\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho; \mu; \omega) \\ \mathsf{aID}(c) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}\end{array}}{b : \mathtt{nat} \mid \Omega \Vdash \mathsf{Ckpt}[\mathsf{aID}, \rho, \mu, \omega](c); p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-CKPT-SEMANTIC)}$$

Case 2. Suppose that $p = c; p'$ where T-P-SEQ is the last step of the derivation.

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \qquad b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-SEQ)}$$

By inversion, we know that

(1) $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$

(2) $b : \mathtt{nat} \mid \Omega \vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

From (1), we know that c is well-typed for static context Ω and $\Sigma = \cdot$. By applying the inductive hypothesis to (2), we learn that $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$. To complete the proof, we need to show that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$. By the definition of logical relation, this is equivalent to establishing that c is related to itself in the term interpretation for arbitrary n_0, m_0, γ_0, N_0, and V_0 where $\vdash_{\gamma_0}^{\mathsf{jit}} \mathsf{N}_0 \mid \mathsf{V}_0 : \Omega \mid \Sigma$. By assumption, the program p, and by extension the command c, does not contain any worlds W, so it follows by definition 26 that $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ and $\gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c$ are well-formed configurations. We need to show that $\forall c.\, \mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$ and $\forall \mathsf{N}^1, \mathsf{N}^2, \mathsf{V}_0, \gamma_0, n_0.\ \vdash_{\gamma_0}^{\mathsf{jit}} \mathsf{N}^1 \mid \mathsf{V}_0 : \Omega \mid \Sigma$ and $\vdash_{\gamma_0}^{\mathsf{jit}} \mathsf{N}^2 \mid \mathsf{V}_0 : \Omega \mid \Sigma$ where $\mathsf{N}^1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}^2 \setminus \{\mathtt{MFstWt}\}$:

$$(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$$

The proof proceeds by induction on m_0:

Base case ($m_0 = 0$). When $m_0 = 0$, the proof is trivial and the desired result follows immediately by the definition of logical relation.

Inductive case ($m_0 = k + 1\ (\exists k)$) Consider the case where $m_0 = k + 1$.

We need to show that

$$(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

By the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing

(i) $\exists.\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1$ such that $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c \to^* \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1$

(ii) $\exists.\gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2$ such that $\gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{V}_0 \mid c \to^* \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2$

(iii) $(\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$

By the progress and preservation theorems, we know that the first configuration can take multiple steps until it becomes a value configuration that continues to be well-typed. We proceed by induction on n_0:

Base case. If $n_0 = 0$, then the configuration is a value.

Inductive case. Suppose that $n_0 = n_0' + 1$ ($\exists n_0'$). Since $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}^1 \mid \mathsf{V}_0 : \Omega \mid \cdot$ and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_\emptyset c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$, it follows by progress for commands that either $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c$ is a value or $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c$ is not a value, in which case $\exists\, \gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$ such that

$$\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c \to \gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$$

where $\gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1''$ is well-formed, $\vdash^{\mathsf{jit}}_{\gamma_1''} \mathsf{N}_1'' \mid \mathsf{V}_1'' : \Omega'' \mid \Sigma''$ and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega''; \Sigma'' \vdash_\emptyset c_1'' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$ (by theorem 5 because $\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c$ is well-formed and $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}^1 \mid \mathsf{V}_0 : \Omega \mid \cdot$).

By the inductive hypothesis, $\gamma_1'' \mid \mathsf{jit} \mid n_1'' \mid \mathsf{N}_1'' \mid \mathsf{V}_1'' \mid c_1'' \to^* \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ where $\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ is well-formed and a value, $\vdash^{\mathsf{jit}}_{\gamma_1'} \mathsf{N}_1' \mid \mathsf{V}_1' : \Omega''' \mid \Sigma'''$, and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega'''; \Sigma''' \vdash_\emptyset c_1' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$. By head expansion, we establish that

$$\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c \to^* \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

Analogously, we step the second configuration until c_1' with the exact same steps as in the first configuration by progress and preservation for commands:

$$\gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c \to^* \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

We have just shown (i) and (ii). The proof of (iii) corresponds to step (2) in the diagram. Proving that the configurations are in the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$ is equivalent to showing

(iv) $n_1' = 0 \wedge (\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1')$
$\in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k$ OR

(v) $n_1' > 0 \ \wedge \ (\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]^k$

The proof proceeds in two subcases depending on the value of n_0':

Case $n_1' = 0$**.** If $n_1' = 0$, then we need to show that

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^k$$

This proof corresponds to point (3) in the diagram. To show that the two configurations are in the value interpretation at type $\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})$, we need to show:

(vi) $\mathtt{PwOff}(\gamma_1', \mathsf{jit}, \mathsf{N}_1', \mathsf{V}_1') = \gamma_1' \mid \mathsf{V}_1'$ AND

(vii) $(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$

To show (vii), we need to show that the configurations are in the value interpretation of $\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}$, which corresponds to step (4) in the diagram:

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

This is equivalent to proving that

$$\forall n > 0.(\gamma_1' \mid \mathsf{jit} \mid n \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

This step corresponds to point (5) in the diagram. Fix an arbitrary n'. We need to show that

$$(\gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{V}_1', \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(n_1' > 0, \uparrow c_1'), \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^k$$

According to the value interpretation at type $\uparrow \mathsf{C}_{\mathtt{unit}}$, this is equivalent to showing:

(viii) $\mathtt{Restore}(\gamma_1', \mathsf{jit}, (\mathsf{V}_1', \mathsf{N}_1'), c_1') = \mathsf{N}' \mid \mathsf{N}'' \mid c_1'$ where $\mathsf{V}_1', \mathsf{N}_1' = \mathsf{N}', \mathsf{N}_{\mathsf{ck}}''$ AND

(ix) $(\gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{N}_{\mathsf{ck}}'' \mid c_1', \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^k$

From (viii), we have that $\mathsf{V}_1' = \mathsf{N}_{\mathsf{ck}}''$ and $\mathsf{N}' = \mathsf{N}_1'$. Recalling from above that $\vdash^{\mathsf{jit}}_{\gamma_1'} \mathsf{N}_1' \mid \mathsf{V}_1' : \Omega''' \mid \Sigma'''$ and $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega'''; \Sigma''' \vdash_\emptyset c_1' : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$, we can apply the inductive hypothesis to prove (ix). Therefore, we have established that

$$(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

Case $n_0' > 0$**.** If $n_0' > 0$, then we need to show that

$$(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_1' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_1' \mid c_1') \in \mathcal{V}[\![\downarrow\uparrow \mathtt{unit}]\!]^k$$

where $\mathsf{N}_1' \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2' \setminus \{\mathtt{MFstWt}\}$.

From the value interpretation at type $\downarrow\uparrow \mathtt{unit}$, we have

(x) $\texttt{Commit}(\gamma_1', \mathsf{jit}, \mathsf{N}_1', \mathsf{V}_1') = \gamma_1 \mid \mathsf{N}_{\mathsf{ck}}^1, \mathsf{N}_{\mathsf{ck}}^2$ where $\mathsf{N}_1' = \mathsf{N}_{\mathsf{RD}}^1, \mathsf{N}_{\mathsf{ck}}^2$

(xi) $\texttt{Commit}(\gamma_1', \mathsf{jit}, \mathsf{N}_2', \mathsf{V}_1') = \gamma_2 \mid \mathsf{N}_{\mathsf{ck}}^3, \mathsf{N}_{\mathsf{ck}}^4$ where $\mathsf{N}_2' = \mathsf{N}_{\mathsf{RD}}^3, \mathsf{N}_{\mathsf{ck}}^4$

(xii) $(\gamma_1 \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_{\mathsf{ck}}^1, \mathsf{N}_{\mathsf{ck}}^2 \mid \mathsf{skip}, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_{\mathsf{ck}}^3, \mathsf{N}_{\mathsf{ck}}^4 \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^k$

From (x) and (xi), we have that $\gamma_1 = \gamma_2$ and $\mathsf{N}_{\mathsf{ck}}^1, \mathsf{N}_{\mathsf{ck}}^2 = \mathsf{N}_{\mathsf{ck}}^3, \mathsf{N}_{\mathsf{ck}}^4$. Therefore, the desired result follows by the value interpretation at type $\downarrow\uparrow \texttt{unit}$:

$$(\gamma_1 \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_{\mathsf{ck}}^1, \mathsf{N}_{\mathsf{ck}}^2 \mid \mathsf{skip}, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_{\mathsf{ck}}^3, \mathsf{N}_{\mathsf{ck}}^4 \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^k$$

Therefore, we have shown that $(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}^1 \mid \mathsf{V}_0 \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{V}_0 \mid c) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{m_0}$ where $\vdash_{\gamma_0}^{\mathsf{jit}} \mathsf{N}^1 \mid \mathsf{V}_0 : \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$ and $\vdash_{\gamma_0}^{\mathsf{jit}} \mathsf{N}^2 \mid \mathsf{V}_0 : \Omega'', \Sigma_{\mathsf{ck}} \mid \Sigma$ and $\mathsf{N}^1 \setminus \{\texttt{MFstWt}\} = \mathsf{N}^2 \setminus \{\texttt{MFstWt}\}$. Since $n_0, m_0 \geq 0$, γ_0, N^1, N^2, V_0 were arbitrary, this result holds for all $n, m \geq 0$, γ, N_1, N_2, V. Therefore, it follows by the definition of logical relation that $\mathsf{jit} \mid b \geq 0 : \texttt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\texttt{unit}}$. Finally, the desired result follows by application of P-SEQ-SEMANTIC.

$$\frac{\mathsf{jit} \mid b \geq 0 : \texttt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\texttt{unit}} \qquad b : \texttt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\texttt{unit}}}{b : \texttt{nat} \mid \Omega \Vdash c; p' : \uparrow \mathsf{C}_{\texttt{unit}}} \text{ (P-SEQ-SEMANTIC)}$$

□

B.7 Adequacy

Definition 6.4 (Idempotency) *A triple of a program p, nonvolatile memory* N*, and a mapping* γ *is idempotent, if every intermittent execution of the program can be simulated by a continuous execution of it: for all* $n, n', \chi_1, \chi_1', \mathsf{N}', p'$*, if* $[\chi_1 \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid\ p \Rightarrow [\chi_1' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'$*, then* $[\chi_2 \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N} \mid\ p \Rightarrow [\chi_2 \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N}' \mid p'$.

Theorem 6.5 (Adequacy) *Consider* $b : \texttt{nat} \mid \Omega \Vdash p : \mathsf{C}_{\texttt{unit}}$*, a nonvolatile memory* N*, and a map* γ *such that* $\vdash_{\gamma}^{\mathsf{jit}} \mathsf{N} \mid \cdot : \Omega \mid \cdot$*. The triple of p,* N*, and* γ *is idempotent.*

Proof: The proof is by cases according to the execution mode.

Stepping a JIT block. Consider a program of form $[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid\ c; p'$ that can take a step using the D-P-SEQ rule to $[\chi'' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'$. By inversion on the D-P-SEQ rule,

$$\frac{\begin{array}{c} n > 0 \\ n' > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid\ \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes\ \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid\ c; p' \Rightarrow_p [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid p'} \text{ (D-P-SEQ)}$$

Suppose that the command c is successfully executed to completion with possibly m power failures and $m + 1$ tries along the way:

- $n > 0$
- $n' > 0$
- $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$

Our goal is to show that we can run the configuration with ∞, an infinite level of energy, as:

$[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2 \mid \mathsf{skip}$.

Figure B.2 shows the main idea of the proof. Here we provide more details. We first need to establish that the configuration with n level of energy is related to itself when provided with an infinite energy level ∞ for every index, including $m+1$ (point (1) in Figure 3.23).

By inversion on T-P-SEQ-SEMANTIC and the assumption $b : \mathtt{nat} \mid \Omega \Vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}$,

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash c; p' : \uparrow \mathsf{C}_{\mathtt{unit}}}\ \text{(P-SEQ-SEMANTIC)}$$

we learn that

(i) $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \Vdash c \leq c : \mathsf{C}_{\mathtt{unit}}$

(ii) $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

By the definition of logical relation applied to (i), we have that $\forall n_0, m_0 \geq 0. \forall \gamma_0, \mathsf{N}_0^1, \mathsf{N}_0^2$ s.t. $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}_0^1 \mid \cdot : \Omega \mid \cdot$ and $\vdash^{\mathsf{jit}}_{\gamma_0} \mathsf{N}_0^2 \mid \cdot : \Omega \mid \cdot$ and $\mathsf{N}_0^1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_0^2 \setminus \{\mathtt{MFstWt}\}$. $(\gamma_0 \mid \mathsf{jit} \mid n_0 \mid \mathsf{N}_0^1 \mid \cdot \mid c, \gamma_0 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_0^2 \mid \cdot \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_0}$.

Instantiating m_0 to be the number of tries $m+1$ (i.e. $m_0 = m+1$), we have that

$$(\gamma \mid \mathsf{jit} \mid n \mid \mathsf{N}^1 \mid \cdot \mid c, \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \cdot \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$$

To prove the desired result, we use induction to prove the following generalized statement:

- $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in m crashes and
- $(\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$

then for any energy stream χ^0, we have $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2'' \mid \mathsf{V}_2'' \mid \mathsf{skip}$ and $\mathsf{N}_2'' = \mathsf{N}'$.

The proof proceeds by induction on the number of crashes (m):

Base case: $m = 0$ (***# tries*** $= 1$)**.** It follows by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$ that

1. $\exists(\gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1')$ s.t. $\gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ AND
2. $\exists(\gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2')$ s.t. $\gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$ AND
3. $(\gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^1$

Since $m = 0$, there are no crashes in (1). So, $n' > 0$ and the first configuration steps to completion via D-STEP where $\mathsf{N}_1' = \mathsf{N}'$ and $\mathsf{V}_1' = \mathsf{V}'$ and $\gamma_1'' = \gamma'$:

$$[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \mathsf{skip}$$

By (2) we know that:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (3) and $n' > 0$, we get that the post steps are related by the value interpretation at type $\downarrow\uparrow \texttt{unit}$. This means that $c_2' = \mathsf{skip}$ and we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}$$

and

$$(\gamma_1'' \mid \mathsf{jit} \mid n' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid \mathsf{skip}, \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^0$$

It then follows by the value relation at type $\downarrow\uparrow \texttt{unit}$ that

1. $\texttt{Commit}(\gamma_1'', \mathsf{jit}, \mathsf{N}_1', \mathsf{V}_1') = \gamma^1 \mid \mathsf{N}^1$ where $\mathsf{range}(\gamma^1) = \mathsf{dom}(\mathsf{N}^1)$ and $\gamma^1 \subseteq \gamma_1''$ and $\mathsf{N}_1' = \mathsf{N}^3_{\mathsf{RD}}, \mathsf{N}^4_{\mathsf{ck}}$
2. $\texttt{Commit}(\gamma_2'', \mathsf{jit}, \mathsf{N}_2', \mathsf{V}_2') = \gamma^2 \mid \mathsf{N}^2$ where $\mathsf{range}(\gamma^2) = \mathsf{dom}(\mathsf{N}^2)$ and $\gamma^2 \subseteq \gamma_2''$ and $\mathsf{N}_2' = \mathsf{N}^5_{\mathsf{RD}}, \mathsf{N}^6_{\mathsf{ck}}$
3. $(\gamma^1 \mid \mathsf{jit} \mid n' \mid \mathsf{N}^1 \mid \mathsf{skip}, \gamma^2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}^2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^0$

By the definition of commit in the jit mode and (1) and (2), we have $\mathsf{N}^1 = \mathsf{N}^3_{\mathsf{ck}}, \mathsf{N}^4_{\mathsf{ck}}$ and $\mathsf{N}^2 = \mathsf{N}^5_{\mathsf{ck}}, \mathsf{N}^6_{\mathsf{ck}}$. Thus, it follows by the value interpretation at type $\uparrow \texttt{unit}$:

$$\mathsf{N}' = \mathsf{N}_1' = \mathsf{N}_2'$$

Therefore, we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2' \mid \mathsf{skip}$$

which completes the proof for this subcase.

Inductive case: $m = k + 1 (\exists\, k)$ **(*# tries* $= k + 2$).** It follows by the term interpretation at type $\mathsf{C}_{\texttt{unit}}$ that

(iii) $\exists(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1')$ s.t. $\gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$

(iv) $\exists(\gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2')$ s.t. $\gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$

(v) $(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+2}$

From (iii), we step the first configuration until it becomes a value. It follows by the rule D-STEP that

$$[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{jit} \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

We step the second configuration via D-STEP applied to (iv):

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (v) we have:

$$(\gamma_1' \mid \mathsf{jit} \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+2}$$

Since there are $m > 0$ crashes, we know that $n_1' = 0$. By application of D-Crash, we have

$$[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid 0 \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon \# \mathsf{in}(b > 0; \uparrow c_1')$$

By the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$, observe that the stepped configuration continues to be related to the second configuration:

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon \# \mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^{k+1}$$

By D-S-Jit, we have

$$[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon \# \mathsf{in}(b > 0; \uparrow c_1') \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}' \mid \varepsilon \# \mathsf{in}(b > 0; \uparrow c_1')$$

By the value interpretation at type $\downarrow (\mathtt{nat} \rightsquigarrow \mathsf{C}_{\mathtt{unit}})$, the post step configuration remains related to the second configuration:

$$(\gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}' \mid \varepsilon \# \mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

as by definition of `PwOff` in the jit mode, we have $\mathtt{PwOff}(\gamma_1', \mathsf{jit}, \mathsf{N}_1', \mathsf{V}_1') = \gamma_1' \mid \mathsf{V}_1'$.
By D-charge, for some $n'' > 0$, we have $\chi = n'' :: \chi'$:

$$[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid \cdot \mid \mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}' \mid \varepsilon \# \mathsf{in}(b > 0; \uparrow c_1') \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}' \mid\!\uparrow c_1'$$

It follows by the value interpretation at type $\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}$ that the stepped configuration and the second configuration remain related for n'' (by $\forall$ elimination):

$$(\gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}' \mid\!\uparrow c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

By D-restore-Jit, we have

$$[\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}' \mid\!\uparrow c_1' \Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

From above, we get $[\chi \rhd \varepsilon]\otimes\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi' \rhd \varepsilon]\otimes\gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$ and $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$. By the value interpretation at type $\uparrow \mathsf{C}_{\mathtt{unit}}$, these configurations continue to be related:

$$(\gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+1}$$

as $\mathtt{Restore}(\gamma_1', \mathsf{jit}, (\mathsf{N}_1', \mathsf{V}_{1\mathtt{Ck}}'), c_1') = \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$.

Since $[\chi \rhd \varepsilon]\otimes\gamma_1 \mid \mathsf{jit} \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi'' \rhd \varepsilon]\otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in k crashes we know that $[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n'' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in $k-1$ crashes.

By the application of the induction hypothesis we get: $[\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2' \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \mathsf{V}_2'' \mid \mathsf{skip}$, which combined with $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$ gives us the desired result of this subcase:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{jit} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \mathsf{V}_2'' \mid \mathsf{skip}$$

With that established, we can apply the generalized statement on assumptions

$$(\gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c, \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$$

and $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ to get

$$[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2 \mid \mathsf{skip}$$

and apply D-P-Seq rule to complete the proof of this case:

$$\frac{\infty > 0 \qquad \infty > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid \infty \mid \mathsf{N} \mid \cdot \mid c \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{jit} \mid \infty \mid \mathsf{N}' \mid \mathsf{V}_2 \mid \mathsf{skip}}{[\chi \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N} \mid c; p' \Rightarrow_p [\chi \rhd \varepsilon] \otimes \gamma \mid \infty \mid \mathsf{N}' \mid p'} \text{(D-P-SEQ)}$$

Stepping an atomic region. Consider a program of form $[\chi \rhd \varepsilon]\otimes\gamma \mid n \mid \mathsf{N} \mid \mathsf{Ckpt}[\mathsf{aID}; \rho; \mu; \omega](c_0); p'$ that can take a step using the D-P-Seq rule to $[\chi'' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p'$. By inversion on the D-P-Ckpt rule,

$$\frac{\begin{array}{c} n > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; \rho; \mu; \omega; \gamma) = \mathsf{N}_0, V_0 \\ [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{Ckpt}[\mathsf{aID}; \rho; \mu; \omega](c_0); p' \Rightarrow_p [\chi'' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}_1 \mid p'} \text{(D-P-CKPT)}$$

we learn that

- $n > 0$
- $\mathsf{InitWorld}_d(\mathsf{N}; \rho; \mu; \omega; \gamma) = \mathsf{N}_0, V_0$
- $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$
- $n' > 0$
- $\mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}')$

Our goal is to simulate this execution in a continuous setting. In particular, we need to find a continuous execution such that $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid \mathsf{skip}$, where $\mathsf{N}_1 = \mathsf{FinWorld}_d(\mathsf{N}'_2; \mathsf{V}'_2)$. To this end, we invert the assumption $b : \mathtt{nat} \mid \Omega \Vdash \mathsf{Ckpt}[\mathsf{aID}; \rho; \mu; \omega](c_0); p' :\uparrow \mathsf{C}_{\mathtt{unit}}$ via P-Ckpt-semantic,

$$\frac{\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho; \mu; \omega) \qquad \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c_0 \leq c_0 : \mathsf{C}_{\mathtt{unit}} \qquad b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \Vdash \mathsf{Ckpt}[\mathsf{aID}; \rho; \mu; \omega](c_0); p' : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (P-Ckpt-semantic)}$$

we learn that

(i) $\Omega' \mid \Sigma = \mathsf{InitWorld}_t(\Omega; \rho; \mu; \omega)$

(ii) $\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \Vdash c_0 \leq c_0 : \mathsf{C}_{\mathtt{unit}}$

(iii) $b : \mathtt{nat} \mid \Omega \Vdash p' : \uparrow \mathsf{C}_{\mathtt{unit}}$

By definition of logical relation applied to (ii), we have that $\forall n_1, m_1 \geq 0. \forall \gamma, \mathsf{N}, \mathsf{V}$ s.t. $\vdash_\gamma^{\mathsf{aID}(c_0)} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and $(\gamma \mid \mathsf{aID}(c_0) \mid n_1 \mid \mathsf{N} \mid \mathsf{V} \mid c_0, \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N} \mid \mathsf{V} \mid c_0) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m_1}$.

By instantiating the memories accordingly, and the index m_1 with the number of tries $m + 1$ (where m is the number of crashes), we have

$$(\gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$$

To get our result, we first prove the following generalized statement: if

- $[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID}(c_0) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n'_1 \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ in m crashes and
- $(\gamma_1 \mid \mathsf{aID}(c_0) \mid n_1 \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1, \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2) \in \mathcal{E}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{m+1}$

then for all energy streams χ^0, we have $[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}''_2 \mid \mathsf{V}''_2 \mid \mathsf{skip}$, where $\mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') = \mathsf{FinWorld}_d(\mathsf{N}''_2; \mathsf{V}''_2)$

The proof proceeds by induction on the number of crashes:

Base case: $m = 0$ (***# tries*** $= 1$)**.** If $m = 0$, then it follows by the term interpretation at type $\mathsf{C}_{\mathtt{unit}}$,

(1) $\exists(\gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1)$ s.t. $\gamma_1 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1$ AND

(2) $\exists(\gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2)$ s.t. $\gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2$ AND

(3) $(\gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1, \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2) \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^1$

The number of crashes is 0, so $n' > 0$ and the first configuration steps to completion via D-STEP where $\mathsf{N}'_1 = \mathsf{N}'$ and $\mathsf{V}'_1 = \mathsf{V}'$ and $\gamma''_1 = \gamma'$:

$$[\chi \rhd \varepsilon] \otimes \gamma_1 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid \mathsf{skip}$$

Applying D-STEP to (2), we know that:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2$$

and by (3) and $n' > 0$, we get that the post steps are related by the value interpretation at type $\downarrow\uparrow \texttt{unit}$. This means that $c'_2 = \mathsf{skip}$ and we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid \mathsf{skip}$$

and

$$(\gamma''_1 \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid \mathsf{skip}, \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid \mathsf{skip}) \in \mathcal{V}[\![\downarrow\uparrow \texttt{unit}]\!]^0$$

It then follows that

(1) $\texttt{Commit}(\gamma''_1, \mathsf{aID}(c_0), \mathsf{N}'_1, \mathsf{V}'_1) = \gamma' \mid \mathsf{N}'_{\mathtt{ck}}, \mathsf{V}'$ where $\gamma' \subseteq \gamma''_1$, $\mathsf{N}'_1 = \mathsf{N}'_{\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}'_{0_{\mathtt{ck}}}$, $\mathsf{V}'_1 = \mathsf{V}'_0, \mathsf{V}'$, $dom(\mathsf{V}') = dom(\mathsf{N}'_0)$, $range(\gamma') = dom(\mathsf{N}'_1) \cup dom(\mathsf{V}')$.

(2) $\texttt{Commit}(\gamma''_2, \mathsf{aID}(c_0), \mathsf{N}'_2, \mathsf{V}'_2) = \gamma^2 \mid \mathsf{N}^2_{\mathtt{ck}}, \mathsf{V}^2$ where $\gamma^2 \subseteq \gamma''_2$, $\mathsf{N}'_2 = \mathsf{N}^2_{\mathtt{RD,Wtn,MFstWt}}, \mathsf{N}^2_{0_{\mathtt{ck}}}$, $\mathsf{V}'_2 = \mathsf{V}^2_0, \mathsf{V}^2$, $dom(\mathsf{V}^2) = dom(\mathsf{N}^2_0)$, $range(\gamma^2) = dom(\mathsf{N}'_2) \cup dom(\mathsf{V}^2)$.

(3) $(\gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}'_{\mathtt{ck}}, \mathsf{V}' \mid \mathsf{skip}, \gamma^2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}^2_{\mathtt{ck}}, \mathsf{V}^2 \mid \mathsf{skip}) \in \mathcal{V}[\![\uparrow \texttt{unit}]\!]^0$

It follows by the value interpretation at type $\uparrow \texttt{unit}$ that $\mathsf{N}', \mathsf{V}' = \mathsf{N}^2, \mathsf{V}^2$. We observe that the `Commit` function copies the values of the checkpointed volatile memory locations into the nonvolatile memory, removes the `ck` qualifier from these locations, and changes the qualifiers of all other locations in the nonvolatile memory to `ck`. This corresponds to the semantics of the $\mathsf{FinWorld}_d$ function, and hence it follows by definition that $\mathsf{FinWorld}_d(\mathsf{N}'_1; \mathsf{V}'_1) = \mathsf{N}', \mathsf{V}'$ and $\mathsf{FinWorld}_d(\mathsf{N}'_2; \mathsf{V}'_2) = \mathsf{N}^2, \mathsf{V}^2$.

Therefore, we have

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma''_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid \mathsf{skip}$$

where $\mathsf{FinWorld}_d(\mathsf{N}'_1; \mathsf{V}'_1) = \mathsf{FinWorld}_d(\mathsf{N}'_2; \mathsf{V}'_2)$, and the proof of this subcase is complete.

Inductive case: $m = k + 1 (\exists\, k)$ **(*# tries* $= k + 2$).** By the term interpretation at type $\mathsf{C}_{\texttt{unit}}$, we have

(i) $\exists\, \gamma'_1 \mid \mathsf{aID}(c_0) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1$ s.t. $\gamma_1 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \rightarrow^* \gamma'_1 \mid \mathsf{aID}(c_0) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1$

(ii) $\exists\, \gamma'_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2$ s.t. $\gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \rightarrow^* \gamma'_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2$.

(iii) $(\gamma'_1 \mid \mathsf{aID}(c_0) \mid n'_1 \mid \mathsf{N}'_1 \mid \mathsf{V}'_1 \mid c'_1, \gamma'_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}'_2 \mid c'_2) \in \mathcal{V}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+2}$

From (i), we step the first configuration until it becomes a value. It follows by the rule D-STEP that

$$[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_1 \mid \mathsf{V}_1 \mid c_1 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1'$$

Since there are $m > 0$ crashes, we know that $n_1' = 0$. By (ii), we step the second configuration via D-STEP:

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

and by (iii), we have

$$(\gamma_1' \mid \mathsf{aID}(c_0) \mid n_1' \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1', \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\mathsf{C}_{\mathtt{unit}}]\!]^{k+2}$$

By application of D-CRASH, we have

$$\begin{aligned}&[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid 0 \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid c_1' \\ &\Rightarrow [\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1')\end{aligned}$$

By the value interpretation at type $\mathsf{C}_{\mathtt{unit}}$, observe that the stepped configuration continues to be related to the second configuration:

$$\begin{aligned}(\gamma_1' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})]\!]^{k+1}\end{aligned}$$

By D-S-AID, for $\gamma^1 \subseteq \gamma_1'$ such that $\mathit{range}(\gamma^1) = \mathit{dom}(\mathsf{N}_1')$, we have:

$$\begin{aligned}&[\chi \rhd \varepsilon] \otimes \gamma_1' \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \mathsf{V}_1' \mid\!\downarrow \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1') \\ &\Rightarrow [\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1')\end{aligned}$$

Note that the semantics of D-S-AID match the semantics of the `Pw0ff` function as it drops the volatile memory locations V_1' just as the `Pw0ff` function does not checkpoint any volatile memory locations, i.e. $\mathtt{Pw0ff}(\gamma_1', \mathsf{aID}(c_0), \mathsf{N}_1', \mathsf{V}_1') = \gamma^1 \mid \emptyset$. By the value interpretation at type $\downarrow (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}})$, and the definition of `Pw0ff` in the atomic case, we have $\mathtt{Pw0ff}(\gamma_1', \mathsf{aID}(c_0), \mathsf{N}_1', \mathsf{V}_1') = \gamma^1 \mid \emptyset$, and thus the stepped configuration continues to be related to the second configuration:

$$\begin{aligned}(\gamma^1 \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1'), \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \\ \in \mathcal{V}[\![\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{unit}}]\!]^{k+1}\end{aligned}$$

By stepping the first configuration according to D-CHARGE, we have for some $n'' > 0$ such that $\chi = n'' :: \chi'$:

$$\begin{aligned}&[\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid \cdot \mid \mathsf{N}_1' \mid \varepsilon\#\mathsf{in}(b > 0; \uparrow c_1') \\ &\Rightarrow [\chi' \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid\!\uparrow c_1'\end{aligned}$$

By the value interpretation at type $\texttt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\texttt{unit}}$, observe that the stepped configuration remains related to the second configuration for $n'' > 0$:

$$(\gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid \uparrow c_1', \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{V}[\![\uparrow \mathsf{C}_{\texttt{unit}}]\!]^{k+1}$$

Stepping the first configuration via D-RESTORE-AID, we have

$$[\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid \uparrow c_1' \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}^3_{\texttt{MFstWt}} \mid \cdot \mid c_0$$

where $\mathsf{N}_1' = \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}^3_{\texttt{Wtn}}$.
By the value interpretation at type $\uparrow \mathsf{C}_{\texttt{unit}}$, the first and second configurations remain related:

$$(\gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}^3_{\texttt{MFstWt}} \mid \cdot \mid c_0, \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2') \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{k+1}$$

since

- $\texttt{Restore}(\gamma^1, \mathsf{aID}(c_0), \mathsf{N}_1', c_1') = \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}^3_{\texttt{MFstWt}} \mid \cdot \mid c_0$ and
- $\mathsf{N}_1' = \mathsf{N}'_{\texttt{RD,MFstWt}}, \mathsf{N}''_{\texttt{ck}}, \mathsf{N}^3_{\texttt{Wtn}}$

By assumption,

$$[\chi \rhd \varepsilon] \otimes \gamma^1 \mid \mathsf{aID}(c_0) \mid n'' \mid \mathsf{N}_1' \mid \cdot \mid c_0 \Rightarrow^* [\chi'' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$$

in k crashes.
By induction hypothesis, we get

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2' \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}_2'' \mid \mathsf{V}_2'' \mid \mathsf{skip}$$

such that $\mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') = \mathsf{FinWorld}_d(\mathsf{N}_2''; \mathsf{V}_2'')$. This combined with

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}_2' \mid c_2'$$

gives us

$$[\chi^0 \rhd \varepsilon] \otimes \gamma_2 \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}_2 \mid c_2 \Rightarrow^* [\chi^0 \rhd \varepsilon] \otimes \gamma_2'' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}_2'' \mid \mathsf{V}_2'' \mid \mathsf{skip}$$

which completes the proof of this subcase.

With that established, we can apply the generalized statement on assumptions

$$(\gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0, \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0) \in \mathcal{E}[\![\mathsf{C}_{\texttt{unit}}]\!]^{m+1}$$

and $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{skip}$ to get $[\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}_0 \mid \mathsf{V}_0 \mid c_0 \Rightarrow^* [\chi \rhd \varepsilon] \otimes \gamma'' \mid \mathsf{aID}(c_0) \mid \infty \mid \mathsf{N}'' \mid \mathsf{V}'' \mid \mathsf{skip}$, where $\mathsf{FinWorld}_d(\mathsf{N}'; \mathsf{V}') = \mathsf{FinWorld}_d(\mathsf{N}''; \mathsf{V}'')$. and apply D-P-CKPT rule to complete the proof of this case.

□

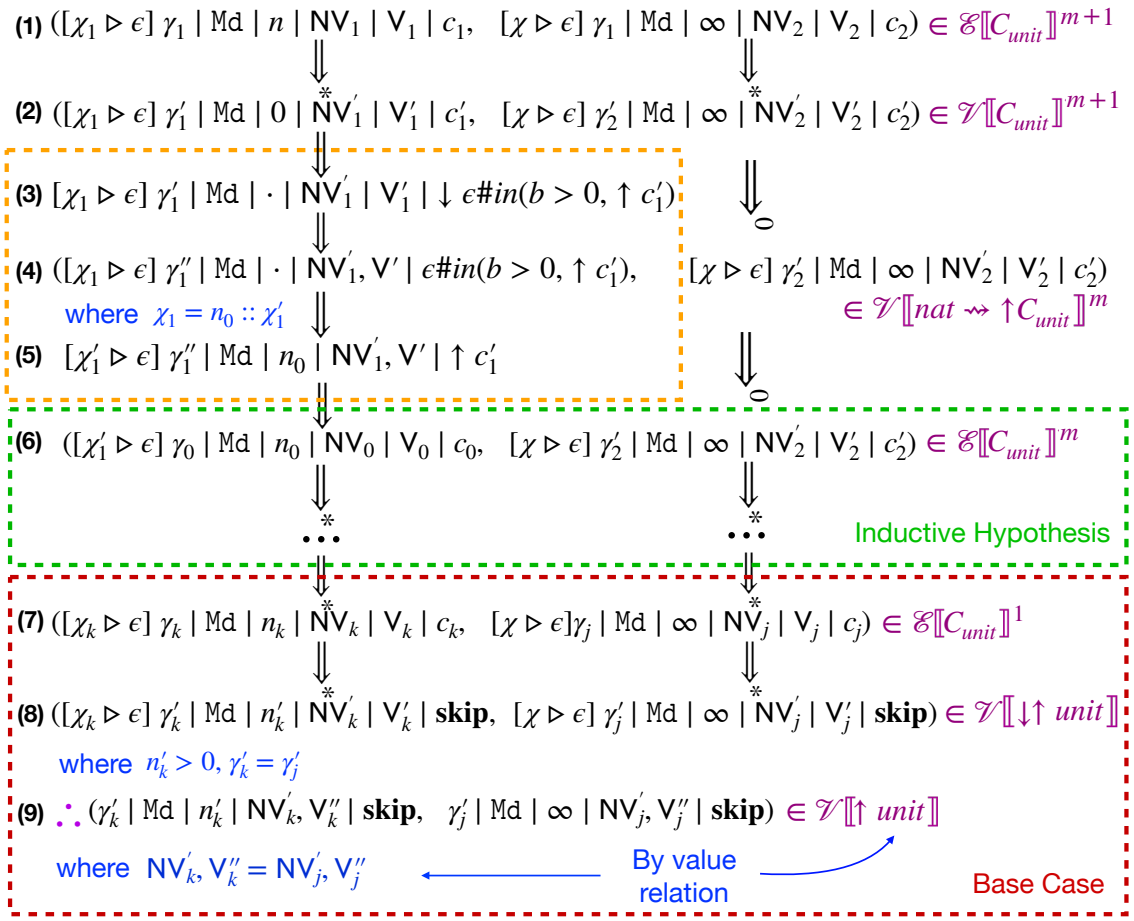

Figure B.2: Illustrations of related configurations evaluate to the same store.

Appendix C

Undo-Logging Appendix

C.1 Syntax and Operational Semantics

C.1.1 Syntax

We present a syntax for programs with atomic regions that assume an undo-logging formulation. The key changes include modified crash types, where $\downarrow\uparrow$ occur together during the power failure steps to restore nonvolatile memory from its checkpoints. Configurations are extended with a checkpoint context K containing nonvolatile and volatile checkpoints that are present in program configurations and closed command configurations.

Commands and expressions

values	v	::=	$\mathsf{n} \mid \mathsf{tt} \mid \mathsf{ff} \mid x$
exprs	e	::=	$v \mid e \odot e$
cmds	c	::=	$\mathsf{skip} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \mid \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \mid x := e \mid c_1; c_2 \mid c_1;_W c_2$
atom. reg. pols.	$aPol$	::=	(ρ, μ, ω)
progs	p	::=	$\mathsf{skip} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c; \mathsf{end}_{\mathsf{atom}}; p \mid c; p$

Access qualifiers and memories

access qualifier	q	::=	$\mathtt{CK} \mid \mathtt{RD} \mid \mathtt{MFstWt} \mid \mathtt{Wtn}$
nonvolatile mem	N	::=	$\cdot \mid \ell \mathbin{@} q \hookrightarrow v, \mathsf{N} \mid \ell_{\mathrm{un}} \mathbin{@} \mathtt{CK} \hookrightarrow v, \mathsf{N}$
volatile mem	V	::=	$\cdot \mid \ell \mathbin{@} \mathtt{CK} \hookrightarrow v, \mathsf{V}$
var loc map	γ	::=	$\cdot \mid \gamma, x \mapsto \ell$

Instructions, statements, and configurations

continuations	κ	::=	$c \mid e$
statements	s	::=	$\kappa \mid i \mid p$
crash instrs	i	::=	$\varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow\kappa) \mid \downarrow\uparrow\kappa$
energy level	g	::=	$\cdot \mid n$
charge stream	χ	::=	$n :: \chi$
checkpoint context	K	::=	$(\mathsf{N}_k; \mathsf{V}_k)$
open config	C_o	::=	$(\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} \mid s) \mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{K} \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{N} \mid \mathsf{V} \mid s)$
closed config	C_c	::=	$[\chi \rhd \varepsilon] \otimes C_o$
exec. mode	$\mathtt{Md}$	::=	$\mathsf{aID}(c) \mid \mathsf{jit}$

C.1.2 Value Configurations

$$\frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{skip})}\ (\text{V-SKIP}) \qquad \frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{n})}\ (\text{V-NAT})$$

$$\frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{tt})}\ (\text{V-BOOL-T}) \qquad \frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{ff})}\ (\text{V-BOOL-F})$$

$$\frac{}{Val(\gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid \kappa)}\ (\text{V-CRASH})$$

$$\frac{}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{N} \mid \mathsf{K} \mid \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow\kappa))}\ (\text{V-\# in})$$

$$\frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{K} \mid \downarrow\uparrow\kappa)}\ (\text{V-}\downarrow\uparrow) \qquad \frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{K} \mid \mathsf{skip})}\ (\text{V-P-DONE})$$

$$\frac{n > 0}{Val([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} \mid \mathsf{skip})}\ (\text{V-C-DONE})$$

C.1.3 Top-level Atomic Region Execution

$$\frac{\begin{array}{c} n > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{K}_0 \mid \gamma_0 \\ [\chi \rhd \varepsilon] \otimes \gamma_0 \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N}_0 \mid \cdot \mid \mathsf{K}_0 \mid c_0 \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0) \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{K}_0 \mid \mathsf{skip} \\ n' > 0 \qquad \mathsf{FinWorld}_d(\mathsf{N}') = \mathsf{N}'' \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{K} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p \ \Rightarrow_p \ [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}'' \mid \mathsf{K} \mid p}\ (\text{D-P-CKPT})$$

$$\frac{\begin{array}{c} n > 0 \qquad n' > 0 \qquad \mathsf{K}_0 = (\emptyset; \emptyset) \\ [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{N} \mid \cdot \mid \mathsf{K}_0 \mid c \ \Rightarrow^* \ [\chi' \rhd \varepsilon] \otimes \ \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathsf{K}' \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{N} \mid \mathsf{K} \mid \ c; p \ \Rightarrow_p \ [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{N}' \mid \mathsf{K}' \mid p}\ (\text{D-P-SEQ})$$

Definition 18 $\mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{K}_0 \mid \gamma_0$ *where* $aPol = (\rho, \mu, \omega)$ *iff*

- $\mathsf{N} = \mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_3$,
- $\mathsf{dom}(\mathsf{N}_1) = \mathsf{range}(\gamma\restriction\rho)$, $\mathsf{dom}(\mathsf{N}_2) = \mathsf{range}(\gamma\restriction\omega)$, *and* $\mathsf{dom}(\mathsf{N}_3) = \mathsf{range}(\gamma\restriction\mu)$,
- $\mathsf{N}_k, \gamma_k = \mathtt{unzip}(\{(\ell^*@\mathtt{CK} \hookrightarrow v,\ x \mapsto \ell^*) \mid \mathsf{N}_2(\ell) = v,\ \ell^* \textit{ fresh}, x \in \omega \textit{ s.t. } \gamma(x) = \ell\})$,
- $\mathsf{N}_0 = \{\ell@\mathtt{RD} \hookrightarrow v \mid \mathsf{N}_1(\ell) = v\} \cup \{\ell_{\mathsf{un}}@\mathtt{CK} \hookrightarrow v \mid \mathsf{N}_2(\ell) = v\} \cup \{\ell@\mathtt{MFstWt} \hookrightarrow v \mid \mathsf{N}_3(\ell) = v\}$,
- $\gamma_0 = \{x_{\mathsf{un}} \mapsto \gamma(x)_{\mathsf{un}} \mid x \in \omega\} \cup \{x \mapsto \gamma(x) \mid x \notin \omega\} \cup \gamma_k$,
- $\mathsf{K}_0 = (\mathsf{N}_k; \emptyset)$

Definition 19 $\mathsf{FinWorld}_d(\mathsf{N}) = \mathsf{N}_f$ *iff*

- $\mathsf{N} = \mathsf{N}_{1\,\mathsf{un}}, \mathsf{N}_2$,
- $\mathsf{N}_f = \{\ell@\mathtt{CK} \hookrightarrow v \mid \ell_{\mathsf{un}}@\mathtt{CK} \hookrightarrow v \in \mathsf{N}_1\} \cup \{\ell@\mathtt{CK} \hookrightarrow v \mid \ell@q \hookrightarrow v \in \mathsf{N}_2, q \in \{\mathtt{RD}, \mathtt{Wtn}\}\}$

C.1.4 Command Execution (Closed Config)

$$\frac{\gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, c \;\rightarrow\; \gamma' \,|\, \mathtt{Md} \,|\, n' \,|\, \mathsf{N}' \,|\, \mathsf{V}' \,|\, c'}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, c \;\Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma' \,|\, \mathtt{Md} \,|\, n' \,|\, \mathsf{N}' \,|\, \mathsf{V}' \,|\, \mathsf{K} \,|\, c'} \text{ (D-STEP)}$$

$$\frac{}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0) \,|\, 0 \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0) \,|\, \cdot \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \downarrow\uparrow c_0)} \text{ (D-CRASH-AID)}$$

$$\frac{}{\begin{array}{l}[n :: \chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0) \,|\, \cdot \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \downarrow\uparrow\kappa) \\ \Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{aID}(c_0) \mid n \mid \mathsf{N} \mid \cdot \mid \mathsf{K} \mid \downarrow\uparrow\kappa\end{array}} \text{ (D-CHARGE-AID)}$$

$$\frac{\mathsf{N} = \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{Wtn}}, \mathsf{N}'''_{\mathtt{CK}} \qquad \mathsf{K} = (\mathsf{N}_k; \emptyset)}{\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0) \,|\, n \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \downarrow\uparrow\kappa \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0) \,|\, n \,|\, \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{MFstWt}}, \mathsf{N}_{k\,\mathtt{un}} \,|\, \cdot \,|\, \mathsf{K} \,|\, \kappa\end{array}} \text{ (D-RESTORE-AID)}$$

$$\frac{}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, 0 \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \kappa \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa)} \text{ (D-CRASH-JIT)}$$

$$\frac{\mathsf{K} = (\cdot; \mathsf{V})}{\begin{array}{l}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \\ \Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa)\end{array}} \text{ (D-S-JIT)}$$

$$\frac{}{[n :: \chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\, \uparrow\kappa} \text{ (D-CHARGE-JIT)}$$

$$\frac{\mathsf{K} = (\cdot; \mathsf{V}_k)}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\uparrow \kappa \;\Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, n \,|\, \mathsf{N} \,|\, \mathsf{V}_k \,|\, \cdot \,|\, \kappa} \text{ (D-RESTORE-JIT)}$$

C.1.5 Command Execution (Open Config)

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{let}\, x = e\, \mathsf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{let}\, x = e'\, \mathsf{in}\, c} \text{ (D-Let-step)}$$

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\ \mathit{fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{let}\, x = e\, \mathsf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{CK} \hookrightarrow e \mid\ c} \text{ (D-Let-v)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ x := e'} \text{ (D-Assign-step)}$$

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q \hookrightarrow e \mid\ \mathsf{skip}} \text{ (D-Assign-V)}$$

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \qquad q' = \delta(q, \mathit{Wt}) \neq \mathtt{UN} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid\ \mathsf{skip}} \text{ (D-Assign-N)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{if}\, e'\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2} \text{ (D-If)}$$

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{if}\, \mathsf{tt}\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1} \text{ (D-If-tt)}$$

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{if}\, \mathsf{ff}\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ c_2} \text{ (D-If-ff)}$$

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_{\gamma \mid \mathsf{V}} c_2} \text{ (D-seq)}$$

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid\ c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid\ c_1';_W c_2} \text{ (D-seq-step)}$$

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction \mathsf{dom}(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathsf{skip};_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid\ c_2} \text{ (D-seq-v)}$$

C.1.6 Expression Execution

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell @ q \hookrightarrow v, \mathsf{V}' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ v} \text{ (D-V-READ)}$$

$$\frac{\begin{array}{c}\gamma = \gamma', [x \mapsto \ell] \\ q' = \delta(q, Rd) \qquad \mathsf{N} = \ell @ q \hookrightarrow v, \mathsf{N}' \qquad \mathsf{N}'' = \ell @ q' \hookrightarrow v, \mathsf{N}' \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}'' \mid \mathsf{V} \mid\ v} \text{ (D-N-READ)}$$

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid\ \mathsf{N} \mid \mathsf{V} \mid\ e_1 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1'}{\gamma \mid \mathtt{Md} \mid n \mid\ \mathsf{N} \mid \mathsf{V} \mid\ e_1 \odot e_2 \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1' \odot e_2} \text{ (D-BINARY-1)}$$

$$\frac{\begin{array}{c}n > 0 \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid\ \mathsf{N} \mid \mathsf{V} \mid\ e_1) \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e_2 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e_2'\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1 \odot e_2 \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1 \odot e_2'} \text{ (D-BINARY-2)}$$

$$\frac{\begin{array}{c}n = n' + 1 \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e_1) \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e_2) \qquad v = e_1 \odot e_2\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid\ \mathsf{N} \mid \mathsf{V} \mid\ e_1 \odot e_2 \to\ \gamma \mid \mathtt{Md} \mid n' \mid\ \mathsf{N} \mid \mathsf{V} \mid\ v} \text{ (D-BINARY-V)}$$

Definition 20 (Well-formedness for configurations) *A configuration* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} \mid c$ *is well-formed iff when* $c = c_1;_{W_1} \cdots ;_{W_{n-1}} c_n;_{W_n} c_{n+1}$ *where* $n > 1$*, all of the following hold*

- $dom(\mathsf{V}_j) \subseteq dom(\mathsf{V}_i) \subseteq dom(\mathsf{V})$
- $\gamma_j \subseteq \gamma_i \subseteq \gamma$

where $1 \leq i < j \leq n$ *and* $W_k = \gamma_k \mid \mathsf{V}_m$ *for all* $m \in [1, n]$.

C.2 Type System

C.2.1 Types

store types	A	::=	$\mathtt{int} \mid \mathtt{bool}$
basic types	T	::=	$\mathtt{unit} \mid A$
unstable types	τ^u	::=	$T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid \mathtt{nat} \rightsquigarrow \tau^u \mid \mathsf{C}_T^{\mathtt{Md}}$
stable types	τ^s	::=	$\uparrow \tau^u$
type variables	$\mathsf{C}_T^{\mathtt{Md}}$	::=	$\mathsf{C}_{\mathtt{unit}} \mid \mathsf{C}_A^{\mathtt{Md}}$
unstable store typing context	Σ	::=	$\cdot \mid x : \downarrow\uparrow A @ \mathtt{CK}, \Sigma \mid x_{\mathtt{un}} : \downarrow\uparrow A @ \mathtt{CK}, \Sigma$
stable store typing context	Ω	::=	$\cdot \mid x : \uparrow A @ q, \Omega$

$$\begin{array}{rl}
\textit{atomic command crash type} & \mathsf{C}_{\mathtt{unit}} = (\mathtt{nat} \rightsquigarrow \downarrow^s_u\uparrow^s_u\mathsf{C}_{\mathtt{unit}}) \vee \downarrow^s_u\uparrow^s_u\mathtt{unit} \\
\textit{atomic expression crash type} & \mathsf{C}^{\mathtt{Md}}_A = (\mathtt{nat} \rightsquigarrow \downarrow^s_u\uparrow^s_u\mathsf{C}_{\mathtt{unit}}) \vee \downarrow^s_u\uparrow^s_u A \\
\textit{jit command crash type} & \mathsf{C}_{\mathtt{unit}} = \downarrow^s_u(\mathtt{nat} \rightsquigarrow \uparrow^s_u\mathsf{C}_{\mathtt{unit}}) \vee \downarrow^s_u\uparrow^s_u\mathtt{unit} \\
\textit{jit expression crash type} & \mathsf{C}^{\mathsf{jit}}_A = \downarrow^s_u(\mathtt{nat} \rightsquigarrow \uparrow^s_u\mathsf{C}^{\mathsf{jit}}_A) \vee \downarrow^s_u\uparrow^s_u A
\end{array}$$

C.2.2 Expression Typing

$$\frac{\Omega = x{:}\uparrow A@q, \Omega_2' \qquad q' = \delta(q, Wt) \neq \mathit{UN}}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{WT}} x : \uparrow A \dashv x{:}\uparrow A@q', \Omega_2'} \text{ (T-Loc-Write)}$$

$$\frac{\Sigma =\downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A \dashv \Omega_1}{\mathtt{Md} \mid\ b : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega_1 \restriction \mathit{dom}(\Omega'), \Omega''_{\mathsf{CK}}} \text{ (T-W-Shift)}$$

$$\frac{\Omega(x) = \uparrow A@q}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} x : \uparrow A} \text{ (T-Loc-Read)} \qquad \frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathtt{tt} :\uparrow \mathtt{bool}} \text{ (T-Bool-t)}$$

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathtt{ff} : \uparrow\mathtt{bool}} \text{ (T-Bool-f)} \qquad \frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathtt{n} : \uparrow\mathtt{int}} \text{ (T-Int)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{un}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \downarrow\uparrow A} \text{ (T-R-Shift)}$$

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A}{\mathtt{Md} \mid b\ >\ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau_1 \vee \downarrow\uparrow A} \text{ (T-}\vee\text{-Succ)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_T \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T'} \qquad \odot : \uparrow T \times \uparrow T' \to \uparrow T''\end{array}}{\mathtt{Md} \mid b\ >\ 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_{T''}} \text{ (T-Binary)}$$

$$\frac{\begin{array}{c}\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau \qquad \mathtt{Md} \mid b\ >\ 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau} \text{ (T-enough?)}$$

C.2.3 Command Typing

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega} \text{ (T-Skip)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \restriction dom(\Omega'), \Omega''_{\mathsf{CK}}} \text{ (T-C-Shift)}$$

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \vee \downarrow\uparrow\mathtt{unit} \dashv \Omega'} \text{ (T-}\vee\text{-Succ)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_A \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}\downarrow\uparrow A@\mathsf{CK} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{let}\, x = e_1 \,\mathsf{in}\, c : \tau \dashv \Omega'} \text{ (T-Let)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'} \text{ (T-Assign)}$$

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{if}\, e \,\mathsf{then}\, c_1 \,\mathsf{else}\, c_2 : \tau \dashv \Omega'} \text{ (T-If)}$$

$$\frac{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''} \text{ (T-seq)}$$

$$\frac{\begin{array}{c}W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \qquad \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2_{\mathtt{RD,MFstWt,Wtn}} \\ \Sigma' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathtt{un}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''} \text{ (T-seq-d)}$$

$$\frac{\begin{array}{c}\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau \dashv \Omega'\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'} \text{ (T-enough?)}$$

C.2.4 Statement Typing

$$\frac{\Sigma = \Sigma^1, \Sigma^2_{\text{un}} \qquad \mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \downarrow\uparrow c_0) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \kappa : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-aID-}\vee\text{-Crash)}$$

$$\frac{\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\uparrow \kappa : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \downarrow\uparrow \kappa) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}} \text{ (T-aID-Charge)}$$

$$\frac{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_{T} \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\uparrow \kappa : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}} \text{ (T-aID-Restore)}$$

$$\frac{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\uparrow \varepsilon \# \mathsf{in}(b > 0, \kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{T'})}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \kappa' : \downarrow\uparrow(\mathtt{nat} \rightsquigarrow \mathsf{C}^{\mathsf{jit}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-Jit-}\vee\text{-Crash)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \mathsf{jit} \mid \cdot \mid \Omega, \Sigma'_{\text{un}} \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa') : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})} \text{ (T-Jit-stop)}$$

$$\frac{\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \uparrow\kappa : \mathsf{C}^{\mathtt{Md}}_{T} \in \mathtt{Sig}}{\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \uparrow\kappa) : \mathtt{nat} \rightsquigarrow \mathsf{C}^{\mathtt{Md}}_{T}} \text{ (T-Jit-Charge)}$$

$$\frac{\Omega = \Omega', \Omega''_{\text{un}} \qquad \mathsf{jit} \mid \cdot \mid \Omega'; \downarrow\Omega'' \mid pcS_t \vdash \kappa' : \mathsf{C}^{\mathtt{Md}}_{T} \in \mathtt{Sig}}{\mathsf{jit} \mid \cdot \mid \Omega \mid \cdot \vdash_{\mathtt{Sig}} \uparrow\kappa' :\uparrow \mathsf{C}^{\mathtt{Md}}_{T}} \text{ (T-Jit-Restore)}$$

C.2.5 Program Typing

$$\frac{\begin{array}{c}\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0 \\ \mathtt{Sig} = \{\mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'\} \\ \mathsf{aID}(c_0) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \\ \Omega' \restriction \{\mathtt{MFstWt}\} = \emptyset \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}\end{array}}{b : \mathtt{nat} \mid \Omega \vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-Ckpt)}$$

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega' \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash c; p : \uparrow \mathsf{C}_{\mathtt{unit}}} \text{ (T-P-seq)}$$

Definition 21 (**$\mathsf{InitWorld}_t$ for undo-logging**) $\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0$ *iff*

- $\mathsf{dom}(\Omega) = \rho \cup \mu \cup \omega$,
- $\rho \cap \mu = \emptyset$, $\mu \cap \omega = \emptyset$, $\rho \cap \omega = \emptyset$,
- $\Sigma_0 = \{x_{\text{un}} : \downarrow\uparrow A@\mathtt{CK} \mid x : \uparrow A@\mathtt{CK} \in \Omega^c\}$
- $\Omega_0 = \{x : \uparrow A@\mathtt{RD} \mid x : \uparrow A@q \in \Omega^r\} \cup \{x : \uparrow A@\mathtt{MFstWt} \mid x : \uparrow A@q \in \Omega^m\} \cup \Omega^c$

where $aPol = (\rho, \mu, \omega)$ *and* $\Omega = \Omega^r, \Omega^m, \Omega^c$ *and* $\Omega^r = \Omega\restriction\rho$, $\Omega^m = \Omega\restriction\mu$, *and* $\Omega^c = \Omega\restriction\omega$.

C.2.6 Store Typing

We present a modified store typing for undo-logging.

$$\frac{}{\vdash^{\mathsf{aID}}_{\gamma} \cdot \mid \cdot : \cdot \mid \cdot} \text{ (Empty)}$$

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V}' : \Omega \mid \Sigma \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v \qquad \mathsf{aID} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma, (x{:}\downarrow\uparrow A@q)} \text{ (V-Loc)}$$

$$\frac{\vdash^{\mathsf{aID}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} : \Omega \mid \Sigma \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v \qquad q \neq \mathsf{CK} \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{aID} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, (x : \uparrow A@q) \mid \Sigma} \text{ (NV-Loc-aID-1)}$$

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{N} = \mathsf{N}', \ell_{\mathrm{un}}@q \hookrightarrow v \qquad q = \mathsf{CK} \qquad \mathsf{aID} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathrm{RD}} v : \uparrow A \qquad \vdash^{\mathsf{aID}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} : \Omega \mid \Sigma}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega, (x : \uparrow A@q) \mid \Sigma, (x_{\mathrm{un}} : \downarrow\uparrow A@q)} \text{ (NV-Loc-aID-2)}$$

The rule V-Loc checks a volatile location's type with respect to Σ and γ. The rule NV-Loc-aID-1 types non-checkpointed locations in nonvolatile memory by checking that the value stored in the location ℓ allocated for x has the same type as x at the type level. Lastly, the rule NV-Loc-aID-2 applies when x is checkpointed, meaning it has locations in both N and K. Though, since K does not appear in the open configuration operational semantic rules, it is not needed for progress and preservation and is therefore omitted from this definition. Therefore each checkpointed location in N corresponds to both an unstable and stable typing. For reasoning about high-level correctness, this rule would be extended with the checkpoint context where it is typed according to the stable types. Since in this system, V_k will always be empty for atomic regions, the rules are complete for the mode aID.

C.3 Progress and Preservation

Lemma 15 (Progress for shifted expressions) *If*

$$\mathtt{Md} \mid b{:}\mathtt{nat} \mid \Omega \vdash_{Rd} e : \uparrow A$$

then $\forall\ n : \mathtt{nat}$ *with* $n > 0$ *and* $\forall\, \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Proof: The proof proceeds by structural induction over $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{Rd} e : \uparrow A$.

Case 1 [T-Loc-Read]**.** Suppose the last rule in the typing derivation is T-Loc-Read:

$$\frac{\Omega = x : \uparrow A@q, \Omega'}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} x : \uparrow A} \text{ (T-Loc-Read)}$$

Then by inversion, we have $\Omega = x : \uparrow A@q, \Omega'$. By assumption, $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$. By inversion of $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$ according to the well-formedness definition, one of the two subcases hold:

Subcase 1. $\mathsf{N} = \ell@q \hookrightarrow v, \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-loc-Read applies:

$$\frac{\mathsf{N} = \ell@q \hookrightarrow v, \mathsf{N}' \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \delta(q, \mathtt{RD}) \neq \mathtt{UN} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-loc-Read)}$$

This yields the desired result.

Subcase 2. $\mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-loc-Read applies:

$$\frac{\mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}' \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \delta(q, \mathtt{RD}) \neq \mathtt{UN} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-var-Read)}$$

This yields exactly the desired result.

Case 2 [T-Bool-t]**.** Suppose that the last rule in the typing derivation is T-Bool-t:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathbf{tt} :\uparrow \mathit{bool}} \text{ (T-Bool-t)}$$

By the value rule V-Bool-t and the assumption $n > 0$ we get

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{tt})} \text{ (V-Bool-t)}$$

Case 3 [T-Bool-f]**.** Suppose that the last rule in the typing derivation is T-Bool-f:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathbf{ff} :\uparrow \mathit{bool}} \text{ (T-Bool-f)}$$

By the value rule V-Bool-f and the assumption $n > 0$ we get

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{ff})} \text{ (V-Bool-f)}$$

Case 4 [T-Int]**.**

Suppose that the last rule in the typing derivation is T-Int:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash_{\mathtt{RD}} \mathbf{n} : \uparrow \mathit{int}} \text{ (T-Int)}$$

By the value rule V-Int and the assumption $n > 0$ we get:

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{n})} \text{ (V-Int)}$$

□

Theorem 14 (Progress for expressions) *If* $\mathtt{Md} \mid b \mathcal{R} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Rd;\mathtt{Sig}} e : \tau$*, then* $\forall\ n : nat$ *with* $n\mathcal{R}m$ *and* $\forall\ \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*, either*

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b \ \mathcal{R}\ m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{Rd;\mathtt{Sig}} e : \tau$. We consider a specific (co-)natural number $n\mathcal{R}m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b\ = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau \qquad \mathtt{Md} \mid b\ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau} \text{ (T-enough?)}$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n (i.e. $n = 0$ or $n > 0$):

Subcase 1 [$n = 0$]**.** By the value rule V-e-Crash, we have

$$\mathit{Val}(\gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid e),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, we have

(1) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} e : \tau$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau$

(3) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \tau\}$

(4) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis to (2) to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

which completes the proof of this subcase.

Case 2 [T-Binary**].**

Suppose the last rule in the typing derivation is T-Binary and $e = e_1 \odot e_2$:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_T^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}} \qquad \odot : \uparrow T \times \uparrow T' \rightarrow \uparrow T''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}_{T''}^{\mathtt{Md}}}\ (\text{T-Binary})$$

By assumption, we know that $n > 0$. By inversion, we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_T^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}_{T'}^{\mathtt{Md}}$

(3) $\odot : \uparrow T \times \uparrow T' \rightarrow \uparrow T''$

By the inductive hypothesis applied to (1), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$, or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$.

From here, the proof proceeds in two subcases:

Subcase 1. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$

We apply the inductive hypothesis to (2) to get two sub-subcases.

Subcase 1a. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2)$.

Since $n > 0$, we can apply the dynamic rule D-Binary-v to get

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v,$$

where $v = e_1 \odot e_2$, and $n = n' + 1$.

Subcase 1b. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_2'$. By D-Binary-2,

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid V \mid e_1 \odot e_2'$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \ \rightarrow \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$. Then, by D-Binary-1,

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \ \rightarrow \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1' \odot e_2$$

The desired result holds in all subcases.

Case 3 [T-$\vee$-succ]**.**

Suppose the last rule in the typing derivation is T-$\vee$-succ:

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A}{\mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau \vee \downarrow\uparrow A} \text{ (T-}\vee\text{-succ)}$$

By assumption, we get $n > 0$. By inversion, we have:

$$\dagger \ \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A$$

We know that the last rule for deriving $\dagger$ is T-R-Shift:

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{un}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} v : \downarrow\uparrow A} \text{ (T-R-Shift)}$$

By inversion, we have:

(1) $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD}} v : \uparrow A$

(2) $\Sigma = \downarrow\Sigma'$

(3) $\Omega = \Omega', \Omega''_{\mathsf{un}}$

By assumption $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and (2), it follows from Lemma 27 that $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. Then the desired result follows directly by application of Lemma 26 to (1) (since $n > 0$).

□

Lemma 16 *If* $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ *and* $\Sigma = \downarrow\Sigma'$*, then* $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$.

Proof: The proof is by induction on the structure of $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. For each step in the derivation, we build the corresponding step of a derivation for $\vdash_\gamma^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$ according to the well-formedness definition. □

Theorem 6.1 (Progress for Commands) *If* $\mathtt{Md} \mid b \ \mathcal{R} \ m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$*, then* $\forall$ $n : \mathtt{nat}$ *with* $n \mathcal{R} m$ *and* $\forall\, \gamma, \mathsf{N}, \mathsf{V}$ *with* $\vdash_\gamma^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ *or*
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b \mathrel{\mathcal{R}} m : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$. We consider a specific (co-)natural number $n\mathcal{R}m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega' \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\} \\ \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'}\ (\text{T-ENOUGH?})$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n:

Subcase 1 [$n = 0$]**.** By the value rule V-c-Crash, we have

$$\mathit{Val}(\gamma \mid \mathsf{Md} \mid 0 \mid \mathsf{N} \mid \mathsf{V} \mid c),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, and since $n > 0$, we have

(1) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c : \tau$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$

(3) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \tau\}$

(4) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis on this judgment to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c)$ or
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

which completes the proof of this subcase.

Case 2 [T-If]**.**

Suppose the last rule in the typing derivation is T-if and $c = \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2$:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau \dashv \Omega'}\ (\text{T-If})$$

By assumption, we know that $n > 0$. By inversion, we have:

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega'$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'$

By Theorem 16 applied to (1), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

From here, the proof proceeds in two subcases:

Subcase 1. $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$.

Since $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool}$, we have by inversion on T-Enough? followed by inversion on T-$\vee$-succ that

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \downarrow\uparrow bool$$

Again, by inversion on T-R-Shift, we have

(1) $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \uparrow bool$

(2) $\Omega = \Omega', \Omega''_{\mathsf{un}}$

(3) $\Sigma = \downarrow\Sigma'$

By inversion on the rules T-bool-t and T-bool-f, we have that either e is **tt** or **ff**.

Subcase 1a. If $e = \mathbf{tt}$, then since $n > 0$ the rule D-If-tt applies and yields the desired result.

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1}\ \text{(D-If-tt)}$$

Subcase 1b. Similarly, if $e = \mathbf{ff}$, then since $n > 0$, the rule D-If-ff applies and yields the desired result.

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{ff}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2}\ \text{(D-If-ff)}$$

Subcase 2. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$. Then the rule D-if applies and produces the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ e \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \rightarrow \\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e'\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\end{array}}\ \text{(D-If)}$$

Case 3 [(T-Let)]**.**

Suppose the last rule in the typing derivation is T-Let:

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e_1\, \mathbf{in}\, c : \tau \dashv \Omega'}\ \text{(T-Let)}$$

By assumption, we know $n > 0$. By inversion, we have a typing derivation for:

(1) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$

By Theorem 16 applied to (1), either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$.

The proof proceeds in two subcases.

Subcase 1. Suppose that $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1)$. Then since $n > 0$, the rule D-Let-step-v applies and yields the desired result. Note that here γ' extends γ by adding a new mapping from x to ℓ.

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \\ \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\ \mathit{fresh} \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e_1\, \mathbf{in}\, c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell \mathbin{@} \mathtt{Ck} \hookrightarrow e_1 \mid c}\ \text{(D-Let-step-v)}$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$. Then since $n > 0$, the rule D-Let-step applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e'\, \mathbf{in}\, c}\ \text{(D-Let-step)}$$

Case 4 [T-∨-Succ].

Suppose the last rule in the typing derivation is T-∨-Succ:

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : {\downarrow}{\uparrow}\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \tau \vee {\downarrow}{\uparrow}\mathtt{unit} \dashv \Omega'}\ \text{(T-∨-Succ)}$$

Then since $n > 0$, the rule V-skip applies and yields the desired result.

$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathsf{Md} \mid n \mid\ \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip})}\ \text{(V-skip)}$$

Case 5 [T-Assign]**.**

Suppose the last rule in the typing derivation is T-Assign:

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid\ b > 0 : \mathtt{nat} \ \mid\ \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A \dashv \Omega_1\end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \ \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} p := e \ : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega_1} \text{ (T-Assign)}$$

By assumption, we know that $n > 0$. By inversion on T-Assign, we have

(i) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \ \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(ii) $\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \ \mid\ \Omega; \Sigma \vdash_{Wt} p : \downarrow\uparrow A \dashv \Omega_1$

By Theorem 16 applied to (i), either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

The proof proceeds in two subcases.

Subcase 1. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$.

By inversion on T-w-Shift applied to (ii), we have:

$$\frac{\Sigma =\downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{WT}} p : \uparrow A \dashv \Omega_2}{\mathtt{Md} \mid\ b : \mathtt{nat} \ \mid\ \Omega; \Sigma \vdash_{\mathtt{WT}} p : \downarrow\uparrow A \dashv \Omega_2 \lceil dom(\Omega'), \Omega''_{\mathsf{CK}}} \text{ (T-W-Shift)}$$

(1) $\Sigma =\downarrow \Sigma'$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{Wt} p : \uparrow A \dashv \Omega_2$

(3) $\Omega_1 = \Omega_2 \lceil dom(\Omega'), \Omega''_{\mathsf{CK}}$

(4) $\Omega = \Omega', \Omega''_{\mathsf{CK}}$

Now we proceed by inversion on T-Loc-Write applied to (2). From this inversion,

$$\frac{\Omega, \Sigma' = x{:}\uparrow A @ q, \Omega'_2 \qquad q' = \delta(q, Wt) \neq \mathtt{UN}}{\kappa \mid \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{Wt} p : \uparrow A \dashv \Omega_2} \text{ (T-Loc-Write)}$$

we have:

(i) $\Omega, \Sigma' = x{:}\uparrow A @ q, \Omega'_2$, and

(ii) $q' = \delta(q, Wt) \neq \mathtt{UN}$

where $\Omega_2 = x{:}\uparrow A @ q', \Omega'_2$

By assumption, we have $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$. By Lemma 27, and $\Sigma =\downarrow \Sigma'$, we have $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega', \Sigma'$. By the well-formedness definition, one of the two subcases hold:

Subcase 1a. $\mathsf{V} = \ell@q \hookrightarrow v', \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, the D-Assign-V rule applies and yields the desired result.

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

Subcase 1b. $\mathsf{N} = \ell@q \hookrightarrow v', \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, the D-Assign-N rule applies and yields the desired result. By definition of δ, it follows that $q \neq \mathtt{Rd}$, for we cannot have $\delta(\mathtt{Rd}, Wt) \neq \mathtt{UN}$.

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad q \neq \mathtt{RD} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}} \text{ (D-Assign-N)}$$

Subcase 2. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$. The rule D-Assign-step applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \to \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid p := e \ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid p := e'} \text{ (D-Assign-step)}$$

Case 6 [(T-seq)]**.**

Suppose the last rule in the typing derivation is T-seq:

$$\frac{\begin{array}{c} \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''} \text{ (T-seq)}$$

Then the desired result follows by D-seq:

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2} \text{ D-seq}$$

Case 7 [(T-seq-d)]**.** Suppose the last rule in the typing derivation is T-seq-d:

$$\frac{\begin{array}{c} \text{T-seq-d} \\ W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathsf{unit}} \dashv \Omega' \qquad \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2 \\ \Sigma'' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{un}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''}$$

By inversion we have

(1) $W = \gamma \mid V$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$

(3) $\Sigma'' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{un}}$

(4) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

(5) $\Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2$

By the inductive hypothesis applied to (1), we have that either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$ or
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$.

The proof proceeds in two subcases.

Subcase 1. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$.

By (2), $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$, and the assumption $n > 0$, it follows by inversion on T-Enough? that

(a) $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}\}$

(b) $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$

(c) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c_1 : \mathsf{C}_{\mathtt{unit}}$

(d) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$

By definition, $\mathsf{C}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Additionally, observe that $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1)$ implies that $c_1 = \mathsf{skip}$. Then by inversion of T-$\vee$-Succ

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit} \dashv \Omega'}\ (\text{T-}\vee\text{-Succ})$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'$.

The assumption $n > 0$ implies that $n = n' + 1$. By this fact, (1), and $\mathsf{V} = \mathsf{V} \restriction \mathit{dom}(\mathsf{V})$ (which is trivially satisfied because $\mathsf{V} = \mathsf{V}$), the rule D-seq-v applies

$$\text{D-seq-v}\ \frac{n = n' + 1 \qquad W = \gamma \mid \mathsf{V} \qquad \mathsf{V} = \mathsf{V} \restriction \mathsf{dom}(\mathsf{V})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ \mathsf{skip};_W c_2 \ \rightarrow \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \ c_2}$$

Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ \mathbf{skip};_W c_2 \ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_2$ is the desired result.

Subcase 2. $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$.

Since $n > 0$, the rule D-cont applies and yields the desired result.

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \;\rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\; c_1 ;_W c_2 \quad \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1' ;_W c_2}\ \text{(D-SEQ-STEP)}$$

□

Axiom 2 (positive input to generation channel) $\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0$.

Lemma 17 (well-typedness of expressions under crash in jit**)** $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}'} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}\}$.

Proof: Note that $\mathsf{C}_{\mathtt{A}}^{\mathsf{jit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}$. Let $\Omega' = \Omega, \Omega''_{\mathsf{un}}$ where $\Sigma = \downarrow\Omega''$. By axiom 3, we have $\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumption that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathtt{A}}^{\mathsf{jit}} \in \mathtt{Sig}' \quad \Omega' = \Omega, \Omega''_{\mathsf{un}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{RD};\mathtt{Sig}'} \uparrow e : \uparrow\mathsf{C}_{\mathtt{A}}^{\mathsf{jit}}}\ \text{(T-JIT-RESTORE)} \quad \varepsilon \# \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega, \Omega''_{\mathsf{un}} \vdash_{\mathtt{RD};\mathtt{Sig}'} \varepsilon \# \mathsf{in}(b > 0, \uparrow e) : (\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{A}}^{\mathsf{jit}})}\ \text{(T-CHARGE)} \quad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}'} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow e) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathtt{A}}^{\mathsf{jit}})}\ \text{(T-JIT-STOP)}}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{A}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}}\ \text{(T-}\vee\text{-CRASH)}$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{A}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}$ yields the desired result.

□

Lemma 18 (well-typedness of expressions under crash in aID**)** *If* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{RD};\mathtt{Sig}} e' : \tau$ *s.t.* $\mathsf{dom}(\Sigma') \supseteq \mathsf{dom}(\Omega^1_{\mathsf{CK}})$ *where* $\Omega = \Omega^1_{\mathsf{CK}}, \Omega^2$, *then* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \tau$ *s.t.* $\mathsf{dom}(\Sigma) \supseteq \mathsf{dom}(\Omega^1_{\mathsf{CK}})$.

Proof: Let the type $\tau = \mathsf{C}_A^{\mathsf{aID}}$ and note that $\mathsf{C}_A^{\mathsf{aID}} = (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}_{\mathtt{unit}}^{\mathsf{aID}}) \vee \downarrow\uparrow\mathtt{A}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma' = \Sigma^1, \Sigma^2_{\mathsf{un}} \quad \varepsilon \# \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{un}} \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_T^{\mathsf{aID}} \in \mathtt{Sig}}\ \text{(T-aID-RESTORE)}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{un}} \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 : \downarrow\uparrow\mathsf{C}_T^{\mathsf{aID}}}\ \text{(T-aID-CHARGE)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\mathsf{un}} \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \downarrow\uparrow c_0) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}_{T'}^{\mathsf{aID}}}\ \text{(T-aID-}\vee\text{-CRASH)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} e' : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}_{T'}^{\mathsf{aID}}) \vee \downarrow\uparrow T}$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{un}} \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_T^{\mathsf{aID}} \in \mathtt{Sig}$

(2) $\Sigma' = \Sigma^1, \Sigma^2_{\text{un}}$

(3) $\varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation where $\Sigma \supseteq \Sigma^2_{\text{un}}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_T}\ \text{(T-aID-Restore)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}}\ \text{(T-aID-Charge)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} e : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}\ \text{(T-aID-}\vee\text{-Crash)}}{\Sigma' = \Sigma^1, \Sigma^2_{\text{un}} \qquad \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0}$$

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} e : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{A}$ yields the desired result.

□

Lemma 19 (well-typedness of commands under crash in `jit`) $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}\}$.

Proof: Note that $\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Let $\Omega' = \Omega, \Omega''_{\text{un}}$ where $\downarrow\Omega'' = \Sigma$. By axiom 3, we have that $\varepsilon \,\#\, \mathsf{iη}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumptions $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}'} \qquad \Omega' = \Omega, \Omega''_{\text{un}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{Sig}'} \uparrow c : \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}}\ \text{(T-Jit-Restore)} \qquad \varepsilon \,\#\, \mathsf{iη}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega' \vdash_{\mathtt{Sig}'} \varepsilon \,\#\, \mathsf{iη}(b > 0, \uparrow c) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ \text{(T-Charge)} \qquad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} \downarrow\varepsilon \,\#\, \mathsf{iη}(b > 0, \uparrow c) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ \text{(T-Jit-stop)}}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ \text{(T-}\vee\text{-Crash)}$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$ yields the desired result. □

Lemma 20 (well-typedness of commands under crash in aID) *If* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \vdash_{\mathtt{Sig}} c' : \tau$ *s.t.* $\mathsf{dom}(\Sigma') \supseteq \mathsf{dom}(\Omega^1_{\text{CK}})$ *where* $\Omega = \Omega^1_{\text{CK}}, \Omega^2$ *then* $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau$ *s.t.* $\mathsf{dom}(\Sigma) \supseteq \mathsf{dom}(\Omega^1_{\text{CK}})$.

Proof: Let the type $\tau = \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}$ and note that $\mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}} = (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma = \Sigma^1, \Sigma^2_{\text{un}} \quad \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0 \quad \mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_T}\ \text{(T-aID-Restore)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}}\ \text{(T-aID-Charge)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c' : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}\ \text{(T-aID-}\vee\text{-Crash)}$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}$

(2) $\Sigma = \Sigma^1, \Sigma^2_{\text{un}}$

(3) $\varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma = \Sigma^1, \Sigma^2_{\text{un}} \quad \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0 \quad \mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_T}\ \text{(T-aID-Restore)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\text{un}} \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}}\ \text{(T-aID-Charge)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}\ \text{(T-aID-}\vee\text{-Crash)}$$

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{unit}$ yields the desired result.

□

Theorem 15 (preservation for expressions) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \geq 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \tau$$

and for some $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ *and natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$$

then

$$\mathtt{Md} \mid b \geq 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \tau$$

with $n' \geq 0$.

Proof: The proof is by induction on the size of e. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$.

Case 1. [D-Binary-1]**.**

$$\dfrac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'_1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'_1 \odot e_2}\ \text{(D-Binary-1)}$$

By the first premise of D-Binary-1, we have that $n > 0$. By inversion on the assumed typing judgment (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau.$$

By inversion on $(\dagger_1)$ via T-Binary

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

By the inductive hypothesis applied to (1) and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e_1'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' : \tau^s,$$

and $n' \geq 0$. By the following application of the T-Binary rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' : \tau^s \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s} \text{ (T-Binary)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$.

In both subcases, we have the typing judgments $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$ and $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s$. Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau^s \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau} \text{ (T-enough?)}$$

Case 2 [D-Binary-2].

$$\frac{\begin{array}{c} n > 0 \qquad \gamma \mid \mathit{Val}(\mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_2' \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_1 \odot e_2'} \text{ (D-Binary-2)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s.$$

By inversion on $(\dagger_1)$ via T-Binary

$$\frac{\begin{array}{c} \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \\ \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau \end{array}}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

By the inductive hypothesis applied to $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$ and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid e_2'$, it follows that

$$\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2' : \tau^s,$$

and $n' \geq 0$. By the following application of the T-Binary rule we get

$$\frac{\begin{array}{c} \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \\ \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2' : \tau \end{array}}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s} \text{ (T-Binary)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c} \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau^s \\ \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau \end{array}}{\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau} \text{ (T-enough?)}$$

Case 3. [D-Binary-v].

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1) \qquad \begin{array}{c} n = n' + 1 \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_2) \end{array} \qquad v = e_1 \odot e_2}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e_1 \odot e_2 \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-Binary-v)}$$

By the first premise $n > 0$ and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s.$$

By inversion on ($\dagger_1$) via T-Binary

$$\frac{\begin{array}{c} \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau^s} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_1 : \tau^s$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e_2 : \tau^s$

From (1) and (2), we apply inversion on T-Enough?, T-$\vee$-Succ, and T-R-Shift to get $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e_1 : \uparrow A^s$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e_2 : \uparrow A^s$ for $\Sigma =\downarrow \Sigma'$. By definition of $\odot$ operator, we have $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} v : \uparrow A^s$ for $v = e_1 \odot e_2$. By application of T-$\vee$-Succ and T-R-Shift we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c} \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau^s \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau \end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} v : \tau} \text{ (T-enough?)}$$

Case 4. [D-Var-Read].

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}' \qquad \delta(q, \mathtt{RD}) \neq \mathtt{UN} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid v} \text{ (D-var-Read)}$$

By the last premise $n > 0$ and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s.$$

By inversion on $(\dagger_1)$ via T-$\vee$-Succ

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau_1 \vee \downarrow\uparrow A^i} \text{ (T-}\vee\text{-Succ)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i$.

By inversion on $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i$ via T-R-Shift

$$\frac{\Sigma = \downarrow\Sigma' \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^i}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A^i} \text{ (T-R-Shift)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^i$ and $\Sigma = \downarrow\Sigma'$.

By Lemma 27 applied to the assumption $\vdash_\gamma^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ and premise $\Sigma = \downarrow\Sigma'$, we know that $\vdash_\gamma^{Md} \mathsf{N} \mid \mathsf{V} : \Omega, \Sigma'$. By definition of well-formedness, we know that $\gamma = \gamma', [x \mapsto \ell]$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} x : \uparrow A^s$

By application of T-$\vee$-Succ and T-R-Shift we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$.

In both subcases, we have $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s$. Then the desired result follows by T-Enough?:

$$\frac{\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau^s \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} x : \tau} \text{ (T-enough?)}$$

Case 5. [D-N-READ]**.**

The proof is analogous to the previous case.

□

Definition 22 *We write* $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$ *where* $x{:}\tau@q \in \Sigma'$ *iff* $\gamma = [x \mapsto \ell], \gamma'$ *and* $x{:}\tau@q \in \Sigma$ *and* $\ell \in dom(\mathsf{V})$.

Lemma 21 (equality of trimmed volatile contexts) *If*

(i) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$
(ii) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,
(iii) $\vdash^{\mathtt{Md}}_{\gamma''} \mathsf{N}' \mid \mathsf{V}' : \Omega \mid \Sigma''$,
(iv) $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ *and* $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$
(v) $\gamma_0 \subseteq \gamma$ *and* $\gamma_0 \subseteq \gamma''$
then $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

Proof: We need to show that $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$. The proof proceeds by proving each direction separately:

Ad ⇒. Let $x : \downarrow\uparrow A@q \in \Sigma'$. Then by (i), we have

(1) $x : \downarrow\uparrow A@q \in \Sigma$
(2) $\gamma_0 = [x \mapsto l], \gamma_0'$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma''$. From (2) and $\gamma_0 \subseteq \gamma''$, it follows that $\exists \gamma_0'' \supseteq \gamma_0'$ such that $\gamma'' = [x \mapsto l], \gamma_0''$ $(*)$. By (iv) and (3), we have that $l \in dom(\mathsf{V}')$. So, $\exists \mathsf{V}_0'$, v such that $\mathsf{V}' = \mathsf{V}_0', l \hookrightarrow v$ $(**)$ and $\cdot \vdash v : \uparrow A$ $(***)$. Note that inverting (iii) via V-LOC yields the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma_0''} \mathsf{N}' \mid \mathsf{V}_0' : \Omega \mid \Sigma_0''$ (†) and $q = \mathtt{Ck}$ (‡). Then, it follows by V-LOC applied to $(*)$, $(**)$, $(***)$, (†), (‡) that $x : \downarrow\uparrow A@q \in \Sigma''$.

Ad ⇐. Let

(1) $x : \downarrow\uparrow A@q \in \Sigma''$
(2) $\gamma_0 = [x \mapsto l], \gamma_0'$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma'$. To this end, it suffices to show that $x : \downarrow\uparrow A@q \in \Sigma$. By (3) and $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$, we have that $l \in dom(\mathsf{V})$. Therefore, $\exists\, \mathsf{V}_0', v$ such that $\mathsf{V} = \mathsf{V}_0', l@q \hookrightarrow v$ $(*)$, $\cdot \vdash v : \uparrow A$ $(**)$, and $q = \mathtt{Ck}$. From (2) and $\gamma_0 \subseteq \gamma$, we have that $\exists\, \gamma' \supseteq \gamma_0'$ such that $\gamma = [x \mapsto l], \gamma'$. Note that inverting (ii) via V-LOC yields the well-formedness judgment $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}_0' : \Omega \mid \Sigma_0$ $(***)$. By V-Loc applied to $(*)$, $(**)$, $(***)$, $q = \mathtt{Ck}$, and $\gamma = [x \mapsto l], \gamma'$, we have

$$x : \downarrow\uparrow A@q \in \Sigma$$

Then it follows by definition 28 applied to (i) that

$$x : \downarrow\uparrow A@q \in \Sigma'.$$

We have shown that $x : \downarrow\uparrow A@q \in \Sigma'$ iff $x : \downarrow\uparrow A@q \in \Sigma''$, $\gamma = [x \mapsto l], \gamma'$, and $l \in dom(\mathsf{V})$. The desired result holds by definition 28. □

Lemma 22 (well-formedness of smaller memories) *If*

(i) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$,

(ii) $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$,

(iii) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma')$, *and*

(iv) $\gamma' \subseteq \gamma$

then $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$.

Proof: By (2), observe that $\mathsf{V} \supseteq \mathsf{V}''$. The proof proceeds by induction on the size of $\mathsf{V} - \mathsf{V}''$.

Base case: $|\mathsf{V} - \mathsf{V}''| = 0$**.** If $|\mathsf{V} - \mathsf{V}''| = 0$, then $\mathsf{V} = \mathsf{V}''$ and the desired result holds by assumption (1).

Inductive case. Let $|\mathsf{V} - \mathsf{V}''| = k + 1$ where $|\mathsf{V}_0 - \mathsf{V}''| = k$ where $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$ and $\cdot \vdash v : \uparrow A$. Let $x{:}\downarrow\uparrow A@q \in \Sigma$ such that $\Sigma = \Sigma_0, (x{:}\downarrow\uparrow A@q)$. By inversion on V-Loc applied to the assumed judgment:

$$\frac{\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 : \Omega \mid \Sigma_0 \qquad \gamma = [x \mapsto l], \gamma_0 \qquad \mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v \qquad \cdot \vdash v : \uparrow A}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma_0, (x{:}\downarrow\uparrow A@q)}\ \text{V-Loc}$$

we learn that

(1) $\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 : \Omega \mid \Sigma_0$

(2) $\gamma = [x \mapsto l], \gamma_0$

(3) $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$

(4) $\cdot \vdash v{:}\uparrow A$

Now we need to show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$.

Ad $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$**.** To show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it suffices to show that $l \notin dom(\mathsf{V}')$:

By assumption, it follows that $l \in \mathsf{V} - \mathsf{V}''$, or equivalently, $l \in \mathsf{V}$ and $l \notin \mathsf{V}''$. Now suppose that $l \in dom(\mathsf{V}')$. Then it follows by (ii) that $l \in \mathsf{V}''$, a contradiction. Therefore, we have $l \notin dom(\mathsf{V}')$.

Therefore, $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ follows by $l \notin dom(\mathsf{V}')$ and (ii).

Ad $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$**.** To show $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we first need to show that

(a) $dom(\mathsf{V}') \subseteq dom(\mathsf{V}_0)$

(b) $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$

(c) $\gamma \supseteq \gamma'$

(d) $\gamma_0 \supseteq \gamma'$

(a) follows by $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$. Towards (b), note that since $\mathsf{V} \supseteq \mathsf{V}''$ by assumption, we have $dom(\mathsf{V}) \supseteq dom(\mathsf{V}'')$. By $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it follows that $dom(\mathsf{V}'') = dom(\mathsf{V}')$. Therefore, $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$ (b) holds. (c) follows by assumption (iv). By definition, $\gamma = [x \mapsto l], \gamma_0$, so γ_0 is the smallest subset of γ not containing $x \mapsto l$. Observe that γ' is a subset of γ that does not contain $x \mapsto l$. So, $\gamma \supseteq \gamma_0 \supseteq \gamma'$, which concludes the proof of (d).

$\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$ follows by lemma 36 applied to (iii), (a), (b), (c), and (d).

By the inductive hypothesis applied to (1), $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we have $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega \mid \Sigma'$. This is the desired result.

□

Lemma 23 (Monotonicity of volatile memories) *If* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *where* $c \neq c_1;_W c_2$*, then* $dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$*,* $\gamma \subseteq \gamma'$*.*

Proof: The proof is straightforward, proceeding in cases on the dynamic rules. □

Definition 23 $\Omega > \Omega'$ *iff* $\forall x{:}\tau@q \in \Omega$*, we have* $x{:}\tau@q' \in \Omega'$ *where* $q' \neq UN$ *and either* $q = q'$ *or* $\delta(q, Wt) = q'$*.*

Lemma 24 *If* $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$*, then* $dom(\Omega) \subseteq dom(\Omega')$*.*

Proof: The proof is by induction on the size of c. We consider possible cases for $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$.

Case [T-C-SHIFT]**.**

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{CK} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \restriction dom(\Omega'), \Omega''_{CK}} \text{ (T-C-SHIFT)}$$

By definition of $\restriction$ and $\Omega = \Omega', \Omega''_{CK}$, we obtain the desired result $dom(\Omega) \subseteq dom(\Omega_1 \restriction dom(\Omega'), \Omega''_{CK})$.

Case [T-SEQ]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''} \text{ (T-SEQ)}$$

By inversion of T-SEQ, we learn that

- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

By applying the inductive hypothesis to each of these judgments, we learn that $dom(\Omega) \subseteq dom(\Omega')$ and $dom(\Omega') \subseteq dom(\Omega'')$. By transitivity, we establish the desired result $dom(\Omega) \subseteq dom(\Omega'')$.

Case [T-SEQ-D]**.** This case is similar to T-SEQ.

Case [T-∨-SUCC, T-LET, T-ASSIGN, T-IF, T-ENOUGH?]**.** In each case, we apply the inductive hypothesis to learn that $dom(\Omega) \subseteq dom(\Omega')$.

□

Theorem 6.2 (Preservation for Commands) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$$

and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c$ *is well-formed,* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$*,* $dom(\mathsf{N}_{\mathsf{un}}) \subseteq dom(\mathsf{V})$*, and for some (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$$

then for some Ω_0

$$\mathtt{Md} \mid b \ \geq \ 0 : \mathit{nat} \mid \Omega_0; \Sigma \vdash_{\mathtt{Sig}} c' : \tau \dashv \Omega'$$

where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' : \Omega_0 \mid \Sigma$*,* $\Omega > \Omega_0$*, and* $n' \geq 0$*. Moreover* $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$ *is well-formed. Moreover,* $dom(\mathsf{N}) = dom(\mathsf{N}')$*, and if* $\mathtt{Md} = \mathsf{aID}(c_0)$*, then* $\mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$.

Proof: The proof is by induction on the size of c. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c \ \rightarrow \ \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c'$.

Case 1 [D-LET-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e \,\mathbf{in}\, c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e' \,\mathbf{in}\, c} \ (\text{D-LET-STEP})$$

By the first premise $n > 0$. By inversion on (†) via T-ENOUGH? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \ > \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \ = \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}'' = \{\mathtt{Md} \mid b \ \geq \ 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau\}$.

By inversion on ($\dagger_1$) via T-LET

$$\frac{\begin{array}{c}\mathtt{Md} \mid \ b \ \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \mathsf{C}^{\mathtt{Md}}_{A} \\ \mathtt{Md} \mid \ b \ \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \ @ \ \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid \ b \ > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau \dashv \Omega'} \ (\text{T-LET})$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$

By the application of Theorem 17 to (1) and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}},$$

and $n' \geq 0$. By the following application of the T-Let rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau \dashv \Omega'} \text{ (T-Let)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau^s\}$. It follows via lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau^s$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau^s\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e' \,\mathbf{in}\, c : \tau \dashv \Omega'} \text{ (T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e' \,\mathbf{in}\, c$ follows by definition and $\Omega > \Omega$ follows by definition , both vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired.

Case 2. [D-Let-v]**.**

$$\frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \ \mathit{fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e \,\mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell \mathbin{@} \mathtt{Ck} \hookrightarrow e \mid c} \text{ (D-Let-v)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{let}\, x = e \,\mathbf{in}\, c : \tau.$$

By inversion of ($\dagger_1$) via T-Let

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash c : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau \dashv \Omega'}\ \text{(T-Let)}$$

we learn that

(1) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A \mathbin{@} \mathtt{Ck} \vdash c : \tau \dashv \Omega'$

The typing judgment (2) completes the proof, if we can show that $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}, \ell @ \mathtt{Ck} \hookrightarrow e : \Omega \mid \Sigma, x{:}{\downarrow}{\uparrow}A @ \mathtt{Ck}$.

By $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid\ \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = {\downarrow}\Sigma'$.

This is enough to prove $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V}, \ell @ \mathtt{Ck} \hookrightarrow e : \Omega \mid \Sigma, x{:}{\downarrow}{\uparrow}A @ \mathtt{Ck}$.

Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell @ \mathtt{Ck} \hookrightarrow e \mid c$ follows by definition vacuously because c is not of the form $c';_W c''$. Observe that $\Omega > \Omega$ follows by definition vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired.

Case 3 [D-Assign-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'}\ \text{(D-Assign-step)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash x := e : \tau\}$.

By inversion on ($\dagger_1$) via T-Assign

$$\frac{\begin{array}{c}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{WT}} x : {\downarrow}{\uparrow} A \dashv \Omega'\end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e\ : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'}\ \text{(T-Assign)}$$

we have

- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ and
- $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'$.

By the application of Theorem 17 on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}},$$

and $n' \geq 0$. By the following application of the T-Assign rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'} \text{ (T-Assign)}$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau\}$. It follows by lemma 30 that $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_1} x := e' : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e' : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} x := e' : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega'} \text{ (T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'$ follows by definition vacuously. The detail $\Omega > \Omega$ follows by definition , vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired.

Case [D-Assign-V]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \rightarrow \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\texttt{Sig}'' = \mathit{if}\ \texttt{Md} = \mathsf{jit}\ \mathit{then}\ \texttt{Sig}'\ \mathit{else}\ \texttt{Sig}$ and $\texttt{Sig}' = \{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash x := e : \tau\}$.

By inversion on $(\dagger_1)$ via T-Assign

$$\frac{\begin{array}{c}\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \mathsf{C}_A^{\texttt{Md}} \\ \texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega'\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} x := e : \mathsf{C}_{\texttt{unit}}^{\texttt{Md}} \dashv \Omega'}\ \text{(T-Assign)}$$

we have

(1) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \mathsf{C}_A^{\texttt{Md}}$

(2) $\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega'$

By V-Loc and the definition of V, we have that $\vdash_\gamma^{\texttt{Md}} \mathsf{N} \mid \mathsf{V}', (\ell@\texttt{Ck} \hookrightarrow v') : \Omega \mid \Sigma_0, (x :\downarrow\uparrow A@\texttt{Ck})$ where $\Sigma = \Sigma_0, (x :\downarrow\uparrow A@\texttt{CK})$. It follows by inversion on T-w-Shift and T-Loc-Write

$$\frac{\dfrac{\begin{array}{cc}\Sigma = \downarrow\Sigma' & \Omega = \Omega_0, \Omega_{1\,\texttt{CK}} \\ \Omega_0, \Sigma' = x : \uparrow A@\texttt{CK}, \Omega_2' = \Omega'' & \texttt{CK} = \delta(\texttt{CK}, Wt) \neq \mathit{UN}\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega_0, \Sigma' \vdash_{\texttt{WT}} x : \uparrow A \dashv \Omega''}\ \text{(T-Loc-Write)}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{WT}} x : \downarrow\uparrow A \dashv \Omega'' \restriction \mathit{dom}(\Omega_0), \Omega_{1\,\texttt{CK}}}\ \text{(T-w-Shift)}$$

that

- $\Omega' = \Omega'' \restriction \mathit{dom}(\Omega_0), \Omega_{1\,\texttt{CK}}\ (\exists \Omega'')$
- $\Omega'' = \Omega_0, \Sigma'$
- $\Sigma = \downarrow\Sigma'$
- $\Omega = \Omega_0, \Omega_{1\,\texttt{CK}}$

By definition of δ on CK, we know that $\Omega = \Omega'$.

By the premise $\mathit{Val}(\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} e : \mathsf{C}_A^{\texttt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\texttt{Md} \mid b : \texttt{nat} \mid \Omega_0, \Sigma' \vdash_{\texttt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$.

Appying V-Loc, we can show that $\vdash_\gamma^{\texttt{Md}} \mathsf{N} \mid \mathsf{V}', (\ell@\texttt{Ck} \hookrightarrow e) : \Omega \mid \Sigma_0, (x :\downarrow\uparrow A@\texttt{Ck})$ where $\Sigma' = \Sigma_0, (x :\downarrow\uparrow A@\texttt{Ck})$.

Furthermore, it follows by $\Omega = \Omega'$ that

$$\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} \mathbf{skip} : \mathsf{C}_{\texttt{unit}} \dashv \Omega.$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\}$. From lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega}\ \text{(T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}$ follows by definition vacuously, and $\Omega > \Omega$ follows by definition , both vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired. Lastly, note that $\mathit{dom}(\mathsf{V}) = \mathit{dom}(\mathsf{V}', \ell@q' \hookrightarrow e)$.

Case [D-Assign-N]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, Wt) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}}\ \text{(D-Assign-N)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash x := e : \tau\}$.

By inversion on ($\dagger_1$) via T-Assign

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} x := e : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'}\ \text{(T-Assign)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'$

By inversion on (2) via T-w-Shift and T-Loc-Write:

$$\dfrac{\dfrac{\dfrac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega_0, \Omega_{1\,\mathsf{un}} \qquad \Omega_0, \Sigma' = x : \uparrow A@q, \Omega_2' \qquad \Omega'' = x : \uparrow A@q', \Omega_2' \qquad q' = \delta(q, Wt) \neq UN}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0, \Sigma' \vdash_{\mathtt{WT}} x : \uparrow A \dashv \Omega''}\ \text{(T-Loc-Write)}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{WT}} x : \downarrow\uparrow A \dashv \Omega'' \restriction dom(\Omega)}\ \text{(T-w-Shift)}$$

we have

(i) $\Omega' = \Omega'' \restriction dom(\Omega)$ $(\exists \Omega'')$

(ii) $\Sigma = \downarrow\Sigma'$

(iii) $\Omega = \Omega_0, \Omega_{1\,\mathsf{un}}$

(iv) $\Omega_0, \Sigma' = x : \uparrow A@q, \Omega_2'$

(v) $\Omega'' = x : \uparrow A@q', \Omega_2'$

(vi) $q' = \delta(q, Wt) \neq UN$

By $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega, \Sigma' \vdash_{\mathtt{RD;Sig}} e : \uparrow A,$$

where $\Sigma = \downarrow\Sigma'$. This is enough to prove $\vdash_\gamma^{\mathtt{Md}} (\ell@\mathtt{Ck} \hookrightarrow e, \mathsf{N}'') \mid \mathsf{V} : (x : \uparrow A@q', \Omega_2') \mid \Sigma$. From here, we note that since $x : \uparrow A@q$, we have $x : \uparrow A@q \in \Omega_0$. Therefore, $\Omega'' = \Omega^0, \Sigma'$ for some $\Omega^0 < \Omega_0$ (i.e., $\Omega^0 = \Omega_0'', x : \uparrow A@q'$ where $\Omega_0 = \Omega_0'', x : \uparrow A@q$).

We now need to show that:

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega).$$

Let $\Omega' = \Omega^0, \Omega^1_{\mathsf{un}}$. By T-Skip, T-C-Shift, and T-$\vee$-Succ, and recalling that $\Omega' = \Omega'' \restriction dom(\Omega)$ (as defined above), we have

$$\dfrac{\dfrac{\dfrac{\Sigma = \downarrow\Sigma' \qquad \Omega' = \Omega^0, \Omega^1_{\mathsf{un}}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^0, \Sigma' \vdash_{\mathtt{Sig}} \mathbf{skip} : \uparrow\mathtt{unit} \dashv \Omega''}\ \text{(T-Skip)}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'' \restriction dom(\Omega')}\ \text{(T-C-Shift)}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction dom(\Omega')}\ \text{(T-}\vee\text{-Succ)}$$

We now need to show that $dom(\Omega') = dom(\Omega)$. From (i), we have that $dom(\Omega') \subseteq dom(\Omega)$. By Lemma 39, we have that $dom(\Omega) \subseteq dom(\Omega')$.

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\}$. By lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}_2} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction \mathit{dom}(\Omega')\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'' \restriction \mathit{dom}(\Omega')}\ (\text{T-ENOUGH?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$ follows by definition vacuously. The detail $\Omega > \Omega'$ follows by definition , (iv) and (v) from above, and the premise $q' = \delta(q, Wt) \neq \mathtt{UN}$. If $\mathtt{Md} = \mathtt{aID}(c_0)$, we want to show that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = (\mathsf{N}', \ell@q' \hookrightarrow e) \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$. By definition, $\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v'$. Since ℓ is written to, $q' = \mathtt{Wtn}$, and either $q = \mathtt{Wtn}$ or $q = \mathtt{MFstWt}$. Now we need to show that $\mathsf{N}', \ell@q \hookrightarrow v' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$. From above, this reduces to showing $\mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N}' \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$ which holds trivially. Lastly, note that $\mathit{dom}(\mathsf{N}) = \mathit{dom}(\mathsf{N}', \ell@q' \hookrightarrow e)$.

Case 6 [D-If]**.**

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e'\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2}\ (\text{D-If})$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau\}$.

By inversion on ($\dagger_1$) via T-If

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD};\mathtt{Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau \dashv \Omega'}\ (\text{T-If})$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{bool}$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega'$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'$.

By the application of Theorem 17 on (1) and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_{bool},$$

and $n' \geq 0$. By the following application of the T-If rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_{bool} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'} \text{ (T-If)}$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau\}$. By lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_1} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau^s$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathtt{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'} \text{ (T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2$ follows by definition and $\Omega > \Omega$ follows by definition , both vacuously. Additionally, observe that $\mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\} = \mathsf{N} \setminus \{\mathtt{MFstWt}, \mathtt{Wtn}\}$, as desired.

Case 7. [D-If-tt]**.**

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \text{ (D-If-tt)}$$

By the first premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau$$

where $\texttt{Sig}'' = \mathit{if}\ \texttt{Md} = \mathsf{jit}\ \mathit{then}\ \texttt{Sig}'\ \mathit{else}\ \texttt{Sig}$ and $\texttt{Sig}' = \{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash \textbf{if}\ \texttt{tt}\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau\}$.

By inversion on $(\dagger_1)$ via T-If

$$\frac{\begin{array}{c}\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} \texttt{tt} : \mathsf{C}^{\texttt{Md}}_{bool} \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1 : \tau \dashv \Omega' \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega'\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} \textbf{if}\ \texttt{tt}\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau \dashv \Omega'}\ \text{(T-If)}$$

we learn that

(1) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{RD;Sig}} \texttt{tt} : \mathsf{C}^{\texttt{Md}}_{bool}$

(2) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1 : \tau \dashv \Omega'$

(3) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega'$

Observe that the well-formedness of $\gamma \mid \texttt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1$ follows by definition vacuously because c_1 is not of the form $c';_W c''$. The detail $\Omega > \Omega$ follows vacuously by definition .

The typing judgment (2) completes the proof, because $\vdash^{\texttt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$ holds by assumption. Additionally, observe that $\mathsf{N} \setminus \{\texttt{MFstWt}, \texttt{Wtn}\} = \mathsf{N} \setminus \{\texttt{MFstWt}, \texttt{Wtn}\}$, as desired.

Case 8 [D-If-ff]**.** Similar to the previous case.

Case 9 [D-seq]**.**

$$\frac{n > 0}{\gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1; c_2 \rightarrow \gamma \mid \texttt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2}\ \text{(D-seq)}$$

By the premise $n > 0$. By inversion on $(\dagger)$ via T-Enough? rule, we have

$$(\dagger_1) \quad \texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1; c_2 : \tau \dashv \Omega''$$

and

$$(\dagger_2) \quad \texttt{Md} \mid b = 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}''} c_1; c_2 : \tau$$

where $\texttt{Sig}'' = \mathit{if}\ \texttt{Md} = \mathsf{jit}\ \mathit{then}\ \texttt{Sig}'\ \mathit{else}\ \texttt{Sig}$ and $\texttt{Sig}' = \{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash c_1; c_2 : \tau\}$.

By inversion on $(\dagger_1)$ via T-Seq

$$\frac{\begin{array}{c}\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1 : \mathsf{C}_{\texttt{unit}} \dashv \Omega' \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega'; \Sigma \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega''\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \vdash_{\texttt{Sig}} c_1; c_2 : \tau \dashv \Omega''}\ \text{(T-Seq)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

Let $W = \gamma \mid \mathsf{V}$. To finish the proof, it remains to be shown that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$$

where $\Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2$ and $\Sigma''' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{un}}$.

By definition of `trim` and the world W, the desired result holds from (2).

Let $x \in \mathsf{dom}(\Sigma')$ where $[x \mapsto \ell] \in \gamma$ be arbitrary such that $\ell \notin \mathsf{dom}(\mathsf{V})$.

By the following application of the T-SEQ-D rule

$$\text{T-SEQ-D}\ \frac{\begin{array}{c} W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega' \qquad \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2 \\ \Sigma''' = \mathtt{trim}(\Sigma, \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{un}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma''' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''}$$

we learn that $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''$

We consider two subcases based on `Md`.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 ;_W c_2 : \tau\}$. By lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 ;_W c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 ;_W c_2 : \tau$.

In both subcases, the desired result follows by T-ENOUGH?:

$$\frac{\begin{array}{c} \mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 ;_W c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}_2} c_1 ;_W c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''}\ (\text{T-ENOUGH?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 ;_{\gamma|\mathsf{V}} c_2$ follows by definition vacuously.

Case [D-SEQ-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 ;_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1' ;_W c_2}\ (\text{D-SEQ-STEP})$$

The premises yield

(a) $n > 0$

(b) $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$

By assumption, note that

- $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega'$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 ;_W c_2$ is well-formed.

Put $W = \gamma_0 \mid \mathsf{V}_0$. Then by definition we have $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$. Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1$ is well-formed, which vacuously follows by definition because c_1 does not have the form $c' ;_{W'} c''$ and we always run the command c_1 before c_2.

By inversion of $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega'$ via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}''} c_1 ;_W c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}'' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c_1 ;_W c_2 : \tau\}$.

By inversion of $(\dagger_1)$ via T-seq-d

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega' \qquad \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2 \\ \Sigma'' = \mathtt{trim}(\Sigma, \mathsf{V}_0, \gamma_0) \cup \downarrow\Omega^1_{\mathsf{un}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''} \text{(T-seq-d)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathsf{unit}} \dashv \Omega'$

(2) $\Sigma'' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''$

(4) $\Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2$

By the inductive hypothesis applied to (1), (b), the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1$, $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, and $n \geq 0$, we get $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathsf{unit}} \dashv \Omega'$, where

(i) $\vdash_{\gamma'}^{\mathtt{Md}} \mathsf{N}' \mid \mathsf{V}' : \Omega_0 \mid \Sigma$,

(ii) $n' \geq 0$,

(iii) $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$ is well-formed,

We now need to show that $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$. Observe that $c_1 \neq c' ;_W c''$ because $c_1 ;_W c_2$ and we always run c_1 first. By lemma 38 and $c_1 \neq c' ;_W c''$, it follows that $dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$ and $\gamma \subseteq \gamma'$. Then it follows by $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$ (as shown above), that $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma \subseteq \gamma'$.

Using $W = \gamma_0 \mid \mathsf{V}_0$, (2), (3), and $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathsf{unit}} \dashv \Omega'; \Sigma'$, we can apply T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathsf{unit}} \dashv \Omega' \\ \Sigma'' = \mathsf{trim}(\Sigma_0, \mathsf{V}_0, \gamma_0) \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash_{\mathtt{Sig}} c_1';_W c_2 : \tau \dashv \Omega''} \text{ (T-SEQ-D)}$$

We now need to show that $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash_{\mathtt{Sig}} c_1';_W c_2 : \tau$. The proof proceeds in two subcases based on Md:

Case $\mathtt{Md} = \mathsf{jit}$**.** Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau\}$. It follows by lemma 30 that $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau$.

Case $\mathtt{Md} = \mathsf{aID}(c_0)$**.** By $(\dagger_2)$ via lemma 31, we can see that $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau$.

In both cases, $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau$.

The desired result follows by T-ENOUGH?:

$$\frac{\begin{array}{c} \mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau \\ \mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau \dashv \Omega'' \end{array}}{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \vdash c_1';_W c_2 : \tau \dashv \Omega''} \text{ (T-ENOUGH?)}$$

Observe that by definition applied to $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$ (as shown above), $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2$ is well-formed, as desired.

Case 11 [D-SEQ-V]**.**

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \mathbf{skip};_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2} \text{ (D-SEQ-V)}$$

Observe that $n = n' + 1$ (from the premise of D-SEQ-V) implies $n > 0$, and hence $b > 0$.

By assumption, we have

$$\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash \mathbf{skip};_W c_2 : \tau \dashv \Omega'$$

where $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega$.

By inversion on T-SEQ-D, we have

(1) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \vdash \mathbf{skip} : \mathsf{C}_{\mathsf{unit}} \dashv \Omega_0$

(2) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma \vdash c_2 : \tau \dashv \Omega'$

(3) $W = \gamma' \mid \mathsf{V}'$ (from the premise of D-SEQ-V)

(4) $\Sigma'_0 = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma') \cup \downarrow\Omega^1_{\mathsf{un}}$

(5) $\Omega_0 = \Omega^1_{\mathsf{CK}}, \Omega^2$

We now need to show that $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' : \Omega_0 \mid \Sigma$.

Since $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{skip};_W c_2$ is well-formed (by assumption), it follows by definition that $\gamma' \subseteq \gamma$.

By inversion of (1) on T-C-Shift and then on T-Skip, we know that $\Omega = \Omega_0$. Hence, it follows by assumption that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega_0 \mid \Sigma(*)$.

Therefore, the desired result follows by lemma 37 applied to (*), the premise $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$, (4), and $\gamma''' \subseteq \gamma$.

Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2$ follows by definition , vacuously because c_2 is not of the form $c';_W c''$.

Note that $dom(\mathsf{N}) = dom(\mathsf{N}')$ holds by observation, and by the inductive hypothesis in case 10 (D-seq-step). □

Lemma 25 *If* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \mathsf{V} \mid c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}'_1 \mid \mathsf{V}' \mid c'$ *and* $\mathsf{N}_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$*, then* $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2 \mid \mathsf{V}' \mid c'$ *and* $\mathsf{N}'_1 \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\mathtt{MFstWt}\}$.

Proof: The proof is by induction on the size of c. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}_1 \mid \mathsf{V} \mid c \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}'_1 \mid \mathsf{V}' \mid c'$.

Case 1 [D-Let-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e'\, \mathbf{in}\, c} \text{ (D-Let-step)}$$

By inversion on D-Let-step, we have:

- $n > 0$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By applying the inductive hypothesis, we have

- $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e'$
- $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$

Observing that $\infty > 0$, we apply D-Let-step to learn that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{let}\, x = e'\, \mathbf{in}\, c$$

Case 2. [D-Let-v]**.**

$$\frac{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell \textit{ fresh} \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c} \text{ (D-Let-v)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By inversion on D-Let-v, we have

(1) $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$

(2) $\gamma' = \gamma, [x \mapsto \ell]$

(3) ℓ *fresh*

(4) $n = n' + 1$

From $n = n' + 1$, we know that $n > 0$. Thus, $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$ is not a crash value (i.e. $n \neq 0$). Therefore, $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$ since $\infty > 0$. By application of D-Let-v to $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$, (2), (3), and $\infty > 0$, we have

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \to \gamma' \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V}, \ell@\mathtt{Ck} \hookrightarrow e \mid c$$

Case 3 [D-Assign-step]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid x := e'} \text{ (D-Assign-step)}$$

By inversion on D-Assign-step, we have

- $n > 0$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By applying the inductive hypothesis, we have

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e'$$

where $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$.

Observing that $\infty > 0$, we apply D-Assign-step to learn that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e'$$

Case 4 [D-Assign-V]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}} \text{ (D-Assign-V)}$$

The proof is analogous to the argument of Case 2 (D-Let-v).

Case 5 [D-Assign-N]**.**

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e) \qquad \mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v' \\ q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN} \qquad \gamma = \gamma', [x \to \ell] \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid x := e \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}} \text{ (D-Assign-N)}$$

By inversion of D-Assign-N, we learn that

(1) $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid e)$

(2) $\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v'$

(3) $q' = \delta(q, \mathtt{wt}) \neq \mathtt{UN}$

(4) $\gamma = \gamma', [x \to \ell]$

(5) $n = n' + 1$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. We want to show that $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e \ \to\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2, \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$ and $\mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2, \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\}$ where $\mathsf{N}_2 = \mathsf{N}'_2, \ell@q \hookrightarrow v'$.

From (5), we know that (1) is not a crash value (i.e. $n > 0$, and hence $n \neq 0$). Since $\infty > 0$, we have $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$. By application of D-Assign-NV to $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid e)$, $\mathsf{N}_2 = \mathsf{N}'_2, \ell@q \hookrightarrow v'$, (3), (4), and $\infty > 0$, we have

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid x := e \ \to\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}'_2, \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$$

Towards $\mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2, \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\}$, observe that the assumption $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$ implies $\mathsf{N}' \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2 \setminus \{\mathtt{MFstWt}\}$. Thus, it follows that $\mathsf{N}', \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\} = \mathsf{N}'_2, \ell@q' \hookrightarrow e \setminus \{\mathtt{MFstWt}\}$, as desired.

Case 6 [D-If]**.**

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \ e \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid e'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ e'\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2} \ \text{(D-If)}$$

The proof is similar to the proof of Case 3 (D-Assign-step).

Case 7. [D-If-tt]**.**

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \ \text{(D-If-tt)}$$

By inversion on D-If-tt, we have

(1) $n = n' + 1$

(2) $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{tt})$

From (1), we know that $n > 0$, and hence (2) is not a crash value. Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. Observing that $\infty > 0$, we have that $\mathit{Val}(\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathtt{tt})$. By applying D-If-tt, we have that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \ \to\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c_1$$

Case 8. [D-IF-FF]**.**

$$\frac{n = n' + 1 \qquad Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \mathtt{ff})}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid \textbf{if}\ \mathtt{ff}\ \textbf{then}\ c_1\ \textbf{else}\ c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V} \mid c_1} \text{ (D-IF-FF)}$$

The proof is similar to the previous case (D-IF-TT).

Case 9 [D-SEQ]**.**

$$\frac{n > 0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1; c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2} \text{ (D-SEQ)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. Observing that $\infty > 0$, it follows by application of D-SEQ that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid\ c_1; c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c_1;_{\gamma \mid \mathsf{V}} c_2$$

Case 10 [D-SEQ-STEP]**.**

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ c_1;_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1';_W c_2} \text{ (D-SEQ-STEP)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By inversion on D-SEQ-STEP, we have

- $n > 0$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}' \mid \mathsf{V}' \mid c_1'$

By the inductive hypothesis, we have

(1) $\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid c_1 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}' \mid c_1'$

(2) $\mathsf{N}' \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2' \setminus \{\mathtt{MFstWt}\}$

From (1) and the observation $\infty > 0$, we apply D-SEQ-STEP to learn that

$$\gamma \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2 \mid \mathsf{V} \mid\ c_1;_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid \infty \mid \mathsf{N}_2' \mid \mathsf{V}' \mid c_1';_W c_2$$

Case 11 [D-SEQ-V]**.**

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N} \mid \mathsf{V} \mid\ \textbf{skip};_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}'' \mid c_2} \text{ (D-SEQ-V)}$$

Let $\mathsf{N} \setminus \{\mathtt{MFstWt}\} = \mathsf{N}_2 \setminus \{\mathtt{MFstWt}\}$. By inversion on D-SEQ-V, we have

- $W = \gamma' \mid \mathsf{V}'$
- $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$

Observe that $\infty > 0$. By applying D-SEQ-V, we have

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{N}_2 \mid \mathsf{V} \mid \mathbf{skip};_W c_2 \;\rightarrow\; \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{N}_2 \mid \mathsf{V}'' \mid c_2$$

□

Appendix D

Inputs and Outputs Appendix

D.1 Syntax

We summarize our system syntax below. Values are standard, including integers, boolean primitives, and variables. Expressions include values and binary operations.

To support outputs, we introduce an output identiifier *ID* which is a number uniquely assigned to each output in an atomic region. If *ID* is in the set of user-declared idempotent outputs O, then its output must be an idempotent value and its effect will be buffered to the end of the atomic region.

Commands include skip instructions, conditional branching, assignments, sequences of commands, outputs identified by an *ID* and taking a value v to output, let bindings of expressions, and let bindings of sensor inputs. $\mathsf{end}_{\mathsf{if}}$ and all of the $@q$ annotated commands are runtime commands. As before, programs include JIT and atomic regions, identified by a unique identifier aID and equipped with a variable policy *aPol* for the region and a set of output identifiers O. In this system, the atomic region variable policy simply includes the sets of checkpointed ω and nonidempotent *nIds* variables.

The reason that these variable sets are sufficient stems from the fact that this type system combines checking with inference, where restrictions on variable accesses are learned throughout the type checking. Therefore, access qualifiers for atomic regions describe how variables have been accessed according to key memory access patterns: checkpointed (`CK`), not read yet (`NRD`), first read on *some* path (`mayFstRD`), written in a tainted branch on *all* paths (`MTntWt`), written on *some* path but not all (`mayTntWt`), and written on *all* paths (`MFstWt`). Idempotence qualifiers indicate whether a variable is nonidempotent Nid or idempotent Id. Idempotent variables accessed within an atomic region are embellished with an access mode qualifier describing why it is idempotent. Taintedness qualifiers indicate whether a variable could store an input-dependent value at a given point of the type checking. We refer to input-dependent as "tainted" (`Tnt`) and non input-dependent as "non-tainted" (`Nt`). Variables are annotated with a tuple of idempotence qualifier and taintedness qualifier which evolve over the course of type checking a program.

The definitions of instructions, statements, and configurations are similar to the undo-logging interpretation with a few key differences: (1) configurations are extended a stream of sensor

inputs, each of the form $\mathsf{in}(v)$ and a queue of delayed outputs that buffers idempotent outputs until the end of an atomic region (they are evaluated from an expression to a value, and then buffered to O); (2) certain value configurations are annotated with a qualifier, which will aid in formulating input-dependent conditional branching; (3) the atomic region execution mode $\mathsf{aID}(c, \mathcal{O})$ must now carry the set of idempotent output identifiers $\mathcal{O}$, to be used when type checking the region; (4) configurations include a stack of *pc*, each of the form $\langle qID, qIO \rangle$, representing both the taint and idempotence level of the command or expression that is running (a stack is needed to enforce proper scoping for conditionals); (5) the continuation in a crash instruction is annotated with *pcS*, which must be remembered to resume execution for JIT regions.

Programs, commands, and expressions

values	v	::=	$\mathsf{n} \mid \mathsf{tt} \mid \mathsf{ff} \mid x$
exprs	e	::=	$v \mid e_1 \odot e_2 \mid e_1 \odot e_2 @ \{q\}$
output id	ID	::=	n
cmds	c	::=	$\mathsf{skip} \mid \mathsf{let}\, x = e\, \mathsf{in}\, c \mid \mathsf{let}\, x = e@q\, \mathsf{in}\, c \mid \mathsf{let}\, x = \mathsf{IN}()\, \mathsf{in}\, c$
			$\mid \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \mid \mathsf{if}\, e@q\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \mid x := e \mid x := e@q$
			$\mid c_1; c_2 \mid c_1 ;_W c_2 \mid \mathsf{output}(ID, e) \mid \mathsf{output}(ID, e@q) \mid \mathsf{end}_{\mathsf{if}}$
idempotent output ids	$\mathcal{O}$	::=	$\cdot \mid ID, \mathcal{O}$
progs	p	::=	$\mathsf{skip} \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol, \mathcal{O}); c; \mathsf{end}_{\mathsf{atom}}; p \mid c; p$
atomic region policy	$aPol$	::=	$(\omega, nIds)$

Qualifiers

access qualifier	$qAcc$	::=	$\mathtt{CK} \mid \mathtt{NRD} \mid \mathtt{mayFstRD} \mid \mathtt{MTntWt} \mid \mathtt{mayTntWt} \mid \mathtt{MFstWt}$
idempotence qualifier	qID	::=	$\mathsf{Id} \mid \mathsf{Id}(qAcc) \mid \mathsf{Nid}$
taintedness qualifier	qIO	::=	$\mathtt{Tnt} \mid \mathtt{Nt}$
qualifier	q	::=	$\langle qID, qIO \rangle$

Instructions, statements, and configurations

continuations	κ	::=	$c \mid e$
statements	s	::=	$\kappa \mid i \mid p$
crash instrs	i	::=	$\varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow\kappa@pcS) \mid \downarrow\uparrow\kappa@pcS$
energy level	g	::=	$\cdot \mid n$
charge stream	χ	::=	$n :: \chi$
nonvolatile mem	N	::=	$\cdot \mid \ell \,@\, q \hookrightarrow v, \mathsf{N} \mid \ell_{\mathsf{ck}} \,@\, \langle \mathsf{Id}(\mathtt{CK}), qIO \rangle \hookrightarrow v, \mathsf{N}$
volatile mem	V	::=	$\cdot \mid \ell \,@\, q \hookrightarrow v, \mathsf{V} \mid \ell \hookrightarrow v, \mathsf{V}$
checkpoint ctx	K	::=	$(\mathsf{N}_k; \mathsf{V}_k)$
var loc map	γ	::=	$\cdot \mid \gamma, x \mapsto \ell$
output buffer	O	::=	$\cdot \mid \mathsf{O} :: (\mathsf{out}(ID, v))$
input stream	I	::=	$\mathsf{in}(v) :: \mathsf{I}$
program counter	pc	::=	$\langle qID, qIO \rangle$
pc. stack	pcS	::=	$\cdot \mid pc \rhd pcS$
open config	C_o	::=	$(\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} \mid pcS \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{K} \mid pcS \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid s)$
			$\mid (\gamma \mid \mathtt{Md} \mid g \mid \mathsf{I} \mid \mathsf{N} \mid \mathsf{K} \mid p)$
closed config	C_c	::=	$[\chi \rhd \varepsilon] \otimes C_o$
exec. mode	$\mathtt{Md}$	::=	$\mathsf{aID}(c, \mathcal{O}) \mid \mathsf{jit}$

D.1.1 Qualifier structure and operations

Lattice structures. Since checking each branch of a conditional separately could result in different qualifiers, we need to define a function to merge them at the end of a conditional. We define the join of two typing contexts $\sqcup$ as the pairwise join of taintedness and idempotence qualifiers. The definitions for join over Ω and Σ are similar.

$$\textsc{EMPTY-S}\ \frac{}{\cdot \sqcup \Omega = \Omega} \qquad \textsc{EMPTY-U}\ \frac{}{\cdot \sqcup \Sigma = \Sigma} \qquad \textsc{JOIN-S}\ \frac{}{(x : A@q_1, \Omega_1) \sqcup (x : A@q_2, \Omega_2) = x : A@q_1 \sqcup_q q_2, (\Omega_1 \sqcup \Omega_2)}$$

$$\textsc{JOIN-U}\ \frac{}{(x : A@q_1, \Sigma_1) \sqcup (x : A@q_2, \Sigma_2) = x : A@q_1 \sqcup_q q_2, (\Sigma_1 \sqcup \Sigma_2)}$$

$$\textsc{JOIN-Q}\ \frac{qID' = qID_1 \sqcup_{id} qID_2 \qquad qIO' = qIO_1 \sqcup_t qIO_2}{\langle qID_1, qIO_1\rangle \sqcup_q \langle qID_2, qIO_2\rangle = \langle qID', qIO'\rangle}$$

We will always have the invariants $\mathsf{dom}(\Omega_1) = \mathsf{dom}(\Omega_2)$ and $\mathsf{dom}(\Sigma_1) = \mathsf{dom}(\Sigma_2)$, because all nonvolatile locations are pre-defined and any variables bound in one branch of a conditional but not the other will reach the end of their scope before the end of the conditional, where the join would occur.

We write $\overline{\mathsf{Id}} = \overline{\mathsf{Id}(qAcc)} = \mathsf{Id}$ and $\overline{\mathsf{Nid}} = \mathsf{Nid}$ to extract the idempotence level from an idempotence qualifier, and note that $\overline{\mathsf{Id}(\_)} \sqsubseteq_{id} \mathsf{Nid}$ where join is denoted as $\sqcup_{id}$. We define $\mathsf{Id}(qAcc_1) \sqcup_{id} \mathsf{Id}(qAcc_2) = \mathsf{Id}(qAcc_1 \vee_a qAcc_2)$. Access qualifiers are ordered as follows: $\texttt{MFstWt} \leq_a \texttt{NRD}$, $\texttt{NRD} \leq_a \texttt{mayFstRD}$, $\texttt{NRD} \leq_a \texttt{mayTntWt}$ and $\texttt{MTntWt} \leq_a \texttt{mayTntWt}$, where join is denoted as $\vee_a$. `CK` is not included in this lattice since there will never be a case where a conditional ends with an idempotent variable being marked as `CK` on one branch and a different access qualifier on the other.

The taintedness qualifiers `Nt` and `Tnt` also form a lattice where $\texttt{Nt} \sqsubseteq_t \texttt{Tnt}$. Join is denoted $\sqcup_t$.

We write $\Omega_1 \sqsubseteq \Omega_2$ to denote that for every variable in Ω_1, the qualifier is less than the qualifier in Ω_2, with respect to idempotence and taint.

δ access transition function. To evolve an idempotence qualifier for a given sequential access occurring under a *pc* context, we define the transition function: $\delta(pc, qID, a){=}qID'$. An access a is either a read *Rd* or write *Wt*.

$$\delta(pc, \mathsf{Id}(\texttt{NRD}), Rd) = \delta(pc, \mathsf{Id}(\texttt{mayFstRD}), Rd) = \mathsf{Id}(\texttt{mayFstRD})$$
$$\delta(\texttt{Nt}, \mathsf{Id}(\texttt{NRD}), Wt) = \delta(\texttt{Nt}, \mathsf{Id}(\texttt{mayTntWt}), Wt) = \delta(pc, \mathsf{Id}(\texttt{Wtn}), a) = \mathsf{Id}(\texttt{Wtn})$$
$$\delta(\texttt{Tnt}, \mathsf{Id}(\texttt{NRD}), Wt) = \delta(\texttt{Tnt}, \mathsf{Id}(\texttt{mayTntWt}), Wt) = \delta(\texttt{Tnt}, \mathsf{Id}(\texttt{MTntWt}), a) = \mathsf{Id}(\texttt{MTntWt})$$
$$\delta(pc, \mathsf{Id}(\texttt{mayFstRD}), Wt) = \delta(pc, \mathsf{Id}(\texttt{mayTntWt}), Rd) = UN$$
$$\delta(pc, \mathsf{Id}(\texttt{CK}), a) = \mathsf{Id}(\texttt{CK}) \qquad \delta(pc, \mathsf{Id}, a) = \mathsf{Id} \qquad \delta(pc, \mathsf{Nid}, a) = \mathsf{Nid}$$

The two transitions that result in undefined *UN* would cause a program to fail type checking due to ill-checkpointed variables in problematic code patterns: WAR ($\delta(pc, \mathsf{ld}(\texttt{mayFstRD}), Wt)$) and RIO ($\delta(pc, \mathsf{ld}(\texttt{mayTntWt}), Rd)$). Our interpretation of a RIO bug varies slightly from previous systems. Here, it is a noncheckpointed variable that is written by some tainted branch, but not in all, and is later read by the program.

Note that it is impossible to have a case where $\delta(\texttt{Nt}, \mathsf{ld}(\texttt{MTntWt}), a)$ because in our system, we will never have a `MTntWt` outside of a tainted conditional (lift would have already changed the `MTntWt` qualifiers to `MFstWt` at the end of a tainted branch).

Lift function definition. Lift is applied at the end of merge, upon exiting a conditional. Upon entering an `Nt` context, it translates `MTntWt` qualifiers into `MFstWt`. Otherwise, it leaves qualifiers unchanged.

$$\mathsf{lift}(\texttt{Nt}, \mathsf{ld}(\texttt{MTntWt})) = \mathsf{ld}(\texttt{MFstWt})$$
$$\mathsf{lift}(pc, qID) = qID \qquad \textit{if } pc = \texttt{Tnt} \textit{ or } qID \neq \mathsf{ld}(\texttt{MTntWt})$$

D.2 Operational Semantic Rules

D.2.1 Value Configurations

Value configurations are listed below. Configurations of primitive values need to carry a qualifier to help with the taint propagation.

V-SKIP
$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{skip})}$$

V-NAT
$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{n}@q)}$$

V-BOOL-T
$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{tt}@q)}$$

V-BOOL-F
$$\frac{n > 0}{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{ff}@q)}$$

V-CRASH
$$\frac{}{\mathit{Val}(\gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \kappa)}$$

V-↓
$$\frac{}{\mathit{Val}([\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} \mid \downarrow\uparrow\varepsilon \# \mathsf{in}(b > 0, \kappa@pcS))}$$

V-↑
$$\frac{n > 0}{\mathit{Val}([\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{K} \mid \uparrow\varepsilon \# \mathsf{in}(b > 0, \kappa@pcS))}$$

V-# in
$$\frac{}{\mathit{Val}([\chi \rhd \varepsilon] \otimes \gamma \mid \mathtt{Md} \mid \cdot \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{K} \mid \varepsilon \# \mathsf{in}(b > 0, \kappa@pcS))}$$

V-P-DONE
$$\frac{n > 0}{\mathit{Val}([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{K} \mid pcS \mid \mathsf{skip})}$$

V-C-DONE
$$\frac{n > 0}{\mathit{Val}([\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathsf{K} \mid pcS \mid \mathsf{skip})}$$

D.2.2 Top-level Atomic Region Execution

The rules for executing atomic regions and JIT blocks are shown below. In D-P-CKPT, the premise $\mathsf{send}(\mathsf{O})$ is new, which plays the buffered idempotent outputs at the end of an atomic region. We assume that there is enough energy at the end of a program to process the output buffer.

$\boxed{K_c \Rightarrow_p K_c}$

D-P-CKPT

$$\frac{\begin{array}{c} n > 0 \qquad \mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{K}_0 \mid \gamma_0 \\ [\chi \rhd \varepsilon] \otimes \gamma_0 \mid \mathsf{aID}(c_0, O) \mid n \mid \mathsf{I} \mid \cdot \mid \mathsf{N}_0 \mid \cdot \mid \mathsf{K}_0 \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \mid c_0 \\ \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{aID}(c_0, O) \mid n' \mid \mathsf{I}' \mid \mathsf{O} \mid \mathsf{N}' \mid \cdot \mid \mathsf{K}_0 \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \mid \mathsf{skip} \\ \mathsf{send}(\mathsf{O}, n') = n'' \qquad n', n'' > 0 \qquad \mathsf{FinWorld}_d(\mathsf{N}') = \mathsf{N}'' \end{array}}{\begin{array}{c} [\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \cdot \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol, O); c_0; \mathsf{end}_{\mathsf{atom}}; p \\ \Rightarrow_p \ [\chi' \rhd \varepsilon] \otimes \gamma \mid n'' \mid \mathsf{I}' \mid \mathsf{N}'' \mid \cdot \mid p \end{array}}$$

D-P-SEQ

$$\frac{\begin{array}{c} n > 0 \qquad n' > 0 \qquad [\chi \rhd \varepsilon] \otimes \gamma \mid \mathsf{jit} \mid n \mid \mathsf{I} \mid \cdot \mid \mathsf{N} \mid \cdot \mid \cdot \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \mid c \\ \Rightarrow^* [\chi' \rhd \varepsilon] \otimes \gamma' \mid \mathsf{jit} \mid n' \mid \mathsf{I}' \mid \cdot \mid \mathsf{N}' \mid \mathsf{V}' \mid \cdot \mid pcS \mid \mathsf{skip} \end{array}}{[\chi \rhd \varepsilon] \otimes \gamma \mid n \mid \mathsf{I} \mid \mathsf{N} \mid \cdot \mid \ c; p \ \Rightarrow_p \ [\chi' \rhd \varepsilon] \otimes \gamma \mid n' \mid \mathsf{I}' \mid \mathsf{N}' \mid \cdot \mid p}$$

Definition 24 $\mathsf{InitWorld}_d(\mathsf{N}; aPol; \gamma) = \mathsf{N}_0 \mid \mathsf{K}_0 \mid \gamma_0$ *where* $aPol = (\omega, nIds)$ *iff*

- $\mathsf{N} = \mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_3$,
- $\mathsf{dom}(\mathsf{N}_1) = \mathsf{range}(\gamma \restriction nIds)$ *and* $\mathsf{dom}(\mathsf{N}_2) = \mathsf{range}(\gamma \restriction \omega)$,
- $\mathsf{N}_k, \gamma_k = \mathtt{unzip}(\{(\ell^* @ \mathtt{CK} \hookrightarrow v,\ x \mapsto \ell^*) \mid \mathsf{N}_2(\ell) = v,\ \ell^* \mathit{fresh}, x \in \omega \mathit{\ s.t.\ } \gamma(x) = \ell\})$,
- $\mathsf{N}_0 = \{\ell @ \langle \mathsf{Nid}, \mathtt{Nt} \rangle \hookrightarrow v \mid \mathsf{N}_1(\ell) = v\} \cup \{\ell_{\mathsf{un}} @ \langle \mathsf{Id}(\mathtt{CK}), \mathtt{Nt} \rangle \hookrightarrow v \mid \mathsf{N}_2(\ell) = v\} \cup \{\ell @ \langle \mathsf{Id}(\mathtt{NRD}), \mathtt{Nt} \rangle \hookrightarrow v \mid \mathsf{N}_3(\ell) = v\}$,
- $\gamma_0 = \{x_{\mathsf{un}} \mapsto \gamma(x)_{\mathsf{un}} \mid x \in \omega\} \cup \{x \mapsto \gamma(x) \mid x \notin \omega\} \cup \gamma_k$,
- $\mathsf{K}_0 = (\mathsf{N}_k; \emptyset)$

Definition 25 $\mathsf{FinWorld}_d(\mathsf{N}) = \mathsf{N}_f$ *iff*

- $\mathsf{N} = \mathsf{N}_{1\,\mathsf{un}}, \mathsf{N}_2$,
- $\mathsf{N}_f = \{\ell @ \langle \mathsf{Id}, \mathtt{Nt} \rangle \hookrightarrow v \mid \ell_{\mathsf{un}} @ \langle \mathsf{Id}(\mathtt{CK}), qIO \rangle \hookrightarrow v \in \mathsf{N}_1\} \cup \{\ell @ \langle \overline{qID}, \mathtt{Nt} \rangle \hookrightarrow v \mid \ell @ \langle qID, qIO \rangle \hookrightarrow v \in \mathsf{N}_2, qID \neq \mathsf{Id}(\mathtt{CK})\}$

D.2.3 Command Execution (Closed Config)

The execution rules for closed command configurations are shown below. When the device shuts off, *pcS* is treated like volatile memory. For JIT mode, it is lumped into the continuation, and in atomic mode it is lost. On restore in JIT mode, the stashed *pcS* will be brought back whereas in atomic mode it is reset to $\langle \mathsf{Id}, \mathtt{Nt} \rangle$. Note that the output buffer disappears when the device shuts off. This has the effect of resetting the buffer for atomic regions. Since outputs in JIT mode are posted immediately, this has no effect for programs running in JIT mode, i.e., the buffer will remain empty.

$\boxed{K_c \Rightarrow K_c}$

$$\frac{\gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{I} \,|\, \mathsf{O} \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, c \;\rightarrow\; \gamma' \,|\, \mathtt{Md} \,|\, n' \,|\, \mathsf{I}' \,|\, \mathsf{O}' \,|\, \mathsf{N}' \,|\, \mathsf{V}' \,|\, c'}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{I} \,|\, \mathsf{O} \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, c \Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma' \,|\, \mathtt{Md} \,|\, n' \,|\, \mathsf{I}' \,|\, \mathsf{O}' \,|\, \mathsf{N}' \,|\, \mathsf{V}' \,|\, \mathsf{K} \,|\, c'} \text{ (D-STEP)}$$

$$\frac{}{\begin{array}{c}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0, \mathcal{O}) \,|\, 0 \,|\, \mathsf{I} \,|\, \mathsf{O} \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \kappa \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0, \mathcal{O}) \,|\, \cdot \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \downarrow\uparrow c_0)\end{array}} \text{ (D-CRASH-AID)}$$

$$\frac{}{\begin{array}{c}[n :: \chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0, \mathcal{O}) \,|\, \cdot \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \downarrow\uparrow \kappa) \\ \Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0, \mathcal{O}) \,|\, n \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \downarrow\uparrow \kappa\end{array}} \text{ (D-CHARGE-AID)}$$

$$\frac{\mathsf{N} = \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{Wtn}}, \mathsf{N}'''_{\mathtt{CK}} \qquad \mathsf{K} = (\mathsf{N}_k; \emptyset)}{\begin{array}{c}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0, \mathcal{O}) \,|\, n \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \cdot \,|\, \mathsf{K} \,|\, \downarrow\uparrow \kappa \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{aID}(c_0, \mathcal{O}) \,|\, n \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N}'_{\mathtt{RD,MFstWt}}, \mathsf{N}''_{\mathtt{MFstWt}}, \mathsf{N}_{k\,\mathtt{ck}} \,|\, \cdot \,|\, \mathsf{K} \,|\, \kappa\end{array}} \text{ (D-RESTORE-AID)}$$

$$\frac{}{\begin{array}{c}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, 0 \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, c \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, \cdot \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow c)\end{array}} \text{ (D-CRASH-JIT)}$$

$$\frac{\mathsf{K} = (\cdot; \mathsf{V})}{\begin{array}{c}[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, \cdot \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V} \,|\, \mathsf{K} \,|\, \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \\ \Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa)\end{array}} \text{ (D-S-JIT)}$$

$$\frac{}{\begin{array}{c}[n :: \chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, \cdot \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\, \varepsilon \,\#\, \mathsf{in}(b > 0; \uparrow\kappa) \\ \Rightarrow [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathtt{Md} \,|\, n \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\, \uparrow\kappa\end{array}} \text{ (D-CHARGE-JIT)}$$

$$\frac{\mathsf{K} = (\cdot; \mathsf{V}_k)}{[\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, n \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{K} \,|\uparrow\; \kappa \;\Rightarrow\; [\chi \rhd \varepsilon] \otimes \gamma \,|\, \mathsf{jit} \,|\, n \,|\, \mathsf{I} \,|\, \cdot \,|\, \mathsf{N} \,|\, \mathsf{V}_k \,|\, \cdot \,|\, \kappa} \text{ (D-RESTORE-JIT)}$$

D.2.4 Command Execution (Open Config)

We present the operational semantics for commands below.

Notably, we include rules for propagating taint in assignments and processing inputs and outputs. Outputs process instantaneously if nonidempotent or the command is running in JIT mode. Otherwise, if the output is declared to be idempotent, it is buffered to the end of the atomic region. To properly handle the scoping of *pcS*, the structure of the conditional branching rules are altered to depend on a runtime command $\mathsf{end}_{\mathsf{if}}$.

Additionally, expression configurations are annotated with a qualifier purely for proof purposes. When an expression evaluation is triggered from the command level, this qualifier is initialized to $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ – primitive values used directly in a program (i.e., are not stored in a location) are assumed to have this qualifier. This will only be replaced when a variable is read within the expression, which will replace the qualifier with that of the read location. For an expression, this approach will default to the qualifier $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ for straight up primitive values, or the join of

qualifiers of variables appearing in the expression.

$\boxed{K_o \to K_o}$

D-IF

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \\ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ \mathsf{if}\, e'@q'\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2\end{array}}$$

D-IF-STEP

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@q \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{if}\, e@q\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \\ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ \mathsf{if}\, e'@q'\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2\end{array}}$$

D-IF-TT

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{tt}@q_v) \qquad pc' = pc \sqcup_q q_v \qquad pcS = pc \rhd pcS_0}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{if}\, \mathsf{tt}@q_v\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \\ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pc' \rhd pcS \mid\ c_1; \mathsf{end}_{\mathsf{if}}\end{array}}$$

D-IF-FF

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \mathsf{ff}@q_v) \qquad pc' = pc \sqcup_q q_v \qquad pcS = pc \rhd pcS_0}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{if}\, \mathsf{ff}@q_v\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 \\ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pc' \rhd pcS \mid\ c_2; \mathsf{end}_{\mathsf{if}}\end{array}}$$

D-IF-END

$$\frac{n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pc' \rhd pcS \mid\ \mathsf{end}_{\mathsf{if}} \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}}$$

D-SEQ

$$\frac{}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ c_1; c_2 \to \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ c_1;_{\gamma|\mathsf{V}} c_2}$$

D-SEQ-STEP

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ c_1 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS \mid\ c_1'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ c_1;_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS \mid\ c_1';_W c_2}$$

D-SEQ-V

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction \mathsf{dom}(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip};_W c_2 \to \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}'' \mid pcS \mid\ c_2}$$

D-Let

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{let}\, x = e\, \mathsf{in}\, c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ \mathsf{let}\, x = e'@q'\, \mathsf{in}\, c}$$

D-Let-step

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid e@q \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{let}\, x = e@q\, \mathsf{in}\, c \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ \mathsf{let}\, x = e'@q'\, \mathsf{in}\, c}$$

D-Let-v

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@q) \\ \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\ \mathit{fresh} \qquad n = n' + 1 \qquad pcS = pc \rhd pcS' \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{let}\, x = e\, \mathsf{in}\, c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}, \ell@q \sqcup pc \hookrightarrow e \mid pcS \mid\ c}$$

D-Assign

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid e'@q'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ x := e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ x := e'@q'}$$

D-Assign-step

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid e@q \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid e'@q'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ x := e@q \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ x := e'@q'}$$

D-Assign-V

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@q) \\ \gamma = \gamma', [x \rightarrow \ell] \qquad \mathsf{V} = \mathsf{V}', \ell@\langle qID, qIO\rangle \hookrightarrow v' \qquad q = \langle qID_e, qIO_e\rangle \\ pcS = pc \rhd pcS' \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad q' = \langle qID, qIO_e \sqcup_t qIO_{pc}\rangle \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ x := e@q \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid pcS \mid\ \mathsf{skip}}$$

D-Assign-N

$$\frac{\begin{array}{c} \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@q) \qquad q = \langle qID_e, qIO_e\rangle \qquad \gamma = \gamma', [x \rightarrow \ell] \\ \mathsf{N} = \mathsf{N}', \ell@\langle qID, qIO\rangle \hookrightarrow v' \qquad pcS = pc \rhd pcS' \qquad qID' = \delta(pc, qID, Wt) \neq \mathit{UN} \\ pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad q' = \langle qID', qIO_e \sqcup_t qIO_{pc}\rangle \qquad n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ x := e@q \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}}$$

D-Input

$$\frac{\begin{array}{c} \gamma' = \gamma, [x \mapsto \ell] \\ \ell\ \mathit{fresh} \qquad n = n' + 1 \qquad \mathsf{I} = \mathsf{in}(v)::\mathsf{I}' \qquad pcS = pc \rhd pcS' \qquad q = \langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{let}\, x = \mathsf{IN}()\, \mathsf{in}\, c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}, (\ell@q \hookrightarrow v) \mid pcS \mid\ c}$$

D-Output

$$\frac{\begin{array}{c} \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q' \\ n = n' + 1 \end{array}}{\begin{array}{c} \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e) \\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e'@q') \end{array}}$$

D-Output-step

$$\frac{\gamma \mid \texttt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@q \rightarrow \gamma \mid \texttt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q' \qquad n = n' + 1}{\begin{array}{c}\gamma \mid \texttt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e@q) \\ \rightarrow \gamma \mid \texttt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e'@q')\end{array}}$$

D-Output-Nid-aid

$$\frac{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ v@q) \qquad ID \notin \mathcal{O} \qquad n = n' + 1}{\gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, v@q) \xrightarrow{o(ID,v)} \gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}}$$

D-Buffer-Id-aid

$$\frac{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ v@q) \qquad ID \in \mathcal{O} \qquad n = n' + 1 \qquad \mathsf{O} = \mathsf{O}'::\mathsf{out}(ID, v)}{\gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, v@q) \rightarrow \gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n' \mid \mathsf{I} \mid \mathsf{O}' \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}}$$

D-Output-jit

$$\frac{Val(\gamma \mid \texttt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ v@q) \qquad n = n' + 1}{\gamma \mid \mathsf{jit} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, v@q) \xrightarrow{o(ID,v)} \gamma \mid \mathsf{jit} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}}$$

D.2.5 Expression Execution

We present the expression execution rules. Since they are defined with a small-step operational semantics, extra work is needed to evaluate the expression's qualifier. To do this, each expression is initially annotated with $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ (as in D-V-Read, D-N-Read, and D-Binary-0). If the expression is a variable, then the qualifier is replaced with the variable's qualifier from nonvolatile and volatile memory, evolving the idempotence qualifier on read in the former case. Otherwise, if it is a binary expression, $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ is split into a set of two qualifiers corresponding to e_1 and e_2, and the rules D-Binary-1 and D-Binary-2 solve for each qualifier one at a time. Lastly, they are joined together in D-Binary-v. This approach also enforces that reading constants will always return a qualifier $\langle \mathsf{Id}, \mathtt{Nt} \rangle$

$\boxed{K_o \to K_o}$

D-V-Read

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell @ q \hookrightarrow v, \mathsf{V}' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ x @ \langle \mathsf{Id}, \mathtt{Nt} \rangle \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \ pcS \mid v @ q}$$

D-N-Read

$$\frac{\begin{array}{c}\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{N} = \ell @ \langle qID, qIO \rangle \hookrightarrow v, \mathsf{N}' \qquad qID' = \delta(pc, qID, \mathtt{RD}) = \mathit{UN} \\ q' = \langle qID', qIO \rangle \qquad pcS = pc \rhd pcS' \qquad \mathsf{N}'' = \ell @ q' \hookrightarrow v, \mathsf{N}' \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ x @ \langle \mathsf{Id}, \mathtt{Nt} \rangle \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}'' \mid \mathsf{V} \mid \ pcS \mid v @ q'}$$

D-Binary-0

$$\frac{n > 0}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_1 \odot e_2 @ \langle \mathsf{Id}, \mathtt{Nt} \rangle \\ \to \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_1 \odot e_2 @ \{\langle \mathsf{Id}, \mathtt{Nt} \rangle, \langle \mathsf{Id}, \mathtt{Nt} \rangle\}\end{array}}$$

D-Binary-1

$$\frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_1 @ \{q_1\} \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid \ e_1' @ \{q_1'\}}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_1 \odot e_2 @ \{q_1, \langle \mathsf{Id}, \mathtt{Nt} \rangle\} \\ \to \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid \ e_1' \odot e_2 @ \{q_1', \langle \mathsf{Id}, \mathtt{Nt} \rangle\}\end{array}}$$

D-Binary-2

$$\frac{\begin{array}{c}n > 0 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid e_1 @ q_1) \\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_2 @ \{q_2\} \ \to \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid \ e_2' @ \{q_2'\}\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_1 \odot e_2 @ \{q_1, q_2\} \ \to \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid \ e_1 \odot e_2' @ \{q_1, q_2'\}}$$

D-Binary-v

$$\frac{\begin{array}{c}n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_1 @ q_1) \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid \ e_2 @ q_2) \qquad v = e_1 \odot e_2 \qquad q' = q_1 \sqcup_q q_2\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \ pcS \mid e_1 \odot e_2 @ \{q_1, q_2\} \ \to \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \ pcS \mid v @ q'}$$

Definition 26 (Well-formedness for configurations) *A configuration* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid c$ *is well-formed iff when* $c = c_1;_{W_1} \cdots ;_{W_{n-1}} c_n;_{W_n} c_{n+1}$ *where* $n > 1$*, all of the following hold*

- $dom(\mathsf{V}_j) \subseteq dom(\mathsf{V}_i) \subseteq dom(\mathsf{V})$
- $\gamma_j \subseteq \gamma_i \subseteq \gamma$

where $1 \leq i < j \leq n$ *and* $W_k = \gamma_k \mid \mathsf{V}_m$ *for all* $m \in [1, n]$.

D.3 Type system

This section presents the type system for our calculus. Variables in nonvolatile and volatile memory are typed under Ω and Σ, each mapping variables in their respective memories to a base type and qualifier according to the undo-logging interpretation for crash types. Note that unlike previous systems, volatile memory locations are annotated with a q which is meaningful, reflecting the (non-)idempotence and taint level of a location.

store types	A	::=	$\mathtt{int} \mid \mathtt{bool}$
basic types	T	::=	$\mathtt{unit} \mid A$
unstable types	τ^u	::=	$T \mid \downarrow \tau^s \mid \tau^u \vee \tau^u \mid \mathtt{nat} \rightsquigarrow \tau^u \mid \mathsf{C}_T^{\mathtt{Md}}$
stable types	τ^s	::=	$\uparrow \tau^u$
crash type variables	$\mathsf{C}_T^{\mathtt{Md}}$	::=	$\mathsf{C}_{\mathtt{unit}} \mid \mathsf{C}_A^{\mathtt{Md}}$
unstable store typing context	Σ	::=	$\cdot \mid x : \downarrow\uparrow A@q, \Sigma \mid x_{\mathtt{ck}} : \downarrow\uparrow A@q, \Omega$
stable store typing context	Ω	::=	$\cdot \mid x : \uparrow A@q, \Omega$

Typing judgments. Command and expression typing judgments are extended with post-contexts for taint tracking. Expression typing judgments are additionally extended with a qualifier q to enable proper taint tracking throughout the type checking.

command	(J_u)	$\mathtt{Md} \mid b\,\mathcal{R}\,0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma'$	c could crash
	(J_u)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega'; \Sigma'$	c will not crash
	(J_s)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega'$	after commit
expression	(J_u)	$\mathtt{Md} \mid b\,\mathcal{R}\,0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@q \dashv \Omega'$	e read, could crash
	(J_u)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} v : \downarrow\uparrow A@q \dashv \Omega'$	e read no crash
	(J_s)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{RD}} v : \uparrow A@q \dashv \Omega'$	e read, commit
	(J_u)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@q \dashv \Omega'$	write on x, no crash
	(J_s)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{WT}} x : \uparrow A@q \dashv \Omega'$	write on x, commit
program	(J_s)	$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \vdash p : \uparrow\mathsf{C}_{\mathtt{unit}}$	before execution
crash	(J_u)	$\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \kappa : \mathsf{C}_T^{\mathtt{Md}}$	about to crash
	(J_u)	$\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \downarrow\uparrow\varepsilon \,\#\, \mathsf{in}(b > 0, \kappa@pcS) : \downarrow\uparrow(\mathtt{nat} \rightsquigarrow \mathsf{C}_T^{\mathtt{Md}})$	crash state
	(J_s)	$\mathtt{Md} \mid \cdot \mid \Omega \vdash_{\mathtt{Sig}} \uparrow\varepsilon \,\#\, \mathsf{in}(b > 0, \kappa@pcS) : \uparrow(\mathtt{nat} \rightsquigarrow \mathsf{C}_T^{\mathtt{Md}})$	restore state
	(J_u)	$\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \kappa@pcS) : \mathtt{nat} \rightsquigarrow \mathsf{C}_T^{\mathtt{Md}}$	waiting for energy
	(J_u)	$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \kappa : \mathsf{C}_T^{\mathtt{Md}}$	before re-execution

D.3.1 Expression typing rules

We present the expression typing rules. These rules are similar to before, except that since access qualifiers can evolve on read, they are extended with a post-context. Expression types are annotated with a qualifier that is looked up on read for variables (T-Loc-Read), $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ for constants (T-Bool-t, T-Bool-f, T-Int), or joined together for binary expressions(T-Binary).

T-Loc-Write

$$\frac{\Omega = x : \uparrow A@q, \Omega_2' \qquad q = \langle qID_x, qIO_x \rangle \qquad pcS = pc \rhd pcS' \qquad qID' = \delta(pc, qID_x, Wt) \neq UN \qquad q' = \langle qID', qIO_x \rangle}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{WT}} x : \uparrow A@q' \dashv x : \uparrow A@q', \Omega_2'}$$

T-W-Shift

$$\frac{\Sigma = \downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \mid pcS \vdash_{\mathtt{WT}} x : \uparrow A@q \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{WT}} x : \downarrow \uparrow A@q \dashv \Omega_1 \lceil \mathsf{dom}(\Omega'), \Omega''_{\mathsf{CK}}}$$

(T-Loc-Read)

$$\frac{\Omega = \Omega', x : \uparrow A@\langle qID, qIO \rangle \qquad pcS = pc \rhd pcS' \qquad qID' = \delta(pc, qID, Rd) \neq UN \qquad q' = \langle qID', qIO \rangle}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{RD}} x : \uparrow A@q' \dashv \Omega', x : \uparrow A@q'}$$

(T-Bool-t)

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{RD}} \mathsf{tt} :\uparrow \mathtt{bool}@\langle \mathsf{Id}, \mathtt{Nt} \rangle \dashv \Omega}$$

(T-Bool-f)

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{RD}} \mathsf{ff} : \uparrow \mathtt{bool}@\langle \mathsf{Id}, \mathtt{Nt} \rangle \dashv \Omega}$$

(T-Int)

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{RD}} \mathsf{n} : \uparrow \mathtt{int}@\langle \mathsf{Id}, \mathtt{Nt} \rangle \dashv \Omega}$$

(T-R-Shift)

$$\frac{\Sigma = \downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \mid pcS \vdash_{\mathtt{RD}} v : \uparrow A@q \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} v : \downarrow \uparrow A@q \dashv \Omega_1 \lceil \mathsf{dom}(\Omega'), \Omega''_{\mathsf{ck}}}$$

(T-∨-Succ)

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} x : \downarrow \uparrow A@q \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} x : (\tau_1 \vee \downarrow \uparrow A)@q \dashv \Omega'}$$

(T-Binary)

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1} @ q_1 \dashv \Omega_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2} @ q_2 \dashv \Omega_2 \\ \odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T' \qquad \Omega' = \Omega_1 \sqcup \Omega_2\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_{T'} @ q_1 \sqcup_q q_2 \dashv \Omega'}$$

(T-enough?)

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}''} e : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \tau @ q \dashv \Omega' \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD}} e : \tau\} \qquad \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \tau @ q \dashv \Omega'}$$

D.3.2 Command typing rules

The command typing rules are shown below. All interesting rules have been explained in the main text. Outputs for JIT regions behave similarly to nonidempotent outputs since both are posted immediately.

T-Skip

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow \mathtt{unit} \dashv \Omega}$$

T-C-Shift

$$\frac{\Sigma = \downarrow \Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow \mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow \uparrow \mathtt{unit} \dashv \Omega_1 \restriction \mathsf{dom}(\Omega'), \Omega''_{\mathsf{CK}}; \downarrow (\Omega_1 \restriction \mathsf{dom}(\Sigma))}$$

T-∨-Succ

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow \uparrow \mathtt{unit} \dashv \Omega'; \Sigma'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \vee \downarrow \uparrow \mathtt{unit} \dashv \Omega'; \Sigma'}$$

T-Let

$$\frac{\begin{array}{c}pcS = pc \rhd pcS' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{A} @ \langle qID, qIO \rangle \dashv \Omega_0 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma, x{:}\downarrow \uparrow A @ \langle \overline{qID}, qIO \rangle \sqcup pc \mid pcS \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{let}\, x = e_1\, \mathsf{in}\, c : \tau \dashv \Omega'; \Sigma'}$$

T-If

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} @ \langle qID_e, qIO_e \rangle \dashv \Omega_0 \\ pcS = pc \rhd pcS' \qquad pc = \langle qID, qIO \rangle \\ pc' = \langle qID \sqcup_{id} qID_e, qIO \sqcup_t qIO_e \rangle \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pc' \rhd pcS \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega_1; \Sigma_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pc' \rhd pcS \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega_2; \Sigma_2 \\ \Omega' = \mathsf{lift}(pc, \Omega_1 \sqcup \Omega_2) \qquad \Sigma' = \Sigma_1 \sqcup \Sigma_2\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{if}\, e\, \mathsf{then}\, c_1\, \mathsf{else}\, c_2 : \tau \dashv \Omega'; \Sigma'}$$

T-Assign-s

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega^1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \qquad pcS = pc \rhd pcS' \\ pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \\ x \in \mathsf{dom}(\Omega) \qquad \Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0 \qquad \Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega''; \Sigma}$$

T-Assign-u

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega_0 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \qquad pcS = pc \rhd pcS' \\ pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \\ x \in \mathsf{dom}(\Sigma) \qquad \Sigma = x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle, \Sigma_1 \qquad \Sigma' = x : \downarrow\uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Sigma_1\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'; \Sigma'}$$

T-seq

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'; \Sigma' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \mid pcS \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''; \Sigma''}$$

T-seq-d

$$\frac{\begin{array}{c}W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}}^{\mathtt{Md}} \dashv \Omega'; \Sigma' \qquad \Omega' = \Omega^1_{\mathtt{CK}}, \Omega^2 \\ \Sigma'' = \mathtt{trim}(\Sigma', \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{ck}} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \mid pcS \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma'''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c_1 ;_W c_2 : \tau \dashv \Omega''; \Sigma'''}$$

T-enough?

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}''} c : \tau \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma' \\ \mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash c : \tau\} \qquad \mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}',\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'}$$

T-output-jit

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : \downarrow\uparrow A@\langle qID_v, qIO_v\rangle \dashv \Omega'}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{output}(ID, e) : \tau \dashv \Omega'; \Sigma}$$

T-output-nid

$$\frac{\begin{array}{c}\mathsf{aID}(c_0, \mathcal{O}) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : A@\langle qID_v, qIO_v\rangle \dashv \Omega' \\ ID \notin \mathcal{O} \qquad \overline{qID_v} \sqsubseteq_{id} \mathsf{Nid}\end{array}}{\mathsf{aID}(c_0, \mathcal{O}) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{output}(ID, e) : \tau \dashv \Omega'; \Sigma}$$

T-output-id

$$\frac{\begin{array}{c}\mathsf{aID}(c_0, \mathcal{O}) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : A@\langle qID_v, qIO_v\rangle \dashv \Omega' \\ ID \in \mathcal{O} \qquad \overline{qID_v} \sqsubseteq_{id} \mathsf{Id}\end{array}}{\mathsf{aID}(c_0, \mathcal{O}) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{output}(ID, e) : \tau \dashv \Omega'; \Sigma}$$

T-Let-in

$$\frac{pcS = pc \rhd pcS' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}\downarrow\uparrow A@\langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc \mid pcS \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{Sig}} \mathsf{let}\, x = \mathsf{IN}(\,)\, \mathsf{in}\, c : \tau \dashv \Omega'; \Sigma'}$$

D.3.3 Statement Typing

The statement typing rules are similar to before, but additionally stash a pc stack in the crash instruction. For JIT mode, we stash the current pc stack to help resume execution. For atomic mode, we stash the default pc stack $\langle \mathsf{Id}, \mathtt{Nt} \rangle$ to help the program re-execute from the beginning.

$$\frac{\Sigma = \Sigma^1, \Sigma^2_{\mathsf{ck}} \qquad \mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2 \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt} \rangle) : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'})}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \kappa : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-aID-}\vee\text{-Crash)}$$

$$\frac{\varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \downarrow\uparrow \kappa @ \langle \mathsf{Id}, \mathtt{Nt} \rangle : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}}{\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow \kappa @ \langle \mathsf{Id}, \mathtt{Nt} \rangle) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}} \text{ (T-Charge)}$$

$$\frac{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \vdash_{\mathtt{Sig}} \kappa : \mathsf{C}^{\mathsf{aID}}_{T} \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \downarrow\uparrow \kappa @ \langle \mathsf{Id}, \mathtt{Nt} \rangle : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T}} \text{ (T-aID-Restore)}$$

$$\frac{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa' @ pcS_t) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{T'})}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \kappa' : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{T'}) \vee \downarrow\uparrow T} \text{ (T-Jit-}\vee\text{-Crash)}$$

$$\frac{\Sigma = \downarrow\Sigma' \qquad \mathsf{jit} \mid \cdot \mid \Omega, \Sigma'_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa' @ pcS_t) : (\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \downarrow \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa' @ pcS_t) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathtt{Md}}_{T})} \text{ (T-Jit-stop)}$$

$$\frac{\varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0 \qquad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \uparrow \kappa @ pcS_t : \mathsf{C}^{\mathtt{Md}}_{T} \in \mathtt{Sig}}{\mathtt{Md} \mid \cdot \mid \Omega; \Sigma \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow \kappa @ pcS_t) : \mathtt{nat} \rightsquigarrow \mathsf{C}^{\mathtt{Md}}_{T}} \text{ (T-Jit-Charge)}$$

$$\frac{\Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathsf{jit} \mid \cdot \mid \Omega'; \downarrow\Omega'' \mid pcS_t \vdash \kappa' : \mathsf{C}^{\mathtt{Md}}_{T} \in \mathtt{Sig}}{\mathsf{jit} \mid \cdot \mid \Omega \mid \cdot \vdash_{\mathtt{Sig}} \uparrow \kappa' @ pcS_t :\uparrow \mathsf{C}^{\mathtt{Md}}_{T}} \text{ (T-Jit-Restore)}$$

D.3.4 Program Typing

The rules for checking programs are defined below. Both rules initialize the *pc* to $\langle \mathsf{Id}, \mathtt{Nt} \rangle$, but otherwise the rule for T-P-seq for checking JIT regions is similar to before. Notably, the rule T-P-Ckpt changes to check for RIO bugs. The way to still detect RIO bugs while avoiding early failing is to check that at the end of the atomic region, there are no variables that are written (but not on all paths) in a tainted context. That is, the rule must check that there are no remaining `mayTntWts` at the end of checking the atomic region. The $\mathsf{InitWorld}_t$ function now only takes a set of checkpointed variables ω and a set of non-idempotent variables *nIds* and sets these variables' qualifiers to $\mathsf{Id}(\mathtt{CK})$ and Nid, respectively. All other variable qualifiers are initialized to $\mathsf{Id}(\mathtt{NRD})$ ("idempotent because has not been read yet").

T-P-CKPT

$$\frac{\begin{array}{c}\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0 \\ \mathtt{Sig} = \{\mathsf{aID}(c_0, \mathit{O}) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma'\} \\ \mathsf{aID}(c_0, \mathit{O}) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma' \\ \Omega' \restriction \{\mathtt{mayTntWt}\} = \emptyset \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}\end{array}}{b : \mathtt{nat} \mid \Omega \vdash \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol, \mathit{O}); c_0; \mathsf{end}_{\mathsf{atom}}; p : \uparrow \mathsf{C}_{\mathtt{unit}}}$$

T-P-SEQ

$$\frac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \cdot \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash_{\emptyset} c : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma' \qquad b : \mathtt{nat} \mid \Omega \vdash p : \uparrow \mathsf{C}_{\mathtt{unit}}}{b : \mathtt{nat} \mid \Omega \vdash c; p : \uparrow \mathsf{C}_{\mathtt{unit}}}$$

Definition 27 $\mathsf{InitWorld}_t(\Omega; aPol) = \Omega_0 \mid \Sigma_0$ *iff*

- $\mathsf{dom}(\Omega) \supseteq \omega \cup nIds$,
- $\omega \cap nIds = \emptyset$,
- $\Sigma_0 = \{x_{\mathtt{ck}} : \downarrow\uparrow A @ \langle \mathsf{Id}(\mathtt{CK}), \mathtt{Nt}\rangle \mid x : \uparrow A @ \langle \mathsf{Id}, qIO\rangle \in \Omega^c\}$
- $\Omega_0 = \{x{:}\uparrow A @ \langle \mathsf{Nid}, \mathtt{Nt}\rangle \mid x{:}\uparrow A @ \langle \mathsf{Nid}, qIO\rangle \in \Omega^n\}$
  $\cup \{x{:}\uparrow A @ \langle \mathsf{Id}(\mathtt{NRD}), \mathtt{Nt}\rangle \mid x{:}\uparrow A @ \langle \mathsf{Id}, qIO\rangle \in \Omega^{nrd}\} \cup \Omega^c$

where $aPol = (\omega, nIds)$ *and* $\Omega = \Omega^c, \Omega^n, \Omega^{nrd}$ *and* $\Omega^c = \Omega \restriction \omega$ *and* $\Omega^n = \Omega \restriction nIds$.

D.3.5 Store Typing

The rules for store typing are similar to the ones for undo-logging, but extended with a rule for typing the operational pc stack wrt the type level pc stack. In particular, both pc stacks will be the same.

$$\text{(Empty)}\quad \frac{}{\vdash^{\mathtt{Md}}_{\gamma} \cdot \mid \cdot \mid \cdot : \cdot \mid \cdot \mid \cdot}$$

$$\text{(V-loc)}\quad \frac{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V}' \mid pcS_o : \Omega \mid \Sigma \mid pcS_t \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \mathsf{V}', \ell@q \hookrightarrow v \qquad \mathtt{Md} \mid b : \mathtt{nat} \vdash_{\mathtt{RD}} v : \uparrow A@q}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_0 : \Omega \mid \Sigma, (x{:}\downarrow\uparrow A@q) \mid pcS_t}$$

(NV-loc-aid-1)

$$\frac{\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v \qquad q \neq \langle \mathsf{Id}(\mathtt{CK}), qIO \rangle \qquad \vdash^{\mathsf{aID}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{aID} \mid b : \mathtt{nat} \vdash_{\mathtt{RD}} v : \uparrow A@q}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega, (x : \uparrow A@q) \mid \Sigma \mid pcS_t}$$

(NV-loc-aid-2)

$$\frac{\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v \qquad q = \langle \mathsf{Id}(\mathtt{CK}), qIO \rangle \qquad \vdash^{\mathsf{aID}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{aID} \mid b : \mathtt{nat} \vdash_{\mathtt{RD}} v : \downarrow\uparrow A@q}{\vdash^{\mathsf{aID}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega, (x : \uparrow A@q) \mid \Sigma, (x_{\mathtt{ck}} : \downarrow\uparrow A@q) \mid pcS_t}$$

(NV-loc-jit)

$$\frac{\mathsf{N} = \mathsf{N}', \ell@q \hookrightarrow v \qquad \vdash^{\mathsf{jit}}_{\gamma'} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t \qquad \gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{jit} \mid b : \mathtt{nat} \vdash_{\mathtt{RD}} v : \uparrow A@q}{\vdash^{\mathsf{jit}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega, (x : \uparrow A@q) \mid \Sigma \mid pcS_t}$$

$$\text{(pc)}\quad \frac{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pc \rhd pcS_o : \Omega \mid \Sigma \mid pc \rhd pcS_t}$$

D.4 Progress and Preservation

Lemma 26 (Progress for shifted expressions) *If*

$$\mathtt{Md} \mid b{:}\mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathtt{RD}} e : \uparrow A@q \dashv \Omega'$$

then $\forall\ n : \mathtt{nat}$ *with* $n > 0$ *and* $\forall\, \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid pcS_t$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q_0)$ *where* $q_0 = q$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@q')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q_0 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS'_o \mid e'@q'$.

Proof: The proof proceeds by structural induction over $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathtt{RD}} e : \uparrow A@q \dashv \Omega'$.

Case 1 [T-Loc-Read]**.** Suppose the last rule in the typing derivation is T-Loc-Read:

(T-Loc-Read)

$$\frac{\begin{array}{ccc} & \Omega = \Omega', x : \uparrow A@\langle qID, qIO\rangle & \\ pcS_t = pc_t \rhd pcS'_t & qID' = \delta(pc_t, qID, Rd) \neq UN & q' = \langle qID', qIO\rangle \end{array}}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathtt{RD}} x : \uparrow A@q' \dashv \Omega', x : \uparrow A@q'}$$

Then by inversion, we have $\Omega = \Omega', x : \uparrow A@\langle qID, qIO\rangle$. By assumption, $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid pcS_t$. By inversion of $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid pcS_t$ according to the well-formedness definition, we learn that $pc_o = pc_t$ and $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS'_o : \Omega \mid pcS'_t$, where $pcS_o = pc_o \rhd pcS'_o$. Additionally, one of the two subcases hold:

Subcase 1. $\mathsf{N} = \ell@\langle qID, qIO\rangle \hookrightarrow v, \mathsf{N}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-N-Read applies:

D-N-Read

$$\frac{\begin{array}{ccc} \gamma = \gamma', [x \mapsto \ell] & \mathsf{N} = \ell@\langle qID, qIO\rangle \hookrightarrow v, \mathsf{N}' & \\ pcS_o = pc_o \rhd pcS'_o & qID' = \delta(pc_o, qID, Rd) \neq UN & \\ q' = \langle qID', qIO\rangle & \mathsf{N}'' = \ell@q' \hookrightarrow v, \mathsf{N}' & n = n' + 1 \end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ x@\langle qID, qIO\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}'' \mid \mathsf{V} \mid\ pcS_o \mid v@q'}$$

This yields the desired result.

Subcase 2. $\mathsf{V} = \ell@\langle qID, qIO\rangle \hookrightarrow v, \mathsf{V}'$ with $\gamma = \gamma', [x \mapsto \ell]$.

Since $n > 0$, it follows that $\exists n'$ such that $n = n' + 1$, and hence the evaluation rule D-V-Read applies:

D-V-Read

$$\frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ x@q \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid\ pcS_o \mid v@q}$$

This yields exactly the desired result.

Case 2 [T-Bool-t]**.** Suppose that the last rule in the typing derivation is T-Bool-t:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathrm{RD}} \mathsf{tt} :\uparrow \mathtt{bool}@\langle \mathsf{Id}, \mathtt{Nt}\rangle \dashv \Omega} \text{(T-Bool-t)}$$

By the value rule V-Bool-t and the assumption $n > 0$ we get

$$\frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{tt}@\langle \mathsf{Id}, \mathtt{Nt}\rangle)} \text{V-Bool-t}$$

Case 3 [T-Bool-f]**.** Suppose that the last rule in the typing derivation is T-Bool-f:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathrm{RD}} \mathsf{ff} :\uparrow \mathtt{bool}@\langle \mathsf{Id}, \mathtt{Nt}\rangle \dashv \Omega} \text{(T-Bool-f)}$$

By the value rule V-Bool-f and the assumption $n > 0$ we get

$$\frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{ff}@\langle \mathsf{Id}, \mathtt{Nt}\rangle)} \text{V-Bool-f}$$

Case 4 [T-Int]**.**

Suppose that the last rule in the typing derivation is T-Nat:

$$\frac{}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega \mid pcS_t \vdash_{\mathrm{RD}} \mathsf{n} : \uparrow\mathtt{int}@\langle \mathsf{Id}, \mathtt{Nt}\rangle \dashv \Omega} \text{(T-Nat)}$$

By the value rule V-Nat and the assumption $n > 0$ we get:

$$\frac{n > 0}{Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{n}@\langle \mathsf{Id}, \mathtt{Nt}\rangle)} \text{V-Nat}$$

□

Theorem 16 (Progress for expressions) *If* $\mathtt{Md} \mid b \mathcal{R} m : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathrm{RD;Sig}} e : \tau@q \dashv \Omega'$*, then* $\forall\ n$ *: nat with* $n\mathcal{R}m$ *and* $\forall\ \mathsf{N}, \mathsf{V}, \gamma$ *with* $\vdash_{\gamma}^{\mathsf{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q_0)$ *where* $q_0 = q$ *or*
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@\{q'\})$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\{q_0\} \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@\{q'\}$.

Proof: The proof is by structural induction over $\mathtt{Md} \mid b \ \mathcal{R} \ m : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \tau@q \dashv \Omega'$. We consider a specific (co-)natural number $n\mathcal{R}m$ and contexts $\mathsf{N}, \mathsf{V}, \gamma$ with $\vdash^{\mathsf{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$. We consider cases based on the last step in the derivation:

Case 1 [T-Enough?]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig''}} e : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \tau@q \dashv \Omega' \\ \mathtt{Sig'} = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD}} e : \tau\} \\ \mathtt{Sig''} = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig'},\ \mathit{else}\ \mathtt{Sig}\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \tau@q \dashv \Omega'}\ (\text{T-ENOUGH?})$$

By assumption, we know that $n \geq 0$. We consider two subcases based on the value of n (i.e. $n = 0$ or $n > 0$):

Subcase 1 [$n = 0$]**.** By the value rule V-E-Crash, we have

$$\mathit{Val}(\gamma \mid \mathtt{Md} \mid 0 \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e),$$

which completes the proof of this subcase.

Subcase 2 [$n > 0$]**.**

By inversion on T-Enough?, we have

(1) $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig''}} e : \tau$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \tau@q \dashv \Omega'$

(3) $\mathtt{Sig'} = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD}} e : \tau\}$

(4) $\mathtt{Sig''} = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig'},\ \mathit{else}\ \mathtt{Sig}$

We can apply the induction hypothesis to (2) to get for $n > 0$, and $\mathsf{N}, \mathsf{V}, \gamma$ either

- $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$ or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@\{q'\})$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\{q^0\} \ \rightarrow \ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@\{q'\}$.

which completes the proof of this subcase.

Case 2 [T-Binary]**.**

Suppose the last rule in the typing derivation is T-Binary and $e = e_1 \odot e_2$:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1}@q_1 \dashv \Omega_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2}@q_2 \dashv \Omega_2 \\ \odot : \uparrow T_1 \times \uparrow T_2 \rightarrow \uparrow T' \qquad \Omega' = \Omega_1 \sqcup \Omega_2\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_{T'}@q_1 \sqcup_q q_2 \dashv \Omega'}$$

By assumption, we know that $n > 0$. By inversion, we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1}@q_1 \dashv \Omega_1$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2} @ q_2 \dashv \Omega_2$

(3) $\odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T'$

By the inductive hypothesis applied to (1), either

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 @ q_1)$, or
- $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1' @ \{q_1'\})$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 @ \{q_1^0\} \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1' @ \{q_1'\}$.

From here, the proof proceeds in two subcases:

Subcase 1. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 @ q_1)$

We apply the inductive hypothesis to (2) to get two sub-subcases.

Subcase 1a. $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_2 @ q_2)$.

Since $n > 0$, we can apply the dynamic rule D-Binary-v to get

$$\begin{aligned} &\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2 @ \{q_1, q_2\} \\ &\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid v @ q_1 \sqcup_q q_2, \end{aligned}$$

where $v = e_1 \odot e_2$, and $n = n' + 1$.

Subcase 1b. $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_2' @ \{q_2'\})$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid e_2 @ \{q_2^0\} \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_2' @ \{q_2'\}$. By D-Binary-2,

$$\begin{aligned} &\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2 @ \{q_1, q_2^0\} \\ &\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2' @ \{q_1, q_2'\} \end{aligned}$$

Subcase 2. Suppose that $\exists(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1' @ q_1')$ such that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 @ q_1^0 \;\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1' @ q_1'$. Then, by D-Binary-1,

$$\begin{aligned} &\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2 @ \{q_1^0, q_2\} \\ &\to\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1' \odot e_2 @ \{q_1', q_2\} \end{aligned}$$

The desired result holds in all subcases.

Case 3 [T-∨-succ]**.**

Suppose the last rule in the typing derivation is T-∨-succ:

$$\text{(T-}\vee\text{-Succ)}\quad \frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A @ q \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : (\tau_1 \vee \downarrow\uparrow A) @ q \dashv \Omega'}$$

By assumption, we get $n > 0$. By inversion, we have:

$$\dagger\ \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A@q \dashv \Omega'$$

We know that the last rule for deriving † is T-R-Shift:

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{ck}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \mid pcS_t \vdash_{\mathtt{RD}} v : \uparrow A@q \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \downarrow\uparrow A@q \dashv \Omega_1 \restriction \mathsf{dom}(\Omega'), \Omega''_{\mathsf{ck}}}\ \text{(T-R-Shift)}$$

By inversion, we have:

(1) $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \mid pcS_t \vdash_{\mathtt{RD}} v : \uparrow A@q \dashv \Omega_1$

(2) $\Sigma = \downarrow\Sigma'$

(3) $\Omega = \Omega', \Omega''_{\mathsf{ck}}$

By assumption $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$ and (2), it follows from Lemma 27 that $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega', \Sigma' \mid pcS_t$. Then the desired result follows directly by application of Lemma 26 to (1) (since $n > 0$).

□

Lemma 27 *If* $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$ *and* $\Sigma = \downarrow\Sigma'$*, and* $\Omega = \Omega', \Omega''_{\mathsf{ck}}$*, then* $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega', \Sigma' \mid pcS_t$*.*

Proof: The proof is by induction on the structure of $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$. For each step in the derivation, we build the corresponding step of a derivation for $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega, \Sigma' \mid pcS_t$ according to the well-formedness definition. □

Theorem 6.1 (Progress for Commands) *If* $\mathtt{Md} \mid b\ \mathcal{R}\ m : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'$*, then* $\forall\ n : \mathtt{nat}$ *with* $n\mathcal{R}m$ *and* $\forall\ \gamma, \mathsf{N}, \mathsf{V}$ *with* $\vdash^{\mathsf{Md}}_{\gamma}\ \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$*, either*

- $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c)$ *or*
- $\exists(\gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c')$ *such that* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c \rightarrow \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'$.

Axiom 3 (positive input to generation channel) $\varepsilon\ \#\ \mathsf{i\eta}() : \mathtt{nat} > 0$.

Lemma 28 (well-typedness of expressions under crash in jit**)** $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}'} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD}} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}\}$.

Proof: Note that $\mathsf{C}^{\mathsf{jit}}_{\mathtt{A}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}}) \vee \downarrow\uparrow\mathtt{A}$. Let $\Omega' = \Omega, \Omega''_{\mathsf{ck}}$ where $\Sigma = \downarrow\Omega''$. By axiom 3, we have $\varepsilon\ \#\ \mathsf{i\eta}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumption that $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD}} e : \mathsf{C}^{\mathsf{jit}}_{\mathtt{A}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash_{\mathtt{RD}} e : \mathsf{C}_{\mathsf{A}}^{\mathsf{jit}} \in \mathtt{Sig}'}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{RD;Sig}'} \uparrow e : \uparrow\mathsf{C}_{\mathsf{A}}^{\mathsf{jit}}}\ \Omega' = \Omega, \Omega''_{\mathsf{ck}}}{\mathsf{jit} \mid \cdot \mid \Omega, \Omega''_{\mathsf{ck}} \vdash_{\mathtt{RD;Sig}'} \varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow e) : (\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathsf{A}}^{\mathsf{jit}})}\ \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} \downarrow\varepsilon \,\#\, \mathsf{in}(b > 0, \uparrow e) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{\mathsf{A}}^{\mathsf{jit}})}\ \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{A}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}}$$

(T-Jit-Restore), (T-Charge), (T-Jit-stop), (T-∨-Crash)

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{RD;Sig}'} e : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}_{A}^{\mathsf{jit}}) \vee \downarrow\uparrow\mathtt{A}$ yields the desired result.

□

Lemma 29 (well-typedness of expressions under crash in aID) *If* $\mathsf{aID}(c_0, \vec{O}) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \mid pcS_t \vdash_{\mathtt{RD;Sig}} e' : \tau$ *s.t.* $\mathsf{dom}(\Sigma') \supseteq \mathsf{dom}(\Omega^1_{\mathsf{CK}})$ *where* $\Omega = \Omega^1_{\mathsf{CK}}, \Omega^2$, *then* $\mathsf{aID}(c_0, \vec{O}) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS'_t \vdash_{\mathtt{RD;Sig}} e : \tau$ *s.t.* $\mathsf{dom}(\Sigma) \supseteq \mathsf{dom}(\Omega^1_{\mathsf{CK}})$.

Proof: Let the type $\tau = \mathsf{C}_A^{\mathsf{aID}}$ and note that $\mathsf{C}_A^{\mathsf{aID}} = (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{A}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_T^{\mathsf{aID}} \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt}\rangle : \downarrow\uparrow\mathsf{C}_T^{\mathsf{aID}}}\ \Sigma' = \Sigma^1, \Sigma^2_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt}\rangle) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_{T'}}\ \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \mid pcS_t \vdash_{\mathtt{Sig}} e' : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}$$

(T-aID-Restore), (T-aID-Charge), (T-aID-∨-Crash)

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_T^{\mathsf{aID}} \in \mathtt{Sig}$

(2) $\Sigma' = \Sigma^1, \Sigma^2_{\mathsf{ck}}$

(3) $\varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation where $\Sigma \supseteq \Sigma^2_{\mathsf{ck}}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}_T^{\mathsf{aID}} \in \mathtt{Sig}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt}\rangle : \downarrow\uparrow\mathsf{C}_T^{\mathsf{aID}}}\ \Sigma' = \Sigma^1, \Sigma^2_{\mathsf{ck}}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt}\rangle) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_{T'}}\ \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS'_t \vdash_{\mathtt{Sig}} e : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}$$

(T-aID-Restore), (T-aID-Charge), (T-aID-∨-Crash)

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS'_t \vdash_{\mathtt{Sig}} e : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{A}$ yields the desired result.

□

Lemma 30 (well-typedness of commands under crash in `jit`) $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}'} c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}$ *for* $\mathtt{Sig}' = \{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}\}$.

Proof: Note that $\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} = \downarrow(\mathtt{nat} \rightsquigarrow \uparrow \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. Let $\Omega' = \Omega, \Omega''_{\mathsf{ck}}$ where $\downarrow\Omega'' = \Sigma$. By axiom 3, we have that $\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0$. Then the typing derivation follows by the assumptions $\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}'$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{jit} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \downarrow\Omega'' \vdash c : \mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}} \in \mathtt{Sig}' \quad \Omega' = \Omega, \Omega''_{\mathsf{ck}}}{\mathsf{jit} \mid b > 0 : \mathtt{nat} \mid \Omega' \vdash_{\mathtt{Sig}'} \uparrow c : \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}}\ (\text{T-Jit-Restore}) \quad \varepsilon \# \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{jit} \mid \cdot \mid \Omega' \vdash_{\mathtt{Sig}'} \varepsilon \# \mathsf{in}(b > 0, \uparrow c) : (\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ (\text{T-Charge}) \quad \Sigma = \downarrow\Omega''}{\mathsf{jit} \mid \cdot \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} \downarrow\varepsilon \# \mathsf{in}(b > 0, \uparrow c) : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}})}\ (\text{T-Jit-stop})}{\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}}\ (\text{T-}\vee\text{-Crash})$$

The conclusion $\mathsf{jit} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \vdash_{\mathtt{Sig}'} c : \downarrow(\mathtt{nat} \rightsquigarrow \uparrow\mathsf{C}^{\mathsf{jit}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$ yields the desired result. □

Lemma 31 (well-typedness of commands under crash in aID) *If* $\mathsf{aID}(c_0, O) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma' \mid pcS'_t \vdash_{\mathtt{Sig}} c' : \tau$ *s.t.* $\mathsf{dom}(\Sigma') \supseteq \mathsf{dom}(\Omega^1_{\mathsf{CK}})$ *where* $\Omega = \Omega^1_{\mathsf{CK}}, \Omega^2$ *then* $\mathsf{aID}(c_0, O) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau$ *s.t.* $\mathsf{dom}(\Sigma) \supseteq \mathsf{dom}(\Omega^1_{\mathsf{CK}})$.

Proof: Let the type $\tau = \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}$ and note that $\mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}} = (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow\mathtt{unit}$. By inversion on the assumed typing judgment

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma = \Sigma^1, \Sigma^2_{\mathsf{ck}} \quad \varepsilon \# \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}}\ (\text{T-aID-Restore})}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt} \rangle : \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_T}\ (\text{T-aID-Charge})}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \varepsilon \# \mathsf{in}(b > 0, \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt} \rangle) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_{T'}}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS'_t \vdash_{\mathtt{Sig}} c' : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow\mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}\ (\text{T-aID-}\vee\text{-Crash})$$

we learn the following:

(1) $\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \langle \mathsf{Id}, \mathtt{Nt} \rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}$

(2) $\Sigma = \Sigma^1, \Sigma^2_{\mathsf{ck}}$

(3) $\varepsilon \# \mathsf{in}() : \mathtt{nat} > 0$

Therefore, we have the following typing derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Sigma = \Sigma^1, \Sigma^2_{\mathsf{ck}} \quad \varepsilon \,\#\, \mathsf{in}() : \mathtt{nat} > 0}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \langle \mathsf{Id}, \mathtt{Nt}\rangle \vdash_{\mathtt{Sig}} c_0 : \mathsf{C}^{\mathsf{aID}}_T \in \mathtt{Sig}}\ \text{(T-aID-Restore)}}{\mathsf{aID}(c_0) \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid \cdot \vdash_{\mathtt{Sig}} \downarrow\uparrow c_0 : \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_T}\ \text{(T-aID-Charge)}}{\mathsf{aID}(c_0) \mid \cdot \mid \Omega; \Sigma^2_{\mathsf{ck}} \mid pcS_t \vdash_{\mathtt{Sig}} \varepsilon \,\#\, \mathsf{in}(b > 0, \downarrow\uparrow c_0 @ \langle \mathsf{Id}, \mathtt{Nt}\rangle) : \mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}}\ \text{(T-aID-}\vee\text{-Crash)}}{\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{T'}) \vee \downarrow\uparrow T}$$

The conclusion $\mathsf{aID}(c_0) \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : (\mathtt{nat} \rightsquigarrow \downarrow\uparrow \mathsf{C}^{\mathsf{aID}}_{\mathtt{unit}}) \vee \downarrow\uparrow \mathtt{unit}$ yields the desired result.

□

Lemma 32 (Ω shift for in-progress binary expressions, e_2) *If*

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e'_1 : \mathsf{C}^{\mathtt{Md}}_{T_1} @ q'_1 \dashv \Omega_1$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2} @ q_2 \dashv \Omega_2$ *and*

(3) $\odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T'$

(4) $\Omega' = \Omega_1 \sqcup \Omega_2$

(5) $\Omega \preccurlyeq_{Rd} \Omega_0$

then, $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e'_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_T @ q'_1 \sqcup_q q_2 \dashv \Omega'$.

Proof: To prove the desired result, we must first show that for some Ω'_2

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2} @ q_2 \dashv \Omega'_2$$

where $\Omega' = \Omega_1 \sqcup \Omega'_2$.

By definition of $\Omega \preccurlyeq_{Rd} \Omega_0$, we know that $\forall x{:}\tau @ \langle qID, qIO\rangle \in \Omega$, it is the case that $x{:}\tau @ \langle qID', qIO\rangle \in \Omega_0$ and $qID' \neq UN$, where either

(i) $\delta(pc, qID, Rd) = qID'$ $(\exists pc)$ or

(ii) $qID = qID'$.

Let $\Omega = \Omega^1, \Omega^2$ where Ω^1 is the portion of Ω such that (i) applies, and Ω^2 is the portion of Ω such that (ii) applies. For Ω^2, we know that $\Omega_0 \lceil \mathsf{dom}(\Omega^2) = \Omega^2$ follows by definition, and therefore $\Omega'_2 \lceil \mathsf{dom}(\Omega^2) = \Omega_2 \lceil \mathsf{dom}(\Omega^2)$ $(*)$.

It remains to reason about Ω^1. Let $x_0{:}\tau @ \langle qID_0, qIO_0\rangle \in \Omega^1$ be arbitrary. Since (i) applies, we know that $\delta(pc_0, qID_0, Rd) = qID'_0$ $(\exists pc_0)$. There are two subcases to consider:

Subcase 1: x_0 is read by e_2. If x_0 is read by e_2, then it follows from (1) that $\Omega_2(x_0) = \tau @ \langle qID'_0, qIO_0\rangle$ where $\delta(pc_0, qID_0, Rd) = qID'_0$ $(\exists pc_0)$. We also know that on another read, the idempotence qualifier of x_0 does not evolve further and remains defined. Put formally, it follows by definition of δ that $\delta(pc_0, qID'_0, Rd) = qID'_0$ where $\Omega'_2(x_0) = \tau @ \langle qID'_0, qIO_0\rangle = \Omega_2(x_0)$.

Subcase 2: x_0 is not read by e_2. If x_0 is not read by e_2, then $\Omega_0(x_0) = \Omega'_2(x_0)$. By Lemma 35 applied to (1), we know that $\Omega_0 \preccurlyeq_{Rd} \Omega_1$, and hence it follows by definition of $\sqcup$ that $\Omega_1 \sqcup \Omega_0 = \Omega_1$. Therefore, $\Omega_1 \sqcup x_0{:}\tau @ \langle qID_0, qIO_0\rangle = \Omega_1$.

Put together, this is enough to show that $\Omega' = \Omega_1 \sqcup \Omega'_2$.
Therefore, we have established that

$$\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e_2 : \mathsf{C}^{\texttt{Md}}_{T_2} @ q \dashv \Omega'_2 \; (**)$$

By application of T-Binary to $(**)$, (1), (3), (4), (5), we establish the desired result:

$$\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e'_1 \odot e_2 : \mathsf{C}^{\texttt{Md}}_{T} @ q \dashv \Omega'$$

□

We provide a very similar lemma where instead the typing judgment of e_1 must be shifted. We choose to make it a separate lemma to avoid requiring that $\odot$ is reflexive, e.g., subtraction or division.

Lemma 33 (Ω shift for in-progress binary expressions, e_1) *If*

(1) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e_1 : \mathsf{C}^{\texttt{Md}}_{T_1} @ q_1 \dashv \Omega_1$

(2) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e'_2 : \mathsf{C}^{\texttt{Md}}_{T_2} @ q'_2 \dashv \Omega_2$ *and*

(3) $\odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T'$

(4) $\Omega' = \Omega_1 \sqcup \Omega_2$

(5) $\Omega \preccurlyeq_{Rd} \Omega_0$

then, $\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e_1 \odot e'_2 : \mathsf{C}^{\texttt{Md}}_{T} @ q_1 \sqcup_q q'_2 \dashv \Omega'$.

Proof: The proof is very similar to Lemma 32. □

Lemma 34 (Post-Ω is more strict than pre-Ω under write) *If* $\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\texttt{WT}} x : \downarrow\uparrow A @ q \dashv \Omega'$, *then* $\Omega \preccurlyeq_{Wt} \Omega'$.

Proof: By inversion of T-W-Shift on the assumed typing judgment:

T-W-Shift

$$\frac{\Sigma = \downarrow\Sigma^1 \qquad \Omega = \Omega^1, \Omega^2_{\texttt{CK}} \qquad \texttt{Md} \mid b : \texttt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\texttt{WT}} x : \uparrow A @ q \dashv \Omega_1 \qquad \Omega' = \Omega_1 \restriction \mathsf{dom}(\Omega^1), \Omega^2_{\texttt{CK}}}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\texttt{WT}} x : \downarrow\uparrow A @ q \dashv \Omega'}$$

we learn that:

(1) $\Sigma = \downarrow\Sigma^1$

(2) $\Omega = \Omega^1, \Omega^2_{\texttt{CK}}$

(3) $\texttt{Md} \mid b : \texttt{nat} \mid \Omega^1, \Sigma^1 \mid pcS \vdash_{\texttt{WT}} x : \uparrow A @ q \dashv \Omega_1$

(4) $\Omega' = \Omega_1 \restriction \mathsf{dom}(\Omega^1), \Omega^2_{\texttt{CK}}$

By inversion of T-Loc-Write on (3):

T-Loc-Write

$$\frac{\Omega^1, \Sigma^1 = x : \uparrow A @ q_0, \Omega'_2 \qquad q_0 = \langle qID_x, qIO_x \rangle \qquad pcS_t = pc \rhd pcS'_t \qquad qID' = \delta(pc, qID_x, Wt) \neq UN \qquad q = \langle qID', qIO_x \rangle \qquad \Omega_1 = x : \uparrow A @ q, \Omega'_2}{\texttt{Md} \mid b : \texttt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\texttt{WT}} x : \uparrow A @ q \dashv \Omega_1}$$

We observe that $\Omega^1, \Sigma^1 \preccurlyeq_{Wt} \Omega_1$ by definition. Therefore, it follows by (1), (2), (4), and (5) that $\mathsf{dom}(\Omega) = \mathsf{dom}(\Omega')$, and $\Omega \preccurlyeq_{Wt} \Omega'$. □

Lemma 35 (Post-Ω is more strict than pre-Ω under read) *If* $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_T^{\mathtt{Md}}@q \dashv \Omega'$, *then* $\Omega \preccurlyeq_{Rd} \Omega'$.

Proof: By induction on the structure of the assumption. In the case T-R-Shift, we invert on the assumption and case on the structure of the learned typing judgment. In all cases, we observe that $\Omega \preccurlyeq_{Rd} \Omega'$. □

Theorem 17 (preservation for expressions) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \geq 0 : nat \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \tau@\langle qID, qIO\rangle \dashv \Omega'$$

and for some $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$ *and natural number* $n \geq 0$ *and sets of qualifiers* $\{q_e\}$ *and* $\{q'_e\}$, *we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\{q_e\} \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@\{q'_e\}$$

then there exists q', Ω_0 *such that*

$$\mathtt{Md} \mid b \geq 0 : nat \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e' : \tau@\langle qID', qIO\rangle \dashv \Omega'$$

where $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega_0 \mid \Sigma \mid pcS_t$ *and* $n' \geq 0$, $\Omega \preccurlyeq_{Rd} \Omega_0$, *and* $qID \sqsubseteq_{id} qID'$.

Proof: The proof is by induction on the size of e. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\{q_e\} \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@\{q'_e\}$.

Case 1. [D-Binary-1]**.**

$$\frac{\begin{array}{c} n > 0 \\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1@\{q\} \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'_1@\{q'\} \end{array}}{\begin{array}{c} \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2@\{q, \langle \mathsf{Id}, \mathtt{Nt}\rangle\} \\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'_1 \odot e_2@\{q', \langle \mathsf{Id}, \mathtt{Nt}\rangle\} \end{array}} \text{(D-Binary-1)}$$

By the first premise of D-Binary-1, we have that $n > 0$. By inversion on the assumed typing judgment (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau@q \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau.$$

By inversion on $(\dagger_1)$ via T-Binary

(T-Binary)

$$\frac{\begin{array}{l}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1}@q_1 \dashv \Omega_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2}@q_2 \dashv \Omega_2 \\ \odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T' \qquad \Omega' = \Omega_1 \sqcup \Omega_2 \qquad q = q_1 \sqcup_q q_2\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_{T'}@q \dashv \Omega'}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1}@q_1 \dashv \Omega_1$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2}@q_2 \dashv \Omega_2$

(3) $q = q_1 \sqcup_q q_2$

(4) $\odot : \uparrow T_1 \times \uparrow T_2 \to \uparrow T'$

(5) $\Omega' = \Omega_1 \sqcup \Omega_2$

Note that trivially, q is a singleton set $\{q\}$. Then, by the inductive hypothesis applied to (1) and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1@\{q\} \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1'@\{q'\}$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' : \tau@q_1' \dashv \Omega_1 \quad (6)$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega_0 \mid \Sigma \mid pcS_t$, $n' \geq 0$, $\Omega \preccurlyeq_{Rd} \Omega_0$, and $q_1 = \langle qID_1, qIO_1 \rangle$, $q_1' = \langle qID_1', qIO_1 \rangle$, and $qID_1 \sqsubseteq_{id} qID_1'$.

By Lemma 32 applied to (2), (4), (5), (6), and $\Omega \preccurlyeq_{Rd} \Omega_0$, we learn that

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau@q_1' \sqcup_q q_2 \dashv \Omega'$$

Where $q_2 = \langle qID_2, qIO_2 \rangle$, we learn that $qID_1 \sqcup_q qID_2 \sqsubseteq_{id} qID_1' \sqcup_q qID_2$ from $qID_1 \sqsubseteq_{id} qID_1'$.

We consider two subcases based on $\mathtt{Md}$.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau@q_1' \sqcup_q q_2 \dashv \Omega'$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau$.

In both subcases, we have the typing judgments $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau$ and $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau@q_1' \sqcup_q q_2 \dashv \Omega'$. Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau@q_1' \sqcup_q q_2 \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1' \odot e_2 : \tau@q_1' \sqcup_q q_2 \dashv \Omega'} \text{ (T-enough?)}$$

Case 2 [D-Binary-2].

$$\frac{n > 0 \qquad Val(\mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1@q_1) \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_2@\{q\} \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_2'@\{q'\}}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2@\{q_1, q\} \\ \rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2'@\{q_1, q'\}\end{array}}\ \text{(D-Binary-2)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau@q \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \;=\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau.$$

By inversion on ($\dagger_1$) via T-Binary

$$\text{(T-Binary)}\ \frac{\begin{array}{l}\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \mathsf{C}^{\mathtt{Md}}_{T_1}@q_1 \dashv \Omega_1 \\ \mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \mathsf{C}^{\mathtt{Md}}_{T_2}@q_2 \dashv \Omega_2 \\ \odot : \uparrow T_1 \times \uparrow T_2 \rightarrow \uparrow T' \qquad \Omega' = \Omega_1 \sqcup \Omega_2 \qquad q = q_1 \sqcup_q q_2\end{array}}{\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \mathsf{C}^{\mathtt{Md}}_{T'}@q \dashv \Omega'}$$

we learn that

(1) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \tau@q_1 \dashv \Omega_1$

(2) $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \tau@q_2 \dashv \Omega_2$

(3) $\odot : \uparrow T_1 \times \uparrow T_2 \rightarrow \uparrow T'$

(4) $\Omega' = \Omega_1 \sqcup \Omega_2$

(6) $q = q_1 \sqcup_q q_2$

By the inductive hypothesis applied to $\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \tau@q_2 \dashv \Omega_2$ and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_2@\{q\} \;\rightarrow\; \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e_2'@\{q'\}$, it follows that

$$\mathtt{Md} \mid b \;\geq\; 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2' : \tau@q_2 \dashv \Omega_2 \;(*)$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega_0 \mid \Sigma \mid pcS_t$ and $n' \geq 0$ and $\Omega \preccurlyeq_{Rd} \Omega_0$, and $qID_2 \sqsubseteq_{id} qID_2'$ where $q_2 = \langle qID_2, qIO_2\rangle$, $q_2' = \langle qID_2', qIO_2\rangle$.
By Lemma 33 applied to (1), (3)-(6), and (∗):

$$\mathtt{Md} \mid b \;>\; 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau@q_1 \sqcup_q q_2' \vdash \Omega'; \Sigma$$

Where $q_2 = \langle qID_2, qIO_2\rangle$, we learn that $qID_1 \sqcup_q qID_2 \sqsubseteq_{id} qID_1 \sqcup_q qID_2'$ from $qID_2 \sqsubseteq_{id} qID_2'$.
We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau @ q_1 \sqcup_q q_2' \dashv \Omega'$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau @ q_1 \sqcup_q q_2' \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2' : \tau @ q_1 \sqcup_q q_2' \dashv \Omega'} \text{ (T-ENOUGH?)}$$

Case 3. [D-Binary-v].

$$\frac{\begin{array}{c}n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 @ q_1) \\ \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_2 @ q_2) \qquad v = e_1 \odot e_2 \qquad q = q_1 \sqcup_q q_2\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e_1 \odot e_2 @ \{q_1, q_2\} \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid v @ q} \text{ (D-Binary-v)}$$

By the first premise, we know that $n > 0$ and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau @ q \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau.$$

By inversion on ($\dagger_1$) via T-Binary

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \tau @ q_1 \dashv \Omega_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \tau @ q_2 \dashv \Omega_2 \\ \Omega' = \Omega_1 \sqcup \Omega_2 \qquad q = q_1 \sqcup_q q_2\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 \odot e_2 : \tau @ q \dashv \Omega'} \text{ (T-Binary)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \tau @ q_1 \dashv \Omega_1$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \tau @ q_2 \dashv \Omega_2$

From (1) and (2), we apply inversion on T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_1 : \uparrow A @ q_1 \dashv \Omega_1'$$

and

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\mathtt{RD;Sig}} e_2 : \uparrow A@q_2 \dashv \Omega'_2$$

for (i) $\Sigma =\downarrow \Sigma^1$ and (ii) $\Omega = \Omega^1, \Omega^2{}_{\mathsf{CK}}$. Additionally, (iii) $\Omega_1 = \Omega'_1 \restriction \mathsf{dom}(\Omega^1), \Omega^2_{\mathsf{CK}}$ and (iv) $\Sigma_1 = \downarrow(\Omega'_1 \restriction \mathsf{dom}(\Sigma^1))$ and (v) $\Omega_2 = \Omega'_2 \restriction \mathsf{dom}(\Omega^1), \Omega''_{\mathsf{CK}}$ and (vi) $\Sigma_2 = \downarrow(\Omega'_2 \restriction \mathsf{dom}(\Sigma^1))$.

By definition of $\odot$ operator and $q = q_1 \sqcup_q q_2$, we have $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \uparrow A@q \dashv \Omega^1, \Sigma^1$ for $v = e_1 \odot e_2$.

Furthermore, by Lemma 27 applied to $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$, (iii), and (iv), we know that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega^1, \Sigma^1 \mid pcS_t$.

By application of T-$\vee$-Succ and T-R-Shift and (i) and (ii), we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'$$

By definition of Ω' and Σ and $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega^1, \Sigma^1 \mid pcS_t$, we know that$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma \mid pcS_t$. We consider two subcases based on $\mathtt{Md}$.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By $(\dagger_2)$ via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau$.

In both subcases, Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'} \text{ (T-ENOUGH?)}$$

Case [D-Binary-0]**.** This case is straightforward, with the desired result following by assumption. Since the stepped configuration is not a value, we do not need to show the equality between its qualifiers and its qualifier in the typing.

Case 4. [D-V-Read]**.**

$$\text{D-V-READ}\quad \frac{\gamma = \gamma', [x \mapsto \ell] \qquad \mathsf{V} = \ell@q \hookrightarrow v, \mathsf{V}' \qquad n = n' + 1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ x@q \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}'' \mid\ pcS_o \mid v@q}$$

By the last premise $n > 0$ and $n' \geq 0$. By inversion on $(\dagger)$ via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \tau@q \dashv \Omega'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \tau.$$

By inversion on ($\dagger_1$) via T-$\vee$-Succ

$$\frac{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A@q \dashv \Omega'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : (\tau_1 \vee \downarrow\uparrow A)@q \dashv \Omega'} \text{ (T-}\vee\text{-Succ)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A@q \dashv \Omega'$.

By inversion on $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A@q \dashv \Omega'$ via T-R-Shift

$$\frac{\begin{array}{c}\Omega = \Omega^1, \Omega^2_{\mathsf{CK}} \\ \Sigma = \downarrow\Sigma^1 \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \uparrow A@q \dashv \Omega_1 \\ \Omega' = \Omega_1 \restriction \mathsf{dom}(\Omega^1), \Omega^2_{\mathsf{CK}} \qquad \Sigma = \downarrow(\Omega_1 \restriction \mathsf{dom}(\Sigma^1))\end{array}}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \downarrow\uparrow A@q \dashv \Omega'} \text{ (T-R-Shift)}$$

we learn that $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\mathtt{RD;Sig}} x : \uparrow A@q \dashv \Omega_1$ and $\Sigma = \downarrow\Sigma^1$ and $\Omega = \Omega^1, \Omega^2_{\mathsf{CK}}$.

By Lemma 27 applied to the assumption $\vdash^{\mathtt{Md}}_{\gamma}$ N | V | pcS_o : Ω | Σ | pcS_t and premise $\Sigma = \downarrow\Sigma^1$, we know that $\vdash^{\mathtt{Md}}_{\gamma}$ N | V | $pcS_o : \Omega^1, \Sigma^1 \mid pcS_t$. By definition of well-formedness, we know that $\gamma = \gamma', [x \mapsto \ell]$ and $\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^1, \Sigma^1 \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \uparrow A@q_v \dashv \Omega^1, \Sigma^1$.

By application of T-$\vee$-Succ and T-R-Shift and $\Sigma = \downarrow\Sigma^1$ and $\Omega = \Omega^1, \Omega^2_{\mathsf{CK}}$, and the definition of Σ which types variables with idempotence qualifier Id(CK) or flat Id/Nid that are unaffected by δ, we get

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \uparrow A@q_v \dashv \Omega'$$

By definition of Ω' and Σ and $\vdash^{\mathtt{Md}}_{\gamma}$ N | V | $pcS_o : \Omega^1, \Sigma^1 \mid pcS_t$, we know that$\vdash^{\mathtt{Md}}_{\gamma}$ N | V | $pcS_o : \Omega' \mid \Sigma \mid pcS_t$.

By definition, $Val(\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}'' \mid pcS_o \mid v@q)$, and hence it follows by Theorem 16 that $q = q_v$.

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. By $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'$ via lemma 28, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 29, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau$.

In both subcases, we have $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau$. Then the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} v : \tau@q \dashv \Omega'} \text{ (T-enough?)}$$

Case 5. [D-N-Read].

The proof is analogous to the previous case.

□

Definition 28 *We write* $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}, \gamma)$ *where* $x{:}\tau@q \in \Sigma'$ *iff* $\gamma = [x \mapsto \ell], \gamma'$ *and* $x{:}\tau@q \in \Sigma$ *and* $\ell \in dom(\mathsf{V})$.

Lemma 36 (equality of trimmed volatile contexts) *If*

(i) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}_0, \gamma_0)$
(ii) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$,
(iii) $\vdash^{\mathtt{Md}}_{\gamma''} \mathsf{N}' \mid \mathsf{V}' \mid pcS_o : \Omega \mid \Sigma'' \mid pcS_t$,
(iv) $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ *and* $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$
(v) $\gamma_0 \subseteq \gamma$ *and* $\gamma_0 \subseteq \gamma''$
then $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$.

Proof: We need to show that $\Sigma' = \mathsf{trim}(\Sigma'', \mathsf{V}_0, \gamma_0)$. The proof proceeds by proving each direction separately:

Ad ⇒. Let $x : \downarrow\uparrow A@q \in \Sigma'$. Then by (i), we have

(1) $x : \downarrow\uparrow A@q \in \Sigma$
(2) $\gamma_0 = [x \mapsto l], \gamma'_0$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma''$. From (2) and $\gamma_0 \subseteq \gamma''$, it follows that $\exists \gamma''_0 \supseteq \gamma'_0$ such that $\gamma'' = [x \mapsto l], \gamma''_0$ $(*)$. By (iv) and (3), we have that $l \in dom(\mathsf{V}')$. So, $\exists \mathsf{V}'_0$, v such that $\mathsf{V}' = \mathsf{V}'_0, l \hookrightarrow v$ $(**)$ and $\cdot \vdash v : \uparrow A$ $(***)$. Note that inverting (iii) via V-Loc yields the well-formedness judgment $\vdash^{\mathsf{Md}}_{\gamma''_0} \mathsf{N}' \mid \mathsf{V}'_0 \mid pcS_o : \Omega \mid \Sigma''_0 \mid pcS_t$ (†). Then, it follows by V-Loc applied to $(*)$, $(**)$, $(***)$, (†) that $x : \downarrow\uparrow A@q \in \Sigma''$.

Ad ⇐. Let

(1) $x : \downarrow\uparrow A@q \in \Sigma''$
(2) $\gamma_0 = [x \mapsto l], \gamma'_0$
(3) $l \in dom(\mathsf{V}_0)$

We need to show that $x : \downarrow\uparrow A@q \in \Sigma'$. To this end, it suffices to show that $x : \downarrow\uparrow A@q \in \Sigma$. By (3) and $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$, we have that $l \in dom(\mathsf{V})$. Therefore, $\exists\, \mathsf{V}'_0, v$ such that $\mathsf{V} = \mathsf{V}'_0, l@q \hookrightarrow v$ $(*)$ and $\cdot \vdash v : \uparrow A$ $(**)$. From (2) and $\gamma_0 \subseteq \gamma$, we have that $\exists\, \gamma' \supseteq \gamma'_0$ such that $\gamma = [x \mapsto l], \gamma'$. Note that inverting (ii) via V-Loc yields the well-formedness judgment $\vdash^{\mathsf{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'_0 \mid pcS_o : \Omega \mid \Sigma_0 \mid pcS_t$ $(***)$. By V-Loc applied to $(*)$, $(**)$, $(***)$, and $\gamma = [x \mapsto l], \gamma'$, we have

$$x : \downarrow\uparrow A@q \in \Sigma$$

Then it follows by definition 28 applied to (i) that

$$x : \downarrow\uparrow A@q \in \Sigma'.$$

We have shown that $x : \downarrow\uparrow A@q \in \Sigma'$ iff $x : \downarrow\uparrow A@q \in \Sigma''$, $\gamma = [x \mapsto l], \gamma'$, and $l \in dom(\mathsf{V})$. The desired result holds by definition 28. □

Lemma 37 (well-formedness of smaller memories) *If*

(i) $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$,

(ii) $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$,

(iii) $\Sigma' = \mathsf{trim}(\Sigma, \mathsf{V}', \gamma')$, *and*

(iv) $\gamma' \subseteq \gamma$

(v) $\mathsf{N} = \mathsf{N}^1_{\mathtt{CK}}, \mathsf{N}^2$

then $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}^2 \mid \mathsf{V}'' \mid pcS_o : \Omega \mid \Sigma' \mid pcS_t$.

Proof: By (2), observe that $\mathsf{V} \supseteq \mathsf{V}''$. The proof proceeds by induction on the size of $\mathsf{V} - \mathsf{V}''$.

Base case: $|\mathsf{V} - \mathsf{V}''| = 0$**.** If $|\mathsf{V} - \mathsf{V}''| = 0$, then $\mathsf{V} = \mathsf{V}''$ and the desired result holds by assumption (1).

Inductive case. Let $|\mathsf{V} - \mathsf{V}''| = k + 1$ where $|\mathsf{V}_0 - \mathsf{V}''| = k$ where $\mathsf{V} = \mathsf{V}_0, l@\mathtt{Ck} \hookrightarrow v$ and $\cdot \vdash v : \uparrow A$. Let $x{:}\downarrow\uparrow A@q \in \Sigma$ such that $\Sigma = \Sigma_0, (x{:}\downarrow\uparrow A@q)$. By inversion on V-Loc applied to the assumed judgment:

$$\frac{\gamma = [x \mapsto l], \gamma_0 \qquad \vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 \mid pcS_o : \Omega \mid \Sigma_0 \mid pcS_t \qquad \mathsf{V} = \mathsf{V}_0, \ell@q \hookrightarrow v \qquad \cdot \vdash v : \uparrow A@q}{\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma_0, (x{:}\downarrow\uparrow A@q)}\ \text{V-Loc}$$

we learn that

(1) $\vdash^{\mathtt{Md}}_{\gamma_0} \mathsf{N} \mid \mathsf{V}_0 \mid pcS_o : \Omega \mid \Sigma_0 \mid pcS_t$

(2) $\gamma = [x \mapsto l], \gamma_0$

(3) $\mathsf{V} = \mathsf{V}_0, l@q \hookrightarrow v$

(4) $\cdot \vdash v{:}\uparrow A@q$

Now we need to show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$.

Ad $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$**.** To show that $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it suffices to show that $l \notin dom(\mathsf{V}')$:

By assumption, it follows that $l \in \mathsf{V} - \mathsf{V}''$, or equivalently, $l \in \mathsf{V}$ and $l \notin \mathsf{V}''$. Now suppose that $l \in dom(\mathsf{V}')$. Then it follows by (ii) that $l \in \mathsf{V}''$, a contradiction. Therefore, we have $l \notin dom(\mathsf{V}')$.

Therefore, $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$ follows by $l \notin dom(\mathsf{V}')$ and (ii).

Ad $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$**.** To show $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we first need to show that

(a) $dom(\mathsf{V}') \subseteq dom(\mathsf{V}_0)$

(b) $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$

(c) $\gamma \supseteq \gamma'$

(d) $\gamma_0 \supseteq \gamma'$

(a) follows by $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$. Towards (b), note that since $\mathsf{V} \supseteq \mathsf{V}''$ by assumption, we have $dom(\mathsf{V}) \supseteq dom(\mathsf{V}'')$. By $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, it follows that $dom(\mathsf{V}'') = dom(\mathsf{V}')$. Therefore, $dom(\mathsf{V}') \subseteq dom(\mathsf{V})$ (b) holds. (c) follows by assumption (iv). By definition, $\gamma = [x \mapsto l], \gamma_0$, so γ_0 is the smallest subset of γ not containing $x \mapsto l$. Observe that γ' is a subset of γ that does not contain $x \mapsto l$. So, $\gamma \supseteq \gamma_0 \supseteq \gamma'$, which concludes the proof of (d).

$\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$ follows by lemma 36 applied to (iii), (a), (b), (c), and (d).

By the inductive hypothesis applied to (1), $\mathsf{V}'' = \mathsf{V}_0 \restriction dom(\mathsf{V}')$, and $\Sigma' = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma')$, we have $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' \mid pcS_o : \Omega \mid \Sigma' \mid pcS_t$. This is the desired result.

□

Lemma 38 (Monotonicity of volatile memories) *If* $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'$ *where* $c \neq c_1;_W c_2$, *then* $dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$, $\gamma \subseteq \gamma'$.

Proof: The proof is straightforward, proceeding in cases on the dynamic rules. □

Definition 29 (Ω-reads) $\Omega \preccurlyeq_{Rd} \Omega'$ *iff* $\forall x{:}\tau@\langle qID, qIO\rangle \in \Omega$, *it is the case that* $x{:}\tau@\langle qID', qIO\rangle \in \Omega'$ *and* $qID' \neq UN$, *where either*

- $\delta(pc, qID, Rd) = qID'$ $(\exists pc)$ *or*
- $qID = qID'$.

Definition 30 (Ω-writes) $\Omega \preccurlyeq_{Wt} \Omega'$ *iff* $\forall x{:}\tau@\langle qID, qIO\rangle \in \Omega$, *it is the case that* $x{:}\tau@\langle qID', qIO\rangle \in \Omega'$ *and* $qID' \neq UN$, *where either*

- $\delta(pc, qID, Wt) = qID'$ $(\exists pc)$ *or*
- $qID = qID'$.

We say that a context Ω' is reachable from Ω if there exists a series of delta or join operations between them. The delta transitions are accounted for by (i) and (ii), and join is accounted for by (iii). Since idempotence qualifiers in the unstable typing context do not change, reachability on Σ is defined where the idempotence qualifiers are the same. Since taint qualifiers can be replaced on assignment, they do not always relate from pre-context to post-context, so they are left out of this definition.

Definition 31 (Context Reachability) $\Omega \preccurlyeq \Omega'$ *iff there exists* Ω_0 *s.t.* $\Omega_0 \preccurlyeq \Omega'$ *and either*

(i) $\Omega \preccurlyeq_{Rd} \Omega_0$ *or*
(ii) $\Omega \preccurlyeq_{Wt} \Omega_0$ *or*
(iii) $\Omega \sqsubseteq \Omega_0$

We write $\Sigma \preccurlyeq \Sigma'$ *iff there exists* Σ_0 *s.t.* $\Sigma \sqsubseteq \Sigma_0$ *and* $\Sigma_0 \preccurlyeq \Sigma'$.

Lemma 39 *If* $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'$, *then* $dom(\Omega) \subseteq dom(\Omega')$.

Proof: The proof is by induction on the size of c. We consider possible cases for $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'; \Sigma'$.

Case [T-C-SHIFT]**.**

$$\frac{\Sigma = \downarrow\Sigma' \qquad \Omega = \Omega', \Omega''_{\mathsf{CK}} \qquad \mathtt{Md} \mid b : \mathtt{nat} \mid \Omega', \Sigma' \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{skip} : \uparrow\mathtt{unit} \dashv \Omega_1}{\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{skip} : \downarrow\uparrow\mathtt{unit} \dashv \Omega_1 \restriction \mathsf{dom}(\Omega'), \Omega''_{\mathsf{CK}}; \downarrow(\Omega_1 \restriction \mathsf{dom}(\Sigma))}\ \text{(T-C-SHIFT)}$$

By definition of $\restriction$, we obtain the desired result $dom(\Omega) \subseteq \mathsf{dom}(\Omega_1 \restriction \mathsf{dom}(\Omega'), \Omega''_{\mathsf{CK}})$.

Case [T-SEQ]**.**

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'; \Sigma' \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''; \Sigma''}\ \text{(T-SEQ)}$$

By inversion of T-SEQ, we learn that

- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'; \Sigma'$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''$

By applying the inductive hypothesis to each of these judgments, we learn that $dom(\Omega) \subseteq dom(\Omega')$ and $dom(\Omega') \subseteq dom(\Omega'')$. By transitivity, we establish the desired result $dom(\Omega) \subseteq dom(\Omega'')$.

Case [T-SEQ-D]**.** This case is similar to T-SEQ.

Case [T-$\vee$-SUCC, T-LET, T-ASSIGN, T-IF, T-ENOUGH?]**.** In each case, we apply the inductive hypothesis to learn that $dom(\Omega) \subseteq dom(\Omega')$.

□

Lemma 40 *If* $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD};\mathtt{Sig}} v : \mathsf{C}^{\mathtt{Md}}_A @ q \dashv \Omega'$ *and* $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid v@q)$ *where* $n = n' + 1$*, then* $\Omega' = \Omega$.

Proof: The proof is by cases on $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid v@q)$. In each case, we invert T-R-SHIFT on the assumed typing judgment and match the corresponding value typing rule to learn that $\Omega = \Omega'$. Note that by definition and by $n = n' + 1$, the values considered here are not crash instructions or variables. □

Theorem 6.2 (Preservation for Commands) *If*

$$(\dagger) \quad \mathtt{Md} \mid b \geq 0 : nat \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'$$

and $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c$ *is well-formed,* $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$*, and for some (co-)natural number* $n \geq 0$*, we have*

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'$$

then for some Σ_0 *and* Ω_0

$$\mathtt{Md} \mid b \ \geq\ 0 : \mathit{nat} \mid \Omega_0; \Sigma_0 \mid \mathit{pcS}'_t \vdash_{\mathtt{Sig}} c' : \tau \dashv \Omega'; \Sigma'$$

where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}'_o : \Omega_0 \mid \Sigma_0 \mid \mathit{pcS}'_t$, $\Omega \preccurlyeq \Omega_0$, $\Sigma \preccurlyeq \Sigma_0$, *and* $n' \geq 0$. *Moreover* $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}'_o \mid c'$ *is well-formed.*

Proof: The proof is by induction on the size of c. We proceed by considering possible cases for $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid c \ \rightarrow\ \gamma' \mid \mathtt{Md}' \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}'_o \mid c'$.

Case 1 [D-Let]**.**

$$\frac{\begin{array}{c} n > 0 \\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid \mathit{pcS}_o \mid e'@q' \end{array}}{\begin{array}{c} \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid \mathbf{let}\, x = e\, \mathbf{in}\, c \\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid \mathit{pcS}_o \mid \mathbf{let}\, x = e'@q'\, \mathbf{in}\, c \end{array}}\ (\text{D-Let})$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b \ >\ 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{Sig}} \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau \dashv \Omega'; \Sigma'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b \ =\ 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{Sig}''} \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}'' = \{\mathtt{Md} \mid b \ \geq\ 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash \mathbf{let}\, x = e\, \mathbf{in}\, c : \tau\}$.

By inversion on ($\dagger_1$) via T-Let

$$\text{T-Let}\ \frac{\begin{array}{c} \mathit{pcS}_t = \mathit{pc}_t \rhd \mathit{pcS}'_t \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle \mathit{qID}, \mathit{qIO}\rangle \dashv \Omega_0 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma, x{:}\downarrow\uparrow A@\langle \overline{\mathit{qID}}, \mathit{qIO}\rangle \sqcup \mathit{pc}_t \mid \mathit{pcS}_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{Sig}} \mathsf{let}\, x = e\, \mathsf{in}\, c : \tau \dashv \Omega'; \Sigma'}$$

we learn that

(1) $\mathtt{Md} \mid b \ \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle \mathit{qID}, \mathit{qIO}\rangle \dashv \Omega_0$

(2) $\mathtt{Md} \mid b \ \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma, x{:}\downarrow\uparrow A\ @\langle \overline{\mathit{qID}}, \mathit{qIO}\rangle \sqcup \mathit{pc}_t \mid \mathit{pcS}_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'$

By application of Theorem 17 to (1) and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V}' \mid \mathit{pcS}_o \mid e'@q'$ and the assumption $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS}_o : \Omega \mid \Sigma \mid \mathit{pcS}_t$, it follows that $\exists \Omega_1$ such that

$$\mathtt{Md} \mid b \ \geq\ 0 : \mathtt{nat} \mid \Omega_1; \Sigma \mid \mathit{pcS}_t \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}^{\mathtt{Md}}_A@\langle \mathit{qID}', \mathit{qIO}'\rangle \dashv \Omega_0$$

where $n' \geq 0$, $\Omega \preccurlyeq_{Rd} \Omega_1$, and $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid \mathit{pcS}_o : \Omega_1 \mid \Sigma \mid \mathit{pcS}_t$. Therefore, by definition we know that $\Omega \preccurlyeq \Omega_1$.

By the following application of the T-Let rule we get

$$\frac{\begin{array}{c} pcS_t = pc_t \rhd pcS_t' \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}}@\langle qID', qIO' \rangle \dashv \Omega_0 \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma, x{:}{\downarrow}{\uparrow}A\ @\ \langle \overline{qID'}, qIO' \rangle \sqcup pc_t \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma' \end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau \dashv \Omega'; \Sigma'}\ \text{(T-Let)}$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau\}$. It follows via lemma 30, we get $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau$.

Subcase 2. [Md = aID(c_0)]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c} \mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}_2} \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau \\ \mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau \dashv \Omega'; \Sigma' \end{array}}{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathbf{let}\ x = e'\ \mathbf{in}\ c : \tau \dashv \Omega'; \Sigma'}\ \text{(T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathbf{let}\ x = e'\ \mathbf{in}\ c$ follows by definition 26, and $\Omega \preccurlyeq \Omega_1$ was shown earlier and $\Sigma \preccurlyeq \Sigma$ follows vacuously.

Case 2. [D-Let-v].

$$\frac{\begin{array}{c} pcS_o = pc_o \rhd pcS_o' \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ e@q) \\ \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\ \mathit{fresh} \qquad n = n' + 1 \end{array}}{\begin{array}{c} \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathbf{let}\ x = e@q\ \mathbf{in}\ c \\ \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}, \ell@q \sqcup pc_o \hookrightarrow e \mid pcS_o \mid c \end{array}}\ \text{(D-Let-v)}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \mid pcS_t \vdash \mathbf{let}\ x = e\ \mathbf{in}\ c : \tau \dashv \Omega'; \Sigma'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid\ \Omega; \Sigma \mid pcS_t \vdash \mathbf{let}\ x = e\ \mathbf{in}\ c : \tau.$$

By inversion of ($\dagger_1$) via T-Let

$$\frac{\begin{array}{c} pcS_t = pc_t \rhd pcS_t' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID, qIO \rangle \dashv \Omega_0 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma, x{:}{\downarrow}{\uparrow}A@\langle \overline{qID}, qIO \rangle \sqcup pc_t \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{let}\ x = e\ \mathsf{in}\ c : \tau \dashv \Omega'; \Sigma'}$$
T-Let

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID, qIO\rangle \dashv \Omega_0$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma, x{:}\downarrow\uparrow A@\langle \overline{qID}, qIO\rangle \sqcup pc_t \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'$

where $pcS_t = pc_t \rhd pcS'_t$.

By Theorem 16, since $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$ and $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID, qIO\rangle \dashv \Omega_0$, and also due to the well-formedness of volatile memory, we know that $q = \langle \overline{qID}, qIO\rangle$.

Since $q = \langle \overline{qID}, qIO\rangle$, the typing judgment (2) completes the proof, if we can show that $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}, \ell@q \hookrightarrow e : \Omega \mid \Sigma, x{:}\downarrow\uparrow A@\langle \overline{qID}, qIO\rangle$.

By $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$, we can apply inversion on $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@q \dashv \Omega_0$ via T-Enough?, T-$\vee$-Succ, and T-R-Shift to get

$$\mathtt{Md} \mid b : \mathtt{nat} \mid \Omega^0 \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \uparrow A@q \dashv \Omega^1$$

where $\Sigma = \downarrow\Sigma^2$ and $\Omega = \Omega^2{}_{\mathsf{CK}}, \Omega^3$ and $\Omega^0 = \Sigma^2, \Omega^3$ and $\Omega_0 = \Omega^2_{\mathsf{CK}}, \Omega^1 \restriction \mathsf{dom}(\Omega)$.

This is enough to prove $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}, \ell@q \hookrightarrow e : \Omega \mid \Sigma, x{:}\downarrow\uparrow A@q$.

Furthermore, since $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$, it follows by Lemma 40 that $\Omega_0 = \Omega^1$, and hence, $\Omega = \Omega_0$. Additionally, observe that by the well-formedness definition, we know that $pc_t = pc_o$. Therefore, it follows by V-Loc that

$$\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}, \ell@q \sqcup pc_o \hookrightarrow e : \Omega_0 \mid \Sigma, x{:}\downarrow\uparrow A@q \sqcup pc_t$$

Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}, \ell@q \hookrightarrow e \mid pcS_o \mid c$ follows by definition 26, vacuously because c is not of the form $c';_W c''$.

Case 3 [D-Assign]**.**

$$\frac{n > 0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@q'}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid x := e \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid x := e'@q'}\ \text{(D-Assign)}$$

By the first premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'; \Sigma'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash x := e : \tau\}$.

There are two cases to consider: $\ell \in \mathsf{dom}(\mathsf{N}^2)$ (persistent) and $\ell \in \mathsf{dom}(\mathsf{V}, \mathsf{N}^1)$ (temporary), where $\mathsf{N} = \mathsf{N}^1_{\mathsf{ck}}, \mathsf{N}^2$ and $x \mapsto \ell \in \gamma$.

Subcase $\ell \in \mathsf{dom}(\mathsf{N}^2)$**.** By the definition of well-formedness, we know that $x \in \mathsf{dom}(\Omega)$. By inversion on $(\dagger_1)$ via T-Assign-s

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega_1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_1 \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \\ qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \qquad x \in \mathsf{dom}(\Omega) \\ \Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0 \qquad \Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega''; \Sigma}\ \text{T-Assign-s}$$

we have

- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega_1$
- $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega'$ $(\dagger)$
- $pcS_t = pc \rhd pcS'_t$
- $pc = \langle qID_{pc}, qIO_{pc}\rangle$
- $qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x$
- $qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x$
- $x \in \mathsf{dom}(\Omega)$
- $\Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0$
- $\Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0$

By the application of Theorem 17 to $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega_1$ and the premise $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'@q'$ and assumption $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} : \Omega \mid \Sigma$, it follows that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega_1$$

where $n' \geq 0$, $\Omega \preccurlyeq_{Rd} \Omega_2$, and $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N}' \mid \mathsf{V} : \Omega_2 \mid \Sigma$. By definition, we know that $\Omega \preccurlyeq \Omega_2$.

By the following application of the T-Assign-s rule we get

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e' : \mathsf{C}_A^{\mathtt{Md}}@\langle qID_e, qIO_e\rangle \dashv \Omega_1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \\ qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \qquad x \in \mathsf{dom}(\Omega) \\ \Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0 \qquad \Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega''}\ \text{T-Assign-s}$$

Subcase $\ell \in \mathsf{dom}(\mathsf{V}, \mathsf{N}^1)$**.** This case follows similar reasoning, instead using the rule T-Assign-u to establish that $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega''$.

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash x := e' : \tau\}$. It follows by lemma 30 that $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}_1} x := e' : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e' : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash x := e' : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}_2} x := e' : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega''; \Sigma\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_2; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e' : \tau \dashv \Omega''; \Sigma}\ (\text{T-ENOUGH?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid x := e'@q'$ follows by definition 26, vacuously.

Note that $\Omega \preccurlyeq \Omega_2$ and $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} : \Omega_2 \mid \Sigma$ were established above.

Case [D-Assign-step]**.** This case follows similar reasoning to the previous case.

Case [D-Assign-V]**.**

D-Assign-V

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q) \\ \gamma = \gamma', [x \to \ell] \qquad \mathsf{V} = \mathsf{V}', \ell@\langle qID, qIO\rangle \hookrightarrow v' \qquad q = \langle qID_e, qIO_e\rangle \\ pcS_o = pc \rhd pcS'_o \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad q' = \langle qID, qIO_e \sqcup_t qIO_{pc}\rangle \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid x := e@q \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid pcS_o \mid \mathsf{skip}}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'; \Sigma''$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash x := e : \tau\}$.

By T-Skip, we know that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \dashv \Omega'; \Sigma'' \ (*)$$

Since skip is not of the form $c_1;_W c_2$, trivially the configuration $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid pcS_o \mid \mathsf{skip}$ is well-formed.

All that remains to be shown is that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid pcS_o : \Omega' \mid \Sigma'' \mid pcS_t$.

Since $\ell \in \mathsf{dom}(\mathsf{V})$, it follows by the definition of well-formedness that $x \in \mathsf{dom}(\Sigma)$.

By inversion on $(\dagger_1)$ via T-Assign

$$\text{T-Assign-u}\quad \frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle qID_e, qIO_e\rangle \dashv \Omega^1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \\ qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \qquad x \in \mathsf{dom}(\Sigma) \\ \Sigma = x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle, \Sigma_0 \qquad \Sigma'' = x : \downarrow\uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Sigma_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'; \Sigma''}$$

we have

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle qID_e, qIO_e\rangle \dashv \Omega^1$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega'$

(3) $pcS_t = pc \rhd pcS'_t$

(4) $pc = \langle qID_{pc}, qIO_{pc}\rangle$

(5) $qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x$

(6) $qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x$

(7) $x \in \mathsf{dom}(\Sigma)$

(8) $\Sigma' = x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle, \Sigma_0$

(9) $\Sigma'' = x : \downarrow\uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Sigma_0$

By Lemma 40 applied to (1), and since $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$, we know that $\Omega = \Omega^1$ and hence $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega^1 \mid \Sigma \mid pcS_t$.

By Lemma 34 applied to (2), we know that $\Omega^1 \preccurlyeq_{Wt} \Omega'$ and $\Sigma^1 \preccurlyeq_{Wt} \Sigma'$, and hence it follows by the well-formedness definitions that

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma' \mid pcS_t$$

By inversion of V-Loc and by (8), we know that:

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V}' \mid pcS_o : \Omega' \mid \Sigma_0 \mid pcS_t$$

Then, by V-Loc, (9), definition of q', and that $qID = qID_x$ (by definition, $\delta(pc, qID, Wt) = qID$ where $qID \in \{\mathsf{Id}, \mathsf{Nid}\}$) it follows that

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid pcS_o : \Omega' \mid \Sigma'' \mid pcS_t$$

We consider two subcases based on Md.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\}$. From lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash \mathbf{skip} : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}_2} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma''\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma''}\ (\text{T-ENOUGH?})$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N} \mid \mathsf{V}', \ell@q' \hookrightarrow e \mid \mathbf{skip}$ follows by definition 26, vacuously.

It remains to establish $\Omega \preccurlyeq \Omega'$ and $\Sigma \preccurlyeq \Sigma''$. By Lemma 35 applied to (1), we know that $\Omega \preccurlyeq_{Rd} \Omega^1$, and hence by definition that $\Omega \preccurlyeq \Omega^1$. By Lemma 34 applied to (2), we know that $\Omega^1 \preccurlyeq_{Wt} \Omega'$ and $\Sigma \preccurlyeq_{Wt} \Sigma'$, and hence by definition that $\Omega^1 \preccurlyeq \Omega'$ (†) and $\Sigma \preccurlyeq \Sigma'$ (‡). The desired result follows by (†) and ‡ and (9).

Case [D-Assign-N]**.**

D-Assign-N

$$\frac{\begin{array}{c}\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q) \qquad q = \langle qID_e, qIO_e\rangle \qquad \gamma = \gamma', [x \to \ell] \\ \mathsf{N} = \mathsf{N}', \ell@\langle qID, qIO\rangle \hookrightarrow v' \qquad pcS_o = pc \rhd pcS'_o \qquad qID' = \delta(pc, qID, Wt) \neq \mathit{UN} \\ pc = \langle qID_{pc}, qIO_{pc}\rangle \qquad q' = \langle qID', qIO_e \sqcup_t qIO_{pc}\rangle \qquad n = n' + 1\end{array}}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid x := e@q \ \to\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o \mid \mathsf{skip}}$$

By the last premise $n > 0$, and $n' \geq 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega^n; \Sigma^n$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} x := e : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash x := e : \tau\}$.

By T-Skip, we know that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{skip} : \tau \dashv \Omega^n; \Sigma^n\ (*)$$

Since skip is not of the form $c_1;_W c_2$, trivially the configuration $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o \mid \mathsf{skip}$ is well-formed.

All that remains to be shown is that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o : \Omega^n \mid \Sigma^n \mid pcS_t$.
Since $\ell \in \mathsf{dom}(\mathsf{N})$, it follows by the definition of well-formedness that either $x \in \mathsf{dom}(\Sigma)$ or $x \in \mathsf{dom}(\Omega)$.

Subcase $x \in \mathsf{dom}(\Sigma)$**.** If $x \in \mathsf{dom}(\Sigma)$, we let $\Omega^n = \Omega'$ and $\Sigma^n = \Sigma''$ and invert ($\dagger_1$) on T-Assign-u

$$\text{T-Assign-u}\quad \frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle qID_e, qIO_e\rangle \dashv \Omega^1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \\ qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \qquad x \in \mathsf{dom}(\Sigma) \\ \Sigma' = x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle, \Sigma_0 \qquad \Sigma'' = x : \downarrow\uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Sigma_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega'; \Sigma''}$$

to learn

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle qID_e, qIO_e\rangle \dashv \Omega^1$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega'$

(3) $pcS_t = pc \rhd pcS'_t$

(4) $pc = \langle qID_{pc}, qIO_{pc}\rangle$

(5) $qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x$

(6) $qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x$

(7) $x \in \mathsf{dom}(\Sigma)$

(8) $\Sigma' = x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle, \Sigma_0$

(9) $\Sigma'' = x : \downarrow\uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Sigma_0$

By Lemma 40 applied to (1), and since $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$, we know that $\Omega = \Omega^1$ and hence it follows by assumption that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega^1 \mid \Sigma \mid pcS_t$.
By Lemma 34 applied to (2), we know that $\Omega^1 \preccurlyeq_{Wt} \Omega'$, and hence it follows by the well-formedness definitions that

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma' \mid pcS_t$$

If $x \in \mathsf{dom}(\Sigma)$, then it must be that $qID_x = \mathsf{Id}(\mathtt{CK})$. By inversion of NV-loc-aid-2 and by (8), we know that:

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma_0 \mid pcS_t$$

By inversion of T-W-Shift and T-Loc-Write, we learn that $qID_x = \delta(pc, qID_0, Wt) \neq UN$. By the well-formedness of nonvolatile memory, we know that $qID_0 = qID$.

Hence, $qID = qID_x$. Then, by NV-LOC-AID-2, (9), definition of q' and qID', it follows that

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma'' \mid pcS_t$$

It remains to establish $\Omega \preccurlyeq \Omega'$ and $\Sigma \preccurlyeq \Sigma''$. By Lemma 35 applied to (1), we know that $\Omega \preccurlyeq_{Rd} \Omega^1$, and hence by definition that $\Omega \preccurlyeq \Omega^1$. By Lemma 34 applied to (2), we know that $\Omega^1 \preccurlyeq_{Wt} \Omega'$ and $\Sigma \preccurlyeq_{Wt} \Sigma'$, and hence by definition that $\Omega^1 \preccurlyeq \Omega'$ (†) and $\Sigma \preccurlyeq \Sigma'$ (‡). The desired result follows by (†) and ‡ and (9).

Subcase $x \in \mathsf{dom}(\Omega)$**.** If $x \in \mathsf{dom}(\Omega)$, we let $\Omega^n = \Omega''$ and $\Sigma^n = \Sigma'$ and invert ($\dagger_1$) on T-ASSIGN-S

$$\text{T-ASSIGN-S}\quad \frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle qID_e, qIO_e\rangle \dashv \Omega^1 \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega' \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID_{pc}, qIO_{pc}\rangle \\ qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x \qquad qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x \qquad x \in \mathsf{dom}(\Omega) \\ \Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0 \qquad \Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} x := e : \tau \dashv \Omega''; \Sigma}$$

to learn

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} e : \mathsf{C}^{\mathtt{Md}}_A@\langle qID_e, qIO_e\rangle \dashv \Omega^1$

(2) $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^1; \Sigma \mid pcS_t \vdash_{\mathtt{WT}} x : \downarrow\uparrow A@\langle qID_x, qIO_x\rangle \dashv \Omega'$

(3) $pcS_t = pc \rhd pcS'_t$

(4) $pc = \langle qID_{pc}, qIO_{pc}\rangle$

(5) $qID_{pc} \sqcup_{id} qID_e \sqsubseteq_{id} qID_x$

(6) $qIO_{pc} \sqcup_t qIO_e \sqsubseteq_t qIO_x$

(7) $x \in \mathsf{dom}(\Omega)$

(8) $\Omega' = x : \uparrow A@\langle qID_x, qIO_x\rangle, \Omega_0$

(9) $\Omega'' = x : \uparrow A@\langle qID_x, qIO_{pc} \sqcup_t qIO_e\rangle, \Omega_0$

By Lemma 40 applied to (1), and since $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@q)$, we know that $\Omega = \Omega^1$, and hence $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega^1 \mid \Sigma \mid pcS_t$.

By Lemma 34 applied to (2), we know that $\Omega^1 \preccurlyeq_{Wt} \Omega'$, and hence it follows by the well-formedness definitions that

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma \mid pcS_t$$

Since $x \in \mathsf{dom}(\Omega)$, it follows by inversion of the rule NV-LOC-AID-1 on (8):

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid pcS_o : \Omega_0 \mid \Sigma \mid pcS_t$$

By inversion of T-W-Shift and T-Loc-Write, we learn that $qID_x = \delta(pc, qID_0, Wt) \neq UN$ and $\Omega^1 \preccurlyeq \Omega'$ $(*)$ by definition. By the well-formedness of nonvolatile memory, we know that $qID_0 = qID$. Hence, $qID = qID_x$. Then, by NV-loc-aid-1, (9), definition of q' and qID', it follows that

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o : \Omega'' \mid \Sigma \mid pcS_t$$

By Lemma 35 applied to (1), we know that $\Omega \preccurlyeq_{Rd} \Omega^1$, and hence by definition that $\Omega \preccurlyeq \Omega^1$. Therefore, by $(*)$ it follows that $\Omega \preccurlyeq \Omega''$.

In both cases $\exists \Omega^n, \Sigma^n$ s.t. $\Omega \preccurlyeq \Omega^n$ and $\Sigma \preccurlyeq \Sigma^n$ and:

$$\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid pcS_o : \Omega^n \mid \Sigma^n \mid pcS_t$$

We consider two subcases based on $\mathtt{Md}$.

Subcase 1. $[\mathtt{Md} = \mathtt{Jit}]$. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}\}$. From lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

Subcase 2. $[\mathtt{Md} = \mathtt{aID}(c_0)]$. By $(\dagger_2)$ via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}}$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash \mathbf{skip} : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash_{\mathtt{Sig}_2} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega^n; \Sigma^n\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega^n; \Sigma^n \vdash_{\mathtt{Sig}} \mathbf{skip} : \mathsf{C}_{\mathtt{unit}} \dashv \Omega^n; \Sigma^n}\ \text{(T-enough?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{N}', \ell@q' \hookrightarrow e \mid \mathsf{V} \mid \mathbf{skip}$ follows by definition 26, vacuously.

Case 6 [D-If]**.**

$$\frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e'@q'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 \\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathbf{if}\ e'@q'\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2\end{array}}\ \text{(D-If)}$$

By the first premise $n > 0$. By inversion on $(\dagger)$ via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau \dashv \Omega'; \Sigma'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} \mathbf{if}\ e\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2 : \tau$$

where $\texttt{Sig}'' = \mathit{if}\ \texttt{Md} = \mathsf{jit}\ \mathit{then}\ \texttt{Sig}'\ \mathit{else}\ \texttt{Sig}$ and $\texttt{Sig}' = \{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \vdash \textbf{if}\ e\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau\}$.

By inversion on $(\dagger_1)$ via T-If

$$\text{T-If}\ \frac{\begin{array}{c}\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e : \mathsf{C}^{\texttt{Md}}_{\texttt{bool}}@\langle qID_e, qIO_e\rangle \dashv \Omega_0 \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID, qIO\rangle \qquad pc' = \langle qID \sqcup_{id} qID_e, qIO \sqcup_t qIO_e\rangle \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pc' \rhd pcS_t \vdash_{\texttt{Sig}} c_1 : \tau \dashv \Omega_1; \Sigma_1 \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pc' \rhd pcS_t \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega_2; \Sigma_2 \\ \Omega' = \mathsf{lift}(pc, \Omega_1 \sqcup \Omega_2) \qquad \Sigma' = \Sigma_1 \sqcup \Sigma_2\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\texttt{Sig}} \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \tau \dashv \Omega'; \Sigma'}$$

we learn that

(1) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e : \mathsf{C}^{\texttt{Md}}_{\texttt{bool}}@\langle qID_e, qIO_e\rangle \dashv \Omega_0$

(2) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pc' \rhd pcS_t \vdash_{\texttt{Sig}} c_1 : \tau \dashv \Omega_1; \Sigma_1$

(3) $\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pc' \rhd pcS_t \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega_2; \Sigma_2$

(4) $pcS_t = pc \rhd pcS'_t$

(5) $pc = \langle qID, qIO\rangle$

(6) $pc' = \langle qID \sqcup_{id} qID_e, qIO \sqcup_t qIO_e\rangle$

(7) $\Omega' = \mathsf{lift}(pc, \Omega_1 \sqcup \Omega_2)$

(8) $\Sigma' = \Sigma_1 \sqcup \Sigma_2$

By the application of Theorem 17 on (1) and $\gamma \mid \texttt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid e \rightarrow \gamma \mid \texttt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid e'$, it follows that for some $\langle qID'_e, qIO'_e\rangle, \Omega'_0$

$$\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega'_0; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e' : \mathsf{C}^{\texttt{Md}}_{\texttt{bool}}@\langle qID'_e, qIO'_e\rangle \dashv \Omega_0$$

where $n' \geq 0$ and $\Omega \preccurlyeq \Omega'_0$.

By the following application of the T-If rule we get

$$\frac{\begin{array}{c}\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega'_0; \Sigma \mid pcS_t \vdash_{\texttt{RD;Sig}} e' : \mathsf{C}^{\texttt{Md}}_{\texttt{bool}}@\langle qID'_e, qIO'_e\rangle \dashv \Omega_0 \\ pcS_t = pc \rhd pcS'_t \qquad pc = \langle qID, qIO\rangle \qquad pc'' = \langle qID \sqcup_{id} qID'_e, qIO \sqcup_t qIO'_e\rangle \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pc'' \rhd pcS_t \vdash_{\texttt{Sig}} c_1 : \tau \dashv \Omega'_1; \Sigma'_1 \\ \texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega_0; \Sigma \mid pc'' \rhd pcS_t \vdash_{\texttt{Sig}} c_2 : \tau \dashv \Omega'_2; \Sigma'_2 \\ \Omega'' = \mathsf{lift}(pc, \Omega'_1 \sqcup \Omega'_2) \qquad \Sigma'' = \Sigma'_1 \sqcup \Sigma'_2\end{array}}{\texttt{Md} \mid b > 0 : \texttt{nat} \mid \Omega'_0; \Sigma \mid pcS_t \vdash_{\texttt{Sig}} \textbf{if}\ e'\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau \dashv \Omega''; \Sigma''}\ \text{(T-If)}$$

We consider two subcases based on Md.

Subcase 1. [Md = Jit]. Let $\texttt{Sig}_1 = \{\texttt{Md} \mid b \geq 0 : \texttt{nat} \mid \Omega'_0; \Sigma \mid pcS_t \vdash \textbf{if}\ e'\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau\}$. By lemma 30, we get $\texttt{Md} \mid b = 0 : \texttt{nat} \mid \Omega'_0; \Sigma \mid pcS_t \vdash_{\texttt{Sig}_1} \textbf{if}\ e'\ \textbf{then}\ c_1\ \textbf{else}\ c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0'; \Sigma \mid pcS_t \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau$.

In both subcases, the desired result follows by T-Enough?:

$$\frac{\begin{array}{c}\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0'; \Sigma \mid pcS_t \vdash \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega_0'; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}_2} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0'; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega''; \Sigma''\end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0'; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega''; \Sigma''}\ \text{(T-ENOUGH?)}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_t \mid \textbf{if } e' \textbf{ then } c_1 \textbf{ else } c_2$ follows by definition 26, and $\Omega \preccurlyeq \Omega_0'$ and $\Sigma \preccurlyeq \Sigma_0'$ follow by definition, both vacuously.

Case [D-If-step]**.** This case follows similar reaoning to the previous case.

Case 7. [D-If-tt]**.**

D-If-tt

$$\frac{n = n' + 1 \qquad \mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{tt}@q_v) \qquad pc_o' = pc_o \sqcup_q q_v \qquad pcS_o = pc_o \rhd pcS_o^0}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{if\,tt}@q_v \mathsf{\,then\,} c_1 \mathsf{\,else\,} c_2 \\ \rightarrow\ \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pc_o' \rhd pcS_o \mid\ c_1; \mathsf{end}_{\mathsf{if}}\end{array}}$$

By the first premise $n > 0$, and $n' \geq 0$. By inversion on ($\dagger$) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau \dashv \Omega'; \Sigma'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2 : \tau\}$.

By inversion on ($\dagger_1$) via T-If

T-If

$$\frac{\begin{array}{c}\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} \mathsf{tt} : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}}@\langle qID_e, qIO_e\rangle \dashv \Omega_0 \\ pcS_t = pc_t \rhd pcS_t' \qquad pc_t = \langle qID, qIO\rangle \qquad pc_t' = \langle qID \sqcup_{id} qID_e, qIO \sqcup_t qIO_e\rangle \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pc_t' \rhd pcS_t \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega_1; \Sigma_1 \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pc_t' \rhd pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega_2; \Sigma_2 \\ \Omega' = \mathsf{lift}(pc_t, \Omega_1 \sqcup \Omega_2) \qquad \Sigma' = \Sigma_1 \sqcup \Sigma_2\end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{if\,tt\,then\,} c_1 \mathsf{\,else\,} c_2 : \tau \dashv \Omega'; \Sigma'}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{RD;Sig}} \mathsf{tt} : \mathsf{C}^{\mathtt{Md}}_{\mathtt{bool}} @ \langle qID_e, qIO_e \rangle \dashv \Omega_0$

(2) $pcS_t = pc_t \rhd pcS'_t$

(3) $pc_t = \langle qID, qIO \rangle$

(4) $pc'_t = \langle qID \sqcup_{id} qID_e, qIO \sqcup_t qIO_e \rangle$

(5) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pc'_t \rhd pcS_t \vdash_{\mathtt{Sig}} c_1 : \tau \dashv \Omega_1; \Sigma_1$

(6) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pc'_t \rhd pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega_2; \Sigma_2$

(7) $\Omega' = \mathsf{lift}(pc_t, \Omega_1 \sqcup \Omega_2)$

(8) $\Sigma' = \Sigma_1 \sqcup \Sigma_2$

By Theorem 16 applied to $Val(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid \mathsf{tt}@q_v)$, we know that $\Omega = \Omega_0$. By the assumption $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$, we know that $q_v = \langle qID_e, qIO_e \rangle$.

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1; \mathsf{end}_{\mathsf{if}}$ follows by definition 26, vacuously because c_1 is not of the form $c';_W c''$. The detail $\Omega \preccurlyeq \Omega$ follows vacuously by definition.

The typing judgment (2) completes the proof; note that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$ holds by assumption, which implies that $pc_o = pc_t$. Since $q_v = \langle qID_e, qIO_e \rangle$ and $\Omega = \Omega_0$, it follows that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pc'_o \rhd pcS_o : \Omega_0 \mid \Sigma \mid pc'_t \rhd pcS_t$.

Case 8 [D-If-ff]**.** Similar to the previous case.

Case 9 [D-seq]**.**

$$\frac{n > 0}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ c_1; c_2 \ \rightarrow\ \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1;_{\gamma \mid \mathsf{V}} c_2} \text{ (D-seq)}$$

By the premise $n > 0$. By inversion on (†) via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1; c_2 : \tau \dashv \Omega''; \Sigma''$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} c_1; c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}'\ \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c_1; c_2 : \tau\}$.

By inversion on $(\dagger_1)$ via T-Seq

$$\frac{\begin{array}{l}\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma' \\ \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega'; \Sigma' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''\end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1; c_2 \ : \tau \dashv \Omega''; \Sigma''} \text{ (T-Seq)}$$

we learn that

(1) $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma'$

(2) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''$

Let $W = \gamma \mid \mathsf{V}$. To finish the proof, it remains to be shown that

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma''' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''$$

where $\Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2$ and $\Sigma''' = \mathtt{trim}(\Sigma', \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{ck}}$.

By definition of `trim` and the world W, the desired result holds from (2).

Let $x \in \mathsf{dom}(\Sigma')$ where $[x \mapsto \ell] \in \gamma$ be arbitrary such that $\ell \notin \mathsf{dom}(\mathsf{V})$.

By the following application of the T-SEQ-D rule

$$\text{T-SEQ-D}\ \frac{\begin{array}{c} W = \gamma \mid \mathsf{V} \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}^{\mathtt{Md}}_{\mathtt{unit}} \dashv \Omega'; \Sigma' \\ \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2 \qquad \Sigma''' = \mathtt{trim}(\Sigma', \mathsf{V}, \gamma) \cup \downarrow\Omega^1_{\mathsf{ck}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma''' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma'' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''; \Sigma''}$$

we learn that $\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''; \Sigma''$

We consider two subcases based on `Md`.

Subcase 1. [$\mathtt{Md} = \mathtt{Jit}$]. Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c_1;_W c_2 : \tau\}$. By lemma 30, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c_1;_W c_2 : \tau$.

Subcase 2. [$\mathtt{Md} = \mathtt{aID}(c_0)$]. By ($\dagger_2$) via lemma 31, we get $\mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c_1;_W c_2 : \tau$.

In both subcases, the desired result follows by T-ENOUGH?:

$$\text{(T-ENOUGH?)}\ \frac{\begin{array}{c} \mathtt{Sig}_1 = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c_1;_W c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit}\ \mathit{then}\ \mathtt{Sig}_1\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}_2} c_1;_W c_2 : \tau \\ \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''; \Sigma'' \end{array}}{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''; \Sigma''}$$

Observe that the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1;_{\gamma\mid\mathsf{V}} c_2$ follows by definition 26 since $\gamma \subseteq \gamma$ and $dom(\mathsf{V}) \subseteq dom(\mathsf{V})$. The details $\Omega \preccurlyeq \Omega$ and $\Sigma \preccurlyeq \Sigma$ follow by definition, vacuously.

Case [D-SEQ-STEP]**.**

$$\text{(D-SEQ-STEP)}\ \frac{n > 0 \qquad \gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'_1}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1;_W c_2 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'_1;_W c_2}$$

The premises yield

(a) $n > 0$

(b) $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1 \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'_1$

By assumption, note that

- $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$
- $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega'; \Sigma'$
- $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1;_W c_2$ is well-formed.

Put $W = \gamma_0 \mid \mathsf{V}_0$. Then by definition 26, we have $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$. Observe that $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1$ is well-formed, which vacuously follows by definition 26 because c_1 does not have the form $c';_{W'} c''$ and we always run the command c_1 before c_2.

By inversion of $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega'; \Sigma'$ via T-Enough? rule, we have

$$(\dagger_1) \quad \mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega'; \Sigma'$$

and

$$(\dagger_2) \quad \mathtt{Md} \mid b = 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}''} c_1;_W c_2 : \tau$$

where $\mathtt{Sig}'' = \mathit{if}\ \mathtt{Md} = \mathsf{jit}, \mathit{then}\ \mathtt{Sig}', \mathit{else}\ \mathtt{Sig}$ and $\mathtt{Sig}'' = \{\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash c_1;_W c_2 : \tau\}$.

By inversion of $(\dagger_1)$ via T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma' \\ \Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2 \qquad \Sigma'' = \mathtt{trim}(\Sigma', \mathsf{V}_0, \gamma_0) \cup \downarrow\Omega^1_{\mathsf{ck}} \\ \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''' \end{array}}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1;_W c_2 : \tau \dashv \Omega''; \Sigma'''} \text{(T-SEQ-D)}$$

we learn that

(1) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} c_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma'$

(2) $\Sigma'' = \mathsf{trim}(\Sigma', \mathsf{V}_0, \gamma_0)$

(3) $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega'; \Sigma'' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma'''$

(4) $\Omega' = \Omega^1_{\mathsf{CK}}, \Omega^2$

By the inductive hypothesis applied to (1), (b), the well-formedness of $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid c_1$, $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS^0_t$, and $n \geq 0$, we get $\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma_0 \mid pcS_t \vdash_{\mathtt{Sig}} c'_1 : \mathsf{C}_{\mathtt{unit}} \dashv \Omega'; \Sigma'$, where

(i) $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o : \Omega_0 \mid \Sigma_0 \mid pcS^0_t$,

(ii) $n' \geq 0$,

(iii) $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS'_o \mid c'_1$ is well-formed,

(iv) $\Omega \preccurlyeq \Omega_0$ and $\Sigma \preccurlyeq \Sigma_0$

Since c_1 is always executed before c_2 until skip is reached, any conditional started during the execution of c_1 would also conclude during its execution. Therefore, $pcS_t^0 = pcS_t$.

We now need to show that $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$. Observe that $c_1 \neq c';_W c''$ because $c_1;_W c_2$ and we always run c_1 first. By lemma 38 and $c_1 \neq c';_W c''$, it follows that $dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$ and $\gamma \subseteq \gamma'$. Then it follows by $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V})$ and $\gamma_0 \subseteq \gamma$ (as shown above), that $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma \subseteq \gamma'$.

Using $W = \gamma_0 \mid \mathsf{V}_0$, (2), (3), and $\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathsf{unit}} \dashv \Omega'; \Sigma'$, we can apply T-SEQ-D

$$\frac{\begin{array}{c} W = \gamma_0 \mid \mathsf{V}_0 \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash_{\mathtt{Sig}} c_1' : \mathsf{C}_{\mathsf{unit}} \dashv \Omega'; \Sigma' \\ \Sigma'' = \mathsf{trim}(\Sigma', \mathsf{V}_0, \gamma_0) \qquad \mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega'; \Sigma'' \mid pcS_t \vdash_{\mathtt{Sig}} c_2 : \tau \dashv \Omega''; \Sigma''' \end{array}}{\mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash_{\mathtt{Sig}} c_1';_W c_2 : \tau \dashv \Omega''; \Sigma'''} \text{ (T-SEQ-D)}$$

We now need to show that $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash_{\mathtt{Sig}} c_1';_W c_2 : \tau$. The proof proceeds in two subcases based on Md:

Case $\mathtt{Md} = \mathsf{jit}$**.** Let $\mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau\}$. It follows by lemma 30 that $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau$.

Case $\mathtt{Md} = \mathsf{alD}(c_0)$**.** By $(\dagger_2)$ via lemma 31, we can see that $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau$.

In both cases, $\mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau$.

The desired result follows by T-ENOUGH?:

$$\frac{\begin{array}{c} \mathtt{Sig}_1 = \{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau\} \\ \mathtt{Sig}_2 = \mathit{if}\ \mathtt{Md} = \mathsf{jit},\ \mathit{then}\ \mathtt{Sig}_1,\ \mathit{else}\ \mathtt{Sig} \\ \mathtt{Md} \mid\ b = 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau \\ \mathtt{Md} \mid\ b > 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau \dashv \Omega''; \Sigma''' \end{array}}{\mathtt{Md} \mid\ b \geq 0 : \mathtt{nat} \mid\ \Omega_0; \Sigma_0 \mid pcS_t \vdash c_1';_W c_2 : \tau \dashv \Omega''; \Sigma'''} \text{ (T-ENOUGH?)}$$

Observe that by definition 26 applied to $dom(\mathsf{V}_0) \subseteq dom(\mathsf{V}')$ and $\gamma_0 \subseteq \gamma'$ (as shown above), $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O}' \mid \mathsf{N}' \mid \mathsf{V}' \mid pcS_o' \mid c_1';_W c_2$ is well-formed, as desired.

Case 11 [D-SEQ-V]**.**

$$\frac{n = n' + 1 \qquad W = \gamma' \mid \mathsf{V}' \qquad \mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')}{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ \mathbf{skip};_W c_2 \ \rightarrow\ \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}'' \mid pcS_o \mid c_2} \text{ (D-SEQ-V)}$$

Observe that $n = n' + 1$ (from the premise of D-SEQ-V) implies $n > 0$, and hence $b > 0$.

By assumption, we have

$$\mathtt{Md} \mid\ b\ > 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \mid pcS_t \vdash \mathbf{skip};_W c_2\ : \tau \dashv \Omega'; \Sigma'$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$.

By inversion on T-SEQ-D, we have

(1) $\mathtt{Md} \mid\ b\ \geq 0 : \mathtt{nat}\ \mid\ \Omega; \Sigma \mid pcS_t \vdash \mathbf{skip} : \mathtt{C}_{\mathtt{unit}} \dashv \Omega_0; \Sigma_0$

(2) $\mathtt{Md} \mid\ b\ \geq\ 0 : \mathtt{nat}\ \mid\ \Omega_0; \Sigma'_0 \mid pcS_t \vdash c_2 : \tau \dashv \Omega'; \Sigma'$

(3) $W = \gamma' \mid \mathsf{V}'$ (from the premise of D-SEQ-V)

(4) $\Sigma'_0 = \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma') \cup \downarrow\Omega^1_{\mathsf{ck}}$

(5) $\Omega_0 = \Omega^1_{\mathsf{CK}}, \Omega^2$

We now need to show that $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' \mid pcS_o : \Omega_0 \mid \Sigma'_0 \mid pcS_t$.

Since $\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ \mathbf{skip};_W c_2$ is well-formed (by assumption), it follows by definition 26 that $\gamma' \subseteq \gamma$.

By inversion of (1) on T-C-SHIFT and then on T-SKIP, we know that $\Omega = \Omega_0$ and $\Sigma = \Sigma_0$. Hence, it follows by assumption that $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega_0 \mid \Sigma_0 \mid pcS_t$.

By inversion of NV-LOC-AID-2, we have

$$\vdash^{\mathtt{Md}}_{\gamma'''} \mathsf{N}^2 \mid \mathsf{V} \mid pcS_o : \Omega_0 \mid \Sigma_0 \mid pcS_t\ (*)$$

where $\mathsf{N} = \mathsf{N}^1_{\mathsf{CK}}, \mathsf{N}^2$ and $\gamma''' \subseteq \gamma'$ such that $\mathsf{range}(\gamma''') = \mathsf{dom}(\mathsf{N}^2, \mathsf{V})$.

Therefore, by lemma 37 applied to (*), the premise $\mathsf{V}'' = \mathsf{V} \restriction dom(\mathsf{V}')$, (4), and $\gamma''' \subseteq \gamma$, we obtain that $\vdash^{\mathtt{Md}}_{\gamma'''} \mathsf{N}^2 \mid \mathsf{V}'' \mid pcS_o : \Omega \mid \mathsf{trim}(\Sigma_0, \mathsf{V}', \gamma') \mid pcS_t$. Therefore, it follows by NV-LOC-AID-2, that $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' \mid pcS_o : \Omega \mid \Sigma'_0 \mid pcS_t$, as desired.

Observe that (2) yields the desired result where $\vdash^{\mathtt{Md}}_{\gamma'} \mathsf{N} \mid \mathsf{V}'' \mid pcS_o : \Omega \mid \Sigma'_0 \mid pcS_t$. Observe that the well-formedness of $\gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}'' \mid pc \mid c_2$ follows by definition 26, vacuously because c_2 is not of the form $c';_W c''$. We now need to show that $\Omega \preccurlyeq \Omega_0$ and $\Sigma \preccurlyeq \Sigma'_0$. Since $\Omega = \Omega_0$, $\Omega \preccurlyeq \Omega_0$ holds vacuously. Note that $\Sigma \preccurlyeq \Sigma'_0$ follows by definition and $\Sigma = \Sigma_0$.

Case D-INPUT**.**

$$\text{D-INPUT}\quad \frac{pcS_o = pc_o \rhd pcS'_o \qquad \gamma' = \gamma, [x \mapsto \ell] \qquad \ell\, fresh \qquad n = n' + 1 \qquad \mathsf{I} = \mathsf{in}(v)::\mathsf{I}'}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS_o \mid\ \mathsf{let}\, x\ =\ \mathsf{IN}()\, \mathsf{in}\, c \\ \rightarrow \gamma' \mid \mathtt{Md} \mid n' \mid \mathsf{I}' \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V}, (\ell @ \langle \mathsf{Id}, \mathtt{Tnt} \rangle \sqcup pc_o \hookrightarrow v) \mid pcS_o \mid\ c\end{array}}$$

Observe that $n = n' + 1$ (from the premise of D-INPUT) implies $n > 0$, and hence $b > 0$.

By assumption, we have

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash \mathsf{let}\, x\ =\ \mathsf{IN}()\, \mathsf{in}\, c : \tau \dashv \Omega'; \Sigma'$$

where $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t\ (*)$.

By inversion on the rule T-Let-In:

$$\textsc{T-Let-in}\ \frac{pcS_t = pc_t \rhd pcS_t' \qquad \mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A@\langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc_t \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'}{\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{let}\, x = \mathsf{IN}(\,)\, \mathsf{in}\, c : \tau \dashv \Omega'; \Sigma'}$$

we learn that:

$$\mathtt{Md} \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma, x{:}{\downarrow}{\uparrow}A@\langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc_t \mid pcS_t \vdash_{\mathtt{Sig}} c : \tau \dashv \Omega'; \Sigma'$$

By the well-formedness definition, we learn that $pc_t = pc_o$. Therefore, by V-Loc applied to $(*)$ and $\gamma' = \gamma, [x \mapsto \ell]$, we establish $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N} \mid \mathsf{V}, (\ell@\langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc_o \hookrightarrow v) \mid pcS_o : \Omega \mid \Sigma, (x{:}{\downarrow}{\uparrow}A@\langle \mathsf{Id}, \mathtt{Tnt}\rangle \sqcup pc_t) \mid pcS_t$, as desired.

Case D-Output**.**

$$\textsc{D-Output}\ \frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q' \qquad n = n' + 1}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e) \\ \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e'@q')\end{array}}$$

Observe that $n = n' + 1$ (from the premise of D-Output) implies $n > 0$, and hence $b > 0$. Additionally, we know that

$$\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@\langle \mathsf{Id}, \mathtt{Nt}\rangle \rightarrow \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q'$$

By assumption, we know that

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{output}(ID, e) : \tau \dashv \Omega'; \Sigma$$

There are three cases to consider:

Subcase $\mathtt{Md} = \mathsf{aID}(c_0, \mathcal{O})$ *and* $ID \in \mathcal{O}$**.** By inversion on T-output-id, we have

- $\mathsf{aID}(c_0, \mathcal{O}) \mid b \geq 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e : A@\langle qID_v, qIO_v\rangle \dashv \Omega'$
- $qID_v \sqsubseteq_{id} \mathsf{Id}$

By application of Theorem 17, we know that $\mathsf{aID}(c_0, \mathcal{O}) \mid b \geq 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS \vdash_{\mathtt{RD;Sig}} e' : A@\langle qID_v, qIO_v\rangle \dashv \Omega'; \Sigma$ where $\Omega \preccurlyeq_{\mathtt{RD}} \Omega_0$ and $\vdash^{\mathtt{Md}}_{\gamma} \mathsf{N}' \mid \mathsf{V} \mid pcS_o \mid \Omega_0 \mid \Sigma \mid pcS_t$.

Therefore, the desired result holds by application of T-output-id:

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega_0; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{output}(ID, e') : \tau \dashv \Omega'; \Sigma$$

Subcase $\mathtt{Md} = \mathsf{aID}(c_0, \mathcal{O})$ *and* $ID \notin \mathcal{O}$**.** The reasoning for this case is similar, but instead relies on T-OUTPUT-ID.

Subcase $\mathtt{Md} = \mathsf{jit}$**.** The reasoning for this case is similar, but instead relies on T-OUTPUT-JIT.

Case D-OUTPUT-STEP**.**

$$\text{D-OUTPUT-STEP}\quad \frac{\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ e@q \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N}' \mid \mathsf{V} \mid pcS \mid\ e'@q' \qquad n = n' + 1}{\begin{array}{c}\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e@q) \\ \to \gamma \mid \mathtt{Md} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, e'@q')\end{array}}$$

This case is similar to the previous one for D-OUTPUT.

Case D-OUTPUT-NID-AID**.**

$$\text{D-OUTPUT-NID-AID}\quad \frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ v@q) \qquad ID \notin \mathcal{O} \qquad n = n' + 1}{\begin{array}{c}\gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, v@q) \\ \xrightarrow{o(ID,v)} \gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}\end{array}}$$

Observe that $n = n' + 1$ (from the premise of D-OUTPUT-NID-AID) implies $n > 0$, and hence $b > 0$.

By assumption, we know that

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{output}(ID, v) : \tau \dashv \Omega'; \Sigma\ (*)$$

where $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega \mid \Sigma \mid pcS_t$.

By Lemma 40 applied to (*) and $\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ v@q)$, we learn that $\Omega' = \Omega$. Thus, $\vdash_{\gamma}^{\mathtt{Md}} \mathsf{N} \mid \mathsf{V} \mid pcS_o : \Omega' \mid \Sigma \mid pcS_t$.

Therefore, it follows by T-SKIP and T-C-SHIFT that

$$\mathtt{Md} \mid b > 0 : \mathtt{nat} \mid \Omega'; \Sigma \mid pcS_t \vdash_{\mathtt{Sig}} \mathsf{output}(ID, v) : \tau \dashv \Omega'; \Sigma$$

as desired.

Case D-BUFFER-ID-AID**.**

$$\text{D-BUFFER-ID-AID}\quad \frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ v@q) \qquad ID \in \mathcal{O} \qquad n = n' + 1 \qquad \mathsf{O}' = \mathsf{O}::\mathsf{out}(ID, v)}{\begin{array}{c}\gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{output}(ID, v@q) \\ \to \gamma \mid \mathsf{aID}(c_0, \mathcal{O}) \mid n' \mid \mathsf{I} \mid \mathsf{O}' \mid \mathsf{N} \mid \mathsf{V} \mid pcS \mid\ \mathsf{skip}\end{array}}$$

This case is similar to the one above for D-OUTPUT-NID-AID.

Case D-OUTPUT-JIT**.**

$$\text{D-OUTPUT-JIT}\quad \frac{\mathit{Val}(\gamma \mid \mathtt{Md} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS} \mid v@q) \qquad n = n' + 1}{\gamma \mid \mathsf{jit} \mid n \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS} \mid \mathsf{output}(\mathit{ID}, v@q) \xrightarrow{o(\mathit{ID},v)} \gamma \mid \mathsf{jit} \mid n' \mid \mathsf{I} \mid \mathsf{O} \mid \mathsf{N} \mid \mathsf{V} \mid \mathit{pcS} \mid \mathsf{skip}}$$

This case is similar to the previous one.

Note that $\mathit{dom}(\mathsf{N}) = \mathit{dom}(\mathsf{N}')$ holds by observation in all cases, and by the inductive hypothesis in case 10 (D-SEQ-STEP). □

Appendix E

Glacier Appendix

E.1 Simplifications from Full System to the Main Text

To improve the main paper's readability, we abstracted away from the following details which are included in the full system definition. The excluded constructs are mostly needed for low-level proof details or provide more expressive extensions, with neither being essential to understanding the core system behavior.

The simplifications and omissions made to enhance readability are listed below:

- the tracked set of read variables Λ is removed from conditionals, which is needed by the type checking.
- *reID* and *inID*, which identify an associated input, are removed from events and interrupts
- event and interrupt identifiers (*eID* and *iID*) are conflated as *ID*
- atomic region variable sets are abbreviated as *aPol*
- *pcS* is removed from frames, and *pc* is removed from the type system. These are needed for checking conditionals that branch on idempotent or non-idempotent data.
- the tracked set of shared variables *ShAcc* in typing judgments is simplified from *vPol*, whose other variable sets are needed for other parts of the type checking. In particular, *nIds* is needed to set up atomic region typing contexts.

Additionally, variable intialization is supported by our full system but does not appear in the main text to simplify the presentation. At the beginning of a task or atomic region, a programmer could specify a set of variables *Inits* to initialize to 0, which could be used to overwrite unstable and non-idempotent effects, and reset their local qualifier to Id.

E.2 Syntax

We summarize all the syntactic constructs here. Extending sequential, intermittent program syntax, the commands additionally include critical regions for accessing shared variables. For atomic regions, we spell out all the variable policies for the region to allow type checking and operational semantics rules to use them directly. The result argument *res* at the end of atomic regions and

tasks is a fully idempotent expression representing the result to be returned by these processes. Some nestings of programs that are defined by the grammar are not supported by our system, such as nested atomic regions, nested critical sections, and atomic regions occurring in branches. At the highest level, each task's code comprises a sequence of atomic regions, commands, and critical sections. To allow critical sections to occur at the desired granularity, i.e., within the scope of atomic regions as well as surrounding atomic regions, we opt for the given syntax for commands.

We include a runtime command *initN*(*Inits*) to initialize a batch of variables in nonvolatile memory to 0 for a given process. A task can be an interrupt service routine, or an event handler. For each task, we allow flexible specification of re-execution policies *rPol*, indicating a task's behavior upon reboot: it could continue or discard. *r* is a re-execution state, such as reboot count. *d* is a program, taking *r* as argument, and returns a policy *pname*: immediate, discard, or alternative. Each event in the queue is also associated with a policy of whether it should be discarded after a power failure. For example, events posted by tasks that will be discarded should themselves be discarded. We use the mode `Md` field to indicate whether the frame is in JIT or atomic mode, if the re-execution policy evaluates to immediate. If the re-execution policy for the current frame evaluates to discard, the mode field is discard since its atomic regions are ignored at runtime. The mode field is next(*pID*) if the re-execution policy evaluates to alternative and *pID* is the process identifier for the frame of the alternative task to be executed. If a task is an alterative for another task with process identifier *pID*, its mode field will be altOf(*pID*, *rPol*), where *rPol* is the re-execution policy of this replacement task.

We also use modes to store information about variables for tasks that could be discarded if power fails. The sets *NVw* and *Vw* are the shared non-volatile variables that the potentially to be discarded tasks have written to in this power cycle. If this task succeeds, these variables will be written through to the checkpointed memory.

The shared accesses *ShAcc* are combined nonvolatile and volatile. An *Inits* set of variables is specified for each task and atomic region. It includes variables to be initialized to 0 and cleansed of any non-idempotence at the beginning of each (re-)execution of the task or atomic region. For atomic regions, variables in μ and *Inits* are both first written during an atomic region's execution, but the former are strictly idempotent whereas the latter could be, e.g., non-idempotent to be cleansed and reset to idempotent on intialization. Both μ and *Inits* are nonvolatile and are not checkpointed. We have only one *nIds* set for each task (i.e., atomic regions do not have one in this system). The set *nIds* is disjoint from the *Inits* sets of the task, and also any enclosed atomic regions.

A frame has three identifiers: the first one is the process identifier for this frame, it persists across power failures, if the frame's policy is not discard. The second one is a unique id for the irq (event). When multiple irqs (events) with the same *iID* (*eID*) appear in the system, *inID* (*reID*) distinguishes them. The third is the identifier of the irq or events that is being processed. This identifier persists throughout alternate versions of the isr (event handler) that process this irq (event).

We also assume that all variables have base type int, so Γ is defined accordingly. Lastly, we assume that *PDecl* is a global variable, to be used freely by the operational semantics and typing rules.

Values	v	::=	$n \mid \mathsf{true} \mid \mathsf{false}$
Expressions	e	::=	$x \mid v \mid e_1 \odot e_2 \mid \mathsf{uop}\, e$
Commands	c	::=	$\mathsf{skip} \mid x := e \mid \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid c_1; c_2 \mid$
		|	$\mathsf{let}\ x = e\ \mathsf{in}\ c \mid \mathsf{let}\ x = \mathsf{IN}()\ \mathsf{in}\ c$
		|	$\mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, \mathit{Inits}); c; \mathsf{end}_{\mathsf{atom}}$
		|	$\mathsf{post}(\mathit{ID}, e) \mid \mathsf{start}_{\mathsf{cr}}(\mathit{pr}, \mathit{ShAcc}); c; \mathsf{end}_{\mathsf{cr}}$
		|	$\mathsf{end}_{\mathsf{atom}} \mid \mathsf{end}_{\mathsf{cr}} \mid \mathsf{end}_{\mathsf{if}} \mid \mathsf{ret}(\mathit{res}) \mid \mathsf{reboot} \mid \mathit{initN}(\mathit{Inits})$
Atomic checkpoints	ω	::=	$\cdot \mid \omega, x$
Atomic must-first-writes	μ	::=	$\cdot \mid \mu, x$
Atomic (volatile) writes	β	::=	$\cdot \mid \beta, x$
Atomic read-only	ρ	::=	$\cdot \mid \rho, x$
Init. vars.	*Inits*	::=	$\cdot \mid \mathit{Inits}, x$
Shared loc.	*ShAcc*	::=	$\cdot \mid \mathit{ShAcc}, (x, \mathsf{rd}) \mid \mathit{ShAcc}, (x, \mathsf{wt})$
Ids	*ID*	::=	$\mathit{eID} \mid \mathit{iID}$
Unique Ids	*rID*	::=	$\mathit{reID} \mid \mathit{inID}$
Priority	*pr*	::=	n
Program Decl	*PDecl*	::=	$(\mathsf{post}(\mathit{eID}, v)_{\mathsf{entry}}, \mathit{IDecl}, \mathit{EDecl})$
ISR Decls	*IDecl*	::=	$\cdot \mid \mathit{IDecl}, \mathit{isr}$
Event Decls	*EDecl*	::=	$\cdot \mid \mathit{EDecl}, \mathit{eh}$
Task	*tsk*	::=	$\mathit{eh} \mid \mathit{isr}$
Int. Service Routine	*isr*	::=	$\mathsf{isr}(\mathit{iID}, \mathit{version}, \mathit{pr}, \mathit{vPol}, \mathit{rPol}, \lambda x.c)$
Event Handler	*eh*	::=	$\mathsf{eh}(\mathit{eID}, \mathit{version}, \mathit{pr}, \mathit{vPol}, \mathit{rPol}, \lambda x.c)$
Interrupts	*irq*	::=	$\mathsf{interrupt}(\mathit{inID}, \mathit{iID}, v)$
Events	*ev*	::=	$\mathsf{event}(\mathit{reID}, \mathit{eID}, v)$
Variable Policies	*vPol*	::=	$(\mathit{ShAcc}, \mathit{ShCP}, \mathit{nIds}, \mathit{Inits})$
Re-execution Policies	*rPol*	::=	$\mathsf{discard} \mid \mathsf{continue}(r, \lambda x.d)$
Policy Name	*pname*	::=	$\mathsf{discard} \mid \mathsf{immediate} \mid \mathsf{alternative}(\mathit{ID}, \mathit{version})$
Atomic reboot count	*rb*	::=	n
Event Queue	Q	::=	$\cdot \mid Q :: \mathit{ev} \mid Q :: \mathit{irq}$
Mode	`Md`	::=	$\mathsf{jit} \mid \mathsf{atom}(\mathsf{aID}, \mathit{rb}, \omega, \mu, \beta, \rho, \mathit{Inits}, \mathit{NVw}, \mathit{Vw})$
			$\mid \mathsf{discard}(\mathit{NVw}, \mathit{Vw}) \mid \mathsf{next}(\mathit{pID}, \mathit{NVw}, \mathit{Vw}) \mid \mathsf{altOf}(\mathit{pID}, \mathit{rPol})$
Static Simple Mode	`sMd`	::=	$\mathsf{jit} \mid \mathsf{atom} \mid \mathsf{discard}$
Priority Stack	$\overrightarrow{pr}$	::=	$\cdot \mid \mathit{pr} \rhd \overrightarrow{pr}$
Frame	F	::=	$(\mathit{pID}, \mathit{rID}, \mathit{ID}, \mathit{version}, \overrightarrow{pr}, \mathit{pcS}, \mathit{vPol}, \upsilon, E, \mathtt{Md}, c)$
Runtime Config.	Σ	::=	$(I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q) \mid (I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, \mathsf{reboot}, Q)$
Stack	S	::=	$\cdot \mid F \rhd S$
Execution ctx	E	::=	$\cdot \mid ([\,]; c) \rhd E \mid \mathsf{end}_{\mathsf{if}} \rhd E \mid \mathsf{end}_{\mathsf{cr}} \rhd E$
Interrupt inputs	*IRQ*	::=	$\cdot \mid \mathit{IRQ} :: \mathit{irq}$
Sensor inputs	I	::=	$\cdot \mid I :: \mathsf{in}(\mathit{inID}, v)$
access qualifier	*qAcc*	::=	`RD` | `WT` | `CK`
id qualifier	q	::=	$\mathsf{Id}(\mathit{qAcc}) \mid \mathsf{Id} \mid \mathsf{Nid}$
type qualifier	q	::=	$\langle \mathit{qID}, \mathit{qID}, \mathit{qID} \rangle$
Typing context	Γ	::=	$\cdot \mid \Gamma, x : \mathsf{int}@q$

E.3 Continuously-Powered Operational Semantics

In this section, we give the rules for executing concurrent programs continuously (without power failures).

E.3.1 Expression Evaluation Rules

The rules for evaluating expressions use a big-step operational semantics to evaluate an expression to a value according to the current nonvolatile and volatile memories (N and V) and local memory state (υ).

$$\frac{v \in \{n, \mathsf{tt}, \mathsf{ff}\}}{\mathsf{N}, \mathsf{V}, \upsilon \vdash v \Downarrow v}\ \textsc{Value} \qquad \frac{\mathsf{N} = x \mapsto v, \mathsf{N}'}{\mathsf{N}, \mathsf{V}, \upsilon \vdash x \Downarrow v}\ \textsc{Read-N} \qquad \frac{\mathsf{V} = x \mapsto v, \mathsf{V}'}{\mathsf{N}, \mathsf{V}, \upsilon \vdash x \Downarrow v}\ \textsc{Read-V}$$

$$\frac{\upsilon = x \mapsto v, \upsilon'}{\mathsf{N}, \mathsf{V}, \upsilon \vdash x \Downarrow v}\ \textsc{Read-}\upsilon \qquad \frac{\mathsf{N}, \mathsf{V}, \upsilon \vdash e_1 \Downarrow v_1 \qquad \mathsf{N}, \mathsf{V}, \upsilon \vdash e_2 \Downarrow v_2 \qquad [\![v_1 \odot v_2]\!] = v}{\mathsf{N}, \mathsf{V}, \upsilon \vdash e_1 \odot e_2 \Downarrow v}\ \textsc{Binary}$$

$$\frac{\mathsf{N}, \mathsf{V}, \upsilon \vdash e \Downarrow v \qquad [\![\mathsf{uop}\, e]\!] = v}{\mathsf{N}, \mathsf{V}, \upsilon \vdash \mathsf{uop}\, e \Downarrow v}\ \textsc{Unary}$$

E.3.2 Event and Interrupt Handling

$$\frac{\begin{array}{c} IRQ = \mathsf{interrupt}(inID, iID, v) :: IRQ' \\ F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \\ pr > top(\overrightarrow{pr}_c) \qquad \mathsf{isr}(iID, version, pr, vPol, rPol, \lambda x.c_i) \in PDecl \qquad fresh(pID) \\ F_n = (pID, inID, iID, version, pr, vPol, x \mapsto v, \cdot, \mathsf{jit}, initN(vPol.Inits); c_i; \mathsf{ret}(res)) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{trigger}(inID,pID)} I, IRQ', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q} \textsc{TriggerInt-C}$$

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, e)) \\ \mathsf{eh}(eID, version, pr, vPol, rPol, \lambda x.c_e) \in PDecl \\ pr > top(\overrightarrow{pr}_c) \qquad fresh(pID) \qquad fresh(reID) \qquad \mathsf{N}_c, \mathsf{V}_c, \upsilon_c \vdash e \Downarrow v \\ F_n = (pID, reID, eID, version, pr, vPol, x \mapsto v, \cdot, \mathsf{jit}, initN(vPol.Inits); c_e; \mathsf{ret}(res)) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{trigger}(reID,pID)} I, IRQ, \mathsf{N}, \mathsf{V}, F' \rhd S, F_n, Q} \textsc{PostEvent-C}$$

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \\ \mathsf{isr}(iID, version, pr, vPol, rPol, \lambda x.c_i) \in PDecl \\ IRQ = irq :: IRQ' \qquad irq = \mathsf{interrupt}(inID, iID, v) \qquad pr \leq top(\overrightarrow{pr}_c) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{delay}(inID)} I, IRQ', \mathsf{N}, \mathsf{V}, S, F, enqueue(Q, irq)} \textsc{DelayInt-C}$$

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, e)) \\ \mathsf{eh}(eID, version, pr, vPol, rPol, \lambda x.c_e) \in PDecl \qquad pr \leq top(\overrightarrow{pr}_c) \qquad fresh(reID) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \qquad \mathsf{N}_c, \mathsf{V}_c, \upsilon_c \vdash e \Downarrow v \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{delay}(reID)} I, IRQ, \mathsf{N}, \mathsf{V}, S, F', enqueue(Q, \mathsf{event}(reID, eID, v))} \textsc{DelayEvent-C}$$

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \\ \mathsf{isr}(iID, version, pr, vPol, rPol, \lambda x.c_i) \in PDecl \\ appPol(rPol) \in \{\mathsf{discard}, \mathsf{alternative}\} \qquad IRQ = irq :: IRQ' \qquad irq = \mathsf{interrupt}(inID, iID, v) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{delay}(inID)} I, IRQ', \mathsf{N}, \mathsf{V}, S, F, Q} \textsc{DropInt-C}$$

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, e)) \\ \mathsf{isr}(eID, version, pr, vPol, rPol, \lambda x.c_e) \in PDecl \qquad appPol(rPol) \in \{\mathsf{discard}, \mathsf{alternative}\} \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{delay}(reID)} I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{DropEvent-C}$$

The rules TriggerInt-C and PostEvent-C run an interrupt or event (respectively) if that task's priority *pr* is higher than that of the currently-running task $top(\overrightarrow{pr}_c)$. The rule TriggerInt-C triggers an ISR that fires non-deterministically. The first premise picks the next ISR with identifier *iID* that is declared in *PDecl*. The corresponding frame (with re-execution state instantiated) is loaded into the evaluation context and the frame with the interrupted code and its state is pushed to the stack. *v* is an input value on which that interrupt will run. *pID* is a fresh, unique identifier that is generated for each process. This distinguishes events/interrupts that have the same id in *PDecl*. In our system, when a task is triggered, a fresh *pID* consists of the task *ID* and *version*,

and a unique bit. Similarly, PostEvent-C pushes a modified version of F, with the command skip, to the stack to process the explicit event post.

The rules DelayInt-C and DelayEvent-C respectively delay an interrupt or event if that task's priority is not high enough to preempt the currently-running task ($pr \leq top(\overrightarrow{pr}_c)$). The DelayInt-C rule continues executing the same frame and enqueues the task in Q along with its input from IRQ. The DelayEvent-C continues executing the frame with command skip, pushing the delayed task to the queue with the specified input from the post v.

We define the rules DropEvent-C and DropInt-C for correctness purposes; when building the simulation for a failed discard task, the continuously-powered execution must drop the corresponding task.

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, \cdot, \texttt{Md}_c, \mathsf{ret}(res)) \\ S = F \rhd S' \qquad Q = \mathsf{event}(reID, eID, v) :: Q' \\ pr_e > prOf(F) \qquad \mathsf{eh}(eID, version, pr_e, vPol_e, rPol, \lambda x.c_e) \in PDecl \\ fresh(pID_e) \qquad F_n = (pID_e, reID, eID, version, pr_e, vPol_e, x \mapsto v, \cdot, \mathsf{jit}, c_e; \mathsf{ret}(res)) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \qquad o_1 = \mathsf{complete}(pID_c, v_r) \qquad o_2 = \mathsf{trigger}(reID, pID_e) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F_c, Q \xrightarrow{o_1, o_2} I, IRQ, \mathsf{N}, \mathsf{V}, S, F_n, Q'} \textsc{Ret-QEv-C}$$

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, \cdot, \texttt{Md}_c, \mathsf{ret}(res)) \\ S = F \rhd S' \qquad Q = \mathsf{interrupt}(inID, iID, v) :: Q' \\ pr_i > prOf(F) \qquad \mathsf{isr}(iID, version, pr_i, vPol, rPol, \lambda x.c_i) \in PDecl \\ fresh(pID_i) \qquad F_n = (pID_i, inID, iID, version, pr_i, vPol, x \mapsto v, \cdot, \mathsf{jit}, c_i; \mathsf{ret}(res)) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \qquad o_1 = \mathsf{complete}(pID_c, v_r) \qquad o_2 = \mathsf{trigger}(inID, pID_i) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F_c, Q \xrightarrow{o_1, o_2} I, IRQ, \mathsf{N}, \mathsf{V}, S, F_n, Q'} \textsc{Ret-QInt-C}$$

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, \cdot, \texttt{Md}_c, \mathsf{ret}(res)) \\ S = F \rhd S' \qquad \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \qquad o = \mathsf{complete}(pID_c, v_r) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F_c, Q \xrightarrow{o} I, IRQ, \mathsf{N}, \mathsf{V}, S', F, Q} \textsc{Ret-S-C}$$

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, vPol_c, \upsilon_c, \cdot, \texttt{Md}_c, \mathsf{ret}(res)) \\ F_h = (\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{skip}) \qquad \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \qquad o = \mathsf{complete}(pID_c, v_r) \end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, \cdot, F_c, \cdot \xRightarrow{o} I, IRQ, \mathsf{N}, \mathsf{V}, \cdot, F_h, \cdot} \textsc{Ret-Halt-C}$$

When a task finishes, Ret-QEv-C runs the next event in the queue, but only if the next event's priority is higher than the one on top of the stack. Ret-QInt-C does the same for interrupts. Ret-S-C defaults to running the next task on the stack.

E.3.3 Sequencing

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, c_1; c_2) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, ([\,]; c_2) \rhd E, \texttt{Md}, c_1)\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{Seq1-C}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, ([\,]; c) \rhd E, \texttt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, c)\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{Seq2-C}$$

The rules Seq1 and Seq2 are similar to the standard sequencing rules but for execution contexts. Seq1 evaluates a sequence $c_1; c_2$ by pushing c_2 to the execution context and continuing to evaluate c_1. When c_1 is finished executed (detected by skip), Seq2 pops c_2 (here, c) from the execution context and evaluates it.

E.3.4 Let bindings

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, \mathsf{let}\ x = e\ \mathsf{in}\ c) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, (\upsilon, x \mapsto v), E, \texttt{Md}, c) \qquad \mathsf{N}, \mathsf{V}, \upsilon \vdash e \Downarrow v \qquad x\ fresh\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{Let-C}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, \mathsf{let}\ x = \mathsf{in}()\ \mathsf{in}\ c) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, (\upsilon, x \mapsto v), E, \texttt{Md}, c) \qquad x\ fresh\end{array}}{PDecl \vdash \mathsf{in}(inID, v) :: I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \xrightarrow{\mathsf{in}(inID)} I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{Let-In-C}$$

The rule Let-C-In reads input value from the sensor input.

E.3.5 Critical section

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}_c, vPol, \upsilon, E, \texttt{Md}, \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}}) \\ F' = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} \rhd E, \texttt{Md}, c)\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{StartCrit-C}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} \rhd E, \texttt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}_c, vPol, \upsilon, E, \texttt{Md}, \mathsf{skip})\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{EndCrit-C}$$

The rule StartCrit evaluates a critical section. Critical sections are assigned a priority prior to runtime that is the ceiling of all task priorities that have common shared accesses, which is checked by our typing rules. Note that tasks that do not have shared accesses could still interrupt the critical section if the priority of that task is high enough. In StartCrit-C, the priority of the critical section pr is pushed to the priorities stack (so that no other process can interrupt it) and $\mathsf{end}_{\mathsf{cr}}$ to E. The rule EndCrit-C pops the priority and $\mathsf{end}_{\mathsf{cr}}$ at the end of an atomic region.

E.3.6 Branching

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow \mathsf{tt} \qquad \mathsf{task}(ID, v, pr, vPol, rPol, \lambda x.c) \in PDecl \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, c_1)\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{If-true-C}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ pcS = pc_0 \rhd pcS_0 \qquad \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow \mathsf{ff} \qquad \mathsf{task}(ID, v, pr, vPol, rPol, \lambda x.c) \in PDecl \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, c_2)\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{If-false-C}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip})\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{If-end-C}$$

The rules If-true-C, If-false-C, and If-end-C execute branches, pushing $\mathsf{end}_{\mathsf{if}}$ to E and running either c_1 or c_2, as appropriate.

E.3.7 Assignments

$$\frac{\begin{array}{c}F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr_c}, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr_c}, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow v \qquad x \in \mathsf{dom}(\mathsf{V})\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}[x \mapsto v], S, F', Q}\ \textsc{V-Assign-C}$$

$$\frac{\begin{array}{c}F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr_c}, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr_c}, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow v \qquad x \in \mathsf{dom}(\mathsf{N})\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}[x \mapsto v], \mathsf{V}, S, F', Q}\ \textsc{N-Assign-C}$$

The rules N-Assign-C and V-Assign-C evaluate assignments $x := e$, where x is a variable in N and V respectively.

E.3.8 Atomic Regions

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{atom}}(\cdots); c; \mathsf{end}_{\mathsf{atom}}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, c)\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{StartAtom-C}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{end}_{\mathsf{atom}}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip})\end{array}}{PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{EndAtom-C}$$

The rules StartAtom-C and EndAtom-C execute atomic regions. Since there are no power failues in the continuously-powered semantics, these rules simply evaluate the inner command c and do nothing else.

E.4 Operational Semantics for Intermittent Execution

In this section, we give the operational semantic rules for executing programs intermittently and concurrently.

E.4.1 Auxiliary Definitions

empty JIT frame	$F_J(pID)$	::=	(pID, jit)
recovery context	K	::=	$(\mathsf{N}_k, \mathsf{V}_k, S_k)$
process instance ID	*ipID*	::=	$pID \mid (pID, \mathsf{aID}, rb)$
aug. nonvolatile memory	N	::=	$\cdot \mid \mathsf{N}, x \mapsto (v, ipID)$
aug. volatile memory	V	::=	$\cdot \mid \mathsf{V}, x \mapsto (v, ipID)$
pc stack	*pcS*	::=	$\cdot \mid pc \rhd pcS$
id level	*pc*	::=	$\mathsf{Id} \mid \mathsf{Nid}$

We use a recovery context K, containing the memories and stack that should be reverted by the intermittent execution upon reboot.

We extend the stack frame with an empty JIT placeholder frame $F_J(pID)$, indexed by the process *pID*. The idea is that when a low-power signal is received in any execution mode, the system will then checkpoint the stack frames of all unfinished tasks that are in JIT mode. To keep the order of the stack, we will insert into the checkpointed stack a placeholder for a task in JIT mode the moment it enters JIT mode. The frame will be updated either when power is running low, or when the task enters atomic mode.

We write *ipID* to denote the process instance identifier, used to document the last task that writes to a memory location. Different from *pID*, when the process *pID* executes an atomic region aID after *rb* previously failed attempts, the instance identifier will be the tuple (pID, aID, rb). This will be used in our correctness proofs to identify writes from (re)-executions. We augment nonvolatile and volatile memories with an *ipID* that represents the last writing process for a given variable.

To track the idempotence level of a command (e.g., a conditional branching on an Nid expression), we include a *pc* and pc stack *pcS*.

Updating jit placeholders. The helper function *updJitS* recursively updates all jit placeholder frames in a checkpointed stack S_k with their live ones from S. If a frame in S does not have the mode jit, in the context of power failure (rule to be shown later), it means that it is a failed discard or atomic region process. Therefore, the definition of *updJitS* is written to forget such frames by not including them in the checkpointed stack.

$\boxed{updJitS(S_k, S)}$

$$\frac{}{updJitS(\cdot, \cdot) = \cdot} \qquad \frac{modeOf(F) \neq \mathsf{jit}}{updJitS(F \rhd S_k, S) = F \rhd updJitS(S_k, S)}$$

$$\frac{modeOf(F) \neq \mathsf{jit}}{updJitS(S_k, F \rhd S) = updJitS(S_k, S)}$$

$$\frac{S'_k = updJitS(S_k, S) \qquad modeOf(F) = \mathsf{jit} \qquad pidOf(F) = pidOf(F_k)}{updJitS(F_k \rhd S_k, F \rhd S) = F \rhd S'_k}$$

iPidOf(F)**.** We define $ipidOf(F)$ as the *ipID* of frame F, unless the mode is atomic. In that case, we return the *ipID* along with aID and *rb*.

$$\frac{F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \qquad \mathtt{Md} \in \{\mathsf{jit}, \mathsf{next}(\cdots), \mathsf{altOf}(\ldots)\}}{ipidOf(F) = pID}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{ipidOf(F) = (pID, \mathsf{aID}, rb)}$$

incRb(S)**.** We define a function *incRb* to increment the *rb* of all atomic regions on the stack when there is a power failure.

$$\frac{}{incRb(\cdot) = \cdot} \qquad \frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}', c) \\ \mathtt{Md}' = \mathsf{atom}(\mathsf{aID}, rb + 1, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{incRb(F \rhd S) = F' \rhd incRb(S)}$$

$$\frac{F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \qquad \mathtt{Md} \in \{\mathsf{jit}, \mathsf{next}(\ldots), \mathsf{altOf}(\ldots)\}}{incRb(F \rhd S) = F \rhd incRb(S)}$$

Scheduling a frame.

$$\frac{\begin{array}{c} appPol(rPol) = \mathsf{immediate} \\ K = (N_k, V_k, S_k) \qquad S'_k = (pID, \mathsf{jit}) \rhd S_k \qquad K' = (N_k, V_k, S'_k) \end{array}}{schdFrame(PDecl, (pID, ID, v, rPol), K, \mathsf{N}) = (\mathsf{jit}, K')}\ \textsc{schdFrame-Immediate}$$

$$\frac{\begin{array}{c} appPol(rPol) = \mathsf{discard} \qquad vPol = (ShAcc, ShCP, nIds, Inits) \\ K = (N_k, V_k, S_k) \qquad K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}, V_k, S_k) \qquad \mathtt{Md} = \mathsf{discard}(\cdot, \cdot) \end{array}}{schdFrame(PDecl, (pID, ID, v, rPol), K, \mathsf{N}) = (\mathtt{Md}, K')}\ \textsc{schdFrame-Discard}$$

$$\frac{\begin{array}{c} appPol(rPol) = \mathsf{alternative}(ID, j) \\ \mathsf{task}(ID, version_a, pr_a, vPol_a, rPol_a, \lambda x.c_a) \in PDecl \\ fresh(pID_a) \qquad fresh(rID_a) \\ F_a = (pID_a, rID_a, ID, version_a, pr_a, vPol_a, x \mapsto v, \cdot, \mathsf{altOf}(pID, rPol_a), c_a; ret) \\ K = (N_k, V_k, S_k) \qquad S'_k = F_a \rhd S_k \qquad K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}, V_k, S'_k) \\ vPol = (ShAcc, ShCP, nIds, Inits) \qquad \mathtt{Md} = \mathsf{next}(pID_a, \cdot, \cdot) \end{array}}{schdFrame(PDecl, (pID, ID, v, rPol), K, \mathsf{N}) = (\mathtt{Md}, K')}\ \textsc{schdFrame-Alt.}$$

We define a function *schdFrame* that evaluates a frame's re-execution policy and returns a mode, an updated checkpointed context K', and updated nonvolatile memory N'. The function is triggered any time a task is about to begin executing in order to prepare for failure.

When the policy is immediate (schdFrame-Immediate), we place a jit placeholder frame on the checkpointed stack.

When the policy is discard (schdFrame-Discard), we do not update the stack in the checkpointed context, and instead update the checkpointed N_k with the most recent values in N. Our type checking rules ensure that no unstable values could be checkpointed.

When the policy is alternative (schdFrame-Alternative), we set up the alternative frame F_a in the checkpointed stack and update checkpointed N_k with the most recent values in N. We use the execution mode field to remember pID_a is the alternative.

schdFrame will never produce the mode $\mathsf{altOf}(\_, rPol)$, which will only arise when a new frame needs to be scheduled when a task completes, or at reboot time. This case is handled at Reboot-IA and Ret-S-I.

Finalizing a task. We define a helper function *finalizeK* that takes a checkpointed context and a task frame that is completed and update the checkpointed context to clean up the completed frame and to update the memory in the context, as the writes by the tasks have not been effected. We only update the nonvolatile locations that are already checkpointed. This will ensure all the values stored in the checkpointed context are idempotent.

When a frame with an alternative policy finishes, we need to remove the alternative frame from the checkpointed stack, because it is no longer needed.

$$\frac{F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{jit}, \mathsf{ret}(res)) \quad K = (N_k, V_k, F_k \rhd S_k) \quad K' = (N_k{\downarrow_{ckVarOf(S_k)}}, V_k, S_k)}{finalizeK(K, F_c, N, V) = K'}\ \textsc{fin-k-jit}$$

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{discard}(NVw, Vw), \mathsf{ret}(res)) \\ K = (N_k, V_k, S_k) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \textsc{fin-k-discard}$$

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{next}(pID_a, NVw, Vw), \mathsf{ret}(res)) \\ K = (N_k, V_k, F_a \rhd S_k) \\ fresh(rID_a) \qquad F_a = (pID_a, rID_a, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \_, \_, \mathsf{altOf}(pID_c, \_), \_) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \textsc{fin-k-next}$$

Updating memories. The following definitions are the same for volatile and nonvolatile memories. We write each rule for a fixed $M \in \{N, V\}$.

$$\frac{}{\cdot \leftsquigarrow M = M}\ \textsc{mem-to-k-empty}$$

$$\frac{M(x) = (v', ipID')}{(x \mapsto (v, ipID), M_k) \leftsquigarrow M = x \mapsto (v', ipID'), (M_k \leftsquigarrow M)}\ \textsc{mem-to-k-upd}$$

$$\frac{x \notin \mathsf{dom}(M_k)}{(x \mapsto (v, ipID), M_k) \leftsquigarrow M = x \mapsto (v, ipID), (M_k \leftsquigarrow M)}\ \textsc{mem-to-k-skip}$$

We define a function $M_1 \leftsquigarrow M_2$ to update every variable's value in M_1 with a recent one from M_2 (we write M to represent either volatile or nonvolatile memory, for the rules are the same). For example, we write through nonvolatile (or volatile) memory into checkpointed nonvolatile memory in some assignment cases, and do the reverse in reboot. In the latter case, we may write $M_k \rightsquigarrow M$, defined as follows:

$$\frac{}{M_k \rightsquigarrow \cdot = M_k}\ \textsc{k-to-mem-empty}$$

$$\frac{M_k(x) = (v', ipID')}{M_k^{ipID'} \rightsquigarrow (x \mapsto (v, ipID), M) = x \mapsto (v', ipID'), (M_k \rightsquigarrow M)}\ \textsc{k-to-mem-upd}$$

$$\frac{x \notin \mathsf{dom}(M_k)}{M_k^{ipID'} \rightsquigarrow (x \mapsto (v, ipID), M) = x \mapsto (v, ipID), (M_k \rightsquigarrow M)}\ \textsc{k-to-mem-skip}$$

The function $\rightsquigarrow$ performs the reverse, updating variables in M with their checkpointed values. This definition is used by the reboot rules to revert values from prior executions in nonvolatile memory. When doing so, the rules update the process identifier to that of the rebooted process.

Initialization. Lastly, we define a helper function to initialize variables when the command *initN*(*Inits*) runs. We denote the helper function as $initN^*(Inits, ipID, \mathsf{N})$ for intitializing the values of *Inits* in N for a given process. Even though this definition is triggered in most of the same places as *schdFrame*, we made it separate from *schdFrame* because it should not be triggered by RET-S-I which resumes a half-finished task.

$$\frac{}{initN^*(\cdot, ipID, \mathsf{N}) = \mathsf{N}}\ \textsc{Init-empty}$$

$$\frac{Inits = x, Inits' \qquad x \in dom(\mathsf{N}) \qquad \mathsf{N}' = \mathsf{N}[x \mapsto (0, ipID)]}{initN^*(Inits, ipID, \mathsf{N}) = initN^*(Inits', ipID, \mathsf{N}')}\ \textsc{Init-upd}$$

$$\frac{Inits = x, Inits' \qquad x \notin dom(\mathsf{N})}{initN^*(Inits, ipID, \cdot) = initN^*(Inits', ipID, \mathsf{N})}\ \textsc{Init-skip}$$

E.4.2 Expression Evaluation Rules

The rules for evaluating expressions use a big-step operational semantics to evaluate an expression to a value according to the current nonvolatile and volatile memories (N and V) and local memory state (υ).

$$\frac{v \in \{n, \mathsf{tt}, \mathsf{ff}\}}{\mathsf{N}, \mathsf{V}, \upsilon \vdash v \Downarrow v}\ \textsc{Value} \qquad \frac{\mathsf{N} = x \mapsto (v, ipID), \mathsf{N}'}{\mathsf{N}, \mathsf{V}, \upsilon \vdash x \Downarrow v}\ \textsc{Read-N} \qquad \frac{\mathsf{V} = x \mapsto (v, ipID), \mathsf{V}'}{\mathsf{N}, \mathsf{V}, \upsilon \vdash x \Downarrow v}\ \textsc{Read-V}$$

$$\frac{\upsilon = x \mapsto (v, ipID), \upsilon'}{\mathsf{N}, \mathsf{V}, \upsilon \vdash x \Downarrow v}\ \textsc{Read-}\upsilon$$

$$\frac{\mathsf{N}, \mathsf{V}, \upsilon \vdash e_1 \Downarrow v_1 \qquad \mathsf{N}, \mathsf{V}, \upsilon \vdash e_2 \Downarrow v_2 \qquad [\![v_1 \odot v_2]\!] = v}{\mathsf{N}, \mathsf{V}, \upsilon \vdash e_1 \odot e_2 \Downarrow v}\ \textsc{Binary}$$

$$\frac{\mathsf{N}, \mathsf{V}, \upsilon \vdash e \Downarrow v \qquad [\![\mathsf{uop}\, e]\!] = v}{\mathsf{N}, \mathsf{V}, \upsilon \vdash \mathsf{uop}\, e \Downarrow v}\ \textsc{Unary}$$

E.4.3 Assignments

In assignment, we write through to N and V and update the checkpointed context K in jit mode.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow v \qquad x \in \mathsf{dom}(\mathsf{V}) \\ \mathtt{Md}'_c = \mathtt{Md}_c[Vw \leftarrow Vw \cup \{x\}] \qquad K = (N_k, V_k, S_k) \qquad K' = (N_k, V'_k, S_k) \\ V'_k = \begin{cases} V_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ V_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}[x \mapsto (v, ipID)], S, F', Q} \text{ V-Assign-I}$$

V-Assign-I writes a value v to a x in volatile memory. It adds x to the set of written volatile variables Vw, and stores $x \mapsto (v, ipID)$ in checkpointed volatile memory because JIT re-executes from the line of failure, but the region aborts execution if the mode is anything else (in which case we do not want to remember the update to x). We use $ipID$ to identify the process that wrote to x.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_0) \in PDecl \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow v \qquad x \in \mathsf{dom}(\mathsf{N}) \qquad K = (N_k, V_k, S_k) \qquad \Psi(ID_c, version_c, \mathtt{Md}_c) = \Gamma \\ \mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \Gamma(x).1 = \mathsf{Id}(\_) \\ \mathtt{Md}_c & \text{otherwise} \end{cases} \qquad K' = (N'_k, V_k, S_k) \\ N'_k = \begin{cases} N_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ N_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], \mathsf{V}, S, F', Q} \text{ N-Assign-I}$$

N-Assign-I is similar, but assigns a value to a nonvolatile memory location. Here, we want to add the variable x to the nonvolatile writes set if it is locally $\mathsf{Id}(\_)$ (in atomic mode). This is strictly for proof purposes to get the rules to match up. We also update the checkpointed context's nonvolatile memory if we are in jit mode.

E.4.4 Triggering Interrupts and Posting Events

$$\frac{\begin{array}{c} IRQ = \mathsf{interrupt}(inID, iID, v) :: IRQ' \\ F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \\ \mathsf{isr}(iID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \qquad pr > prOf(\overrightarrow{pr}_c) \\ fresh(pID) \qquad (\mathtt{Md}_n, K') = schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N}) \\ ipID = iPidOf(pID, \mathtt{Md}_n) \\ F_n = (pID, inID, iID, 0, pr, \mathsf{Id}, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol.Inits); c_i; ret) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{trigger}(inID, pID, \mathtt{Md}_n)}{\Longrightarrow} I, IRQ', K', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q} \text{ TriggerInt-I}$$

The rule TriggerInt-I runs an interrupt isr() from *PDecl* if its priority is high enough to preempt the currently-running task ($pr > prOf(\overrightarrow{pr}_c)$). We bind the input v from *IRQ* to x to be used in evaluating $c_i; ret$. We also initialize all variables in *Inits* to 0 and evaluate the interrupt on updated memory N'. We push the preempted task frame F to the stack $F \rhd S$ to be resumed later on. The *schdFrame* computes $\mathtt{Md}_n$ for the interrupt and updates the checkpointed context.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, v)) \\ \mathsf{eh}(eID, 0, pr, vPol, rPol, \lambda x.c_e) \in PDecl \qquad pr > prOf(\overrightarrow{pr}_c) \qquad fresh(pID) \\ fresh(reID) \qquad F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \\ (\mathtt{Md}_n, K') = schdFrame(PDecl, (pID, eID, v, rPol), K, \mathsf{N}) \\ ipID = iPidOf(pID, \mathtt{Md}_n) \\ F_n = (pID, reID, eID, 0, pr, \mathsf{ld}, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol.Inits); c_e; ret) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{trigger}(reID, pID, \mathtt{Md}_n)}{\Longrightarrow} I, IRQ, K', \mathsf{N}, \mathsf{V}, F' \rhd S, F_n, Q} \text{ PostEvent-I}$$

The rule PostEvent-I is similar to post interrupts rules, except that the events are posted by an explicit command $\mathsf{post}(eID, v)$. The checking rules make sure that the mode for posting these events is jit, which is necessary for correctness. We push frame F' with command skip to the stack so that after F_n finishes, execution can resume from the next line of code in the interrupted task.

$$\frac{\begin{array}{c} \mathsf{N}' = initN^*(Inits, ipID, \mathsf{N}) \\ ipID = iPidOf(F) \qquad F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, initN(Inits)) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}[NVw \leftarrow NVw \cup Inits], \mathsf{skip}) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}', \mathsf{V}, S, F', Q} \text{ Init-N-I}$$

The rule Init-N-I initializes a batch of variables *Inits* to 0, and updates the *NVw* field of the frame mode to include these variables. The runtime command *initN* is introduced with triggering a task or beginning an atomic region to initialize the *Inits* variables of each.

E.4.5 Delaying Interrupts and Events

$$\frac{\begin{array}{c} IRQ = irq :: IRQ' \qquad irq = \mathsf{interrupt}(inID, iID, v) \\ pr \le prOf(\overrightarrow{pr}_c) \qquad \mathsf{isr}(iID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \\ F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{delay}(inID)}{\Longrightarrow} I, IRQ', K, \mathsf{N}, \mathsf{V}, S, F, enqueue(Q, irq)} \text{ DelayInt-I}$$

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, v)) \\ \mathsf{eh}(eID, 0, pr, vPol, rPol, \lambda x.c_e) \in PDecl \qquad pr \le prOf(\overrightarrow{pr}_c) \\ fresh(reID) \qquad F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \end{array}}{\begin{array}{l} \Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \\ \overset{\mathsf{delay}(reID)}{\Longrightarrow} I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', enqueue(Q, \mathsf{event}(reID, eID, v)) \end{array}} \text{ DelayEvent-I}$$

The rule DelayEvent-I and DelayInt-I enqueue events and interrupts to Q if its priority is not high enough to preempt the currently-running task. In case of a power failure, the reboot rules enforce that only tasks with policy continue will survive (i.e., discard tasks will be forgotten).

E.4.6 Return

At the point of return, we use *finalizeK* to update K with the current N and V (because the effects of this current frame should be observable by other tasks now) and to remove its frame from the checkpointed stack. We have two rules for return, Ret-S-I and Ret-S-IQ, the key difference being whether to play a task from the stack or the queue respectively. When resuming a half-finished task, we should not reinitialize the *Inits* variables, whereas when playing a task from the queue we do need to because the task is running for the first time.

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, \mathsf{ret}(res)) \\ finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K' \\ S = F \rhd S' \qquad F = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \\ Q = \mathsf{tk}(rID_n, ID_n, v) :: Q' \qquad prOf(\overrightarrow{pr}_n) \geq pr_q \\ (\mathtt{Md}'_n, K'') = \begin{cases} schdFrame(PDecl, (pID_n, ID_n, v, rPol), K', \mathsf{N}) & \text{if } \mathtt{Md}_n = \mathsf{altOf}(rPol) \\ (\mathtt{Md}_n, K') & \text{else} \end{cases} \\ c'_n = \begin{cases} initN(vPol_n.Inits); c_n; \mathsf{ret}(res') & \text{if } \mathtt{Md}_n = \mathsf{altOf}(rPol) \\ c_n & \text{else} \end{cases} \\ ipID = iPidOf(pID_n, \mathtt{Md}'_n) \qquad F' = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}'_n, c'_n) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \qquad o_1 = \mathsf{complete}(pID_c, v_r) \qquad o_2 = \mathsf{trigger}(rID_n, pID_n, \mathtt{Md}_n) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q \overset{o_1, o_2}{\Longrightarrow} I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q} \text{ Ret-S-I}$$

The rule Ret-S-I plays a half-finished task from the stack. We use *schdFrame* to prepare the checkpointed context and mode for this execution and run the frame F' with the updated mode under the updated K'' and the popped stack S'.

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, \mathsf{ret}(res)) \\ finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K' \qquad S = F \rhd S' \\ F = (pID_n, rID_n, ID_n, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \qquad Q = \mathsf{tk}(rID_q, ID_q, v) :: Q' \\ \mathsf{task}(ID_q, 0, pr_q, pcS_q, vPol_q, rPol_q, \lambda x.c_q) \in PDecl \qquad pr_q > prOf(\overrightarrow{pr}_n) \qquad fresh(pID_q) \\ fresh(rID_q) \qquad schdFrame(PDecl, (pID_q, ID_q, v, rPol_q), K', \mathsf{N}) = (\mathtt{Md}_q, K'') \\ ipID = iPidOf(pID_q, \mathtt{Md}_q) \\ F' = (pID_q, rID_q, ID_q, 0, pr_q, pcS_q, vPol_q, x \mapsto v, \cdot, \mathtt{Md}_q, initN(vPol_q.Inits); c_q; ret) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q \overset{\mathsf{complete}(pID_c, v_r)}{\overset{\mathsf{trigger}(rID_n, pID_n, \mathtt{Md}_n)}{\Longrightarrow}} I, IRQ, K'', \mathsf{N}, \mathsf{V}, S, F', Q'} \text{ Ret-Q-I}$$

When the enqueued task has higher priority than the frame on the stack, we run the next task from the queue (Ret-S-IQ). The rest of the rule is similar to Ret-S-I, except that here we initialize the variables in *Inits* and run the task on the updated memory N′.

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, \mathsf{ret}(res)) \\ finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K' \qquad F_h = (\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{skip}) \qquad \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r\end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, \cdot, F_c, \cdot \overset{\mathsf{complete}(pID_c, v_r)}{\Longrightarrow} I, IRQ, K', \mathsf{N}, \mathsf{V}, \cdot, F_h, \cdot}\ \textsc{Ret-Halt-I}$$

The rule Ret-Halt-I halts execution when there are no tasks in the stack or the task queue. If there are no more interrupts to be triggered from *IRQ*, this rule would halt the overall execution. Otherwise, this rule simply pauses the execution for when the next interrupt from *IRQ* fires.

E.4.7 Atomic Regions

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \\ \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad K' = (N_k \leftsquigarrow N{\downarrow_\omega}, V_k, F_0 \rhd S_k) \qquad ipidOf(F) = ipID \\ vPol = (ShAcc, nIds, Inits) \qquad F_0 = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, 0, \omega, \mu, \rho, Inits, \cdot, \cdot), initN(Inits); c; \mathsf{end}_{\mathsf{atom}})\end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F_0, Q}\ \textsc{StartAtom-I}$$

The rule StartAtom-I runs an atomic region in jit. $N_k \leftsquigarrow N{\downarrow_\omega}$ updates N_k with the current checkpointed values in N. When we start an atomic region, we replace the jit placeholder frame F_k with the current frame F' so that in case of a power failure, the system re-executes the region from the beginning. There is always a jit frame, even if we just finished another atomic region, because EndAtom-I will place a jit frame there. The set of nonvolatile writes is set to *Inits* because the initialization doesn't trigger the assignment rules and we need to track that they have been written (we will see later that in atomic regions, *Inits* are $\mathsf{Id}(\_)$). The volatile writes set is initialized to $\emptyset$.

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw), \mathsf{end}_{\mathsf{atom}}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad F_k = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw), c) \qquad V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \\ N'_k = N_k \leftsquigarrow N{\downarrow_{NVw}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, F_J(pID) \rhd S_k) \qquad \mathsf{N}, \mathsf{V}, \upsilon \vdash res \Downarrow v_r\end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{endAtom}(pID, \mathsf{aID}, rb, v_r)}{\Longrightarrow} I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{EndAtom-I}$$

The rule EndAtom-I closes off an atomic region, detected by the command $\mathsf{end}_{\mathsf{atom}}$. K is updated to replace the atomic region recovery frame F_k with a jit placeholder to prepare for executing the upcoming jit region, and with updated volatile and nonvolatile memories. V'_k is updated with the most recent writes in V, restricted to the ones written by the atomic region (β). N'_k is updated with the most recent nonvolatile writes *NVw*, restricted to the variables that are already checkpointed and $ckVarOf(S_k)$ which is the set of ωs in the checkpointed stack. We will

lose a frame if jit low power fails to checkpoint the state. We would have to assume this is not the case. Lastly, the rule evaluates the result argument *res* to a value v.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \\ \quad \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}}) \\ \quad \mathtt{Md} \in \{\mathsf{discard}(\dots), \mathsf{next}(pID_a, \dots)\} \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, initN(Inits); c) \\ ipidOf(F) = ipID \qquad vPol = (ShAcc, nIds, Inits) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \textsc{StartAtom-DA}$$

The rule StartAtom-DA starts atomic regions in a to-be-discarded task, meaning it will not re-execute. For this reason, we only initialize the *Inits* variables but do not need to update K. We remove the end atomic command, because we do not need it. In discard mode, we will never execute an end atomic command, meaning that the stack does not update with jit placeholder frames after running an atomic region.

E.4.8 Power Failure

Power failures are nondeterministic, meaning they could happen at any point during a program's execution.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ K = (N_k, V_k, S_k) \qquad S'_k = updJitS(S_k, F \rhd S) \qquad K' = (N_k, V_k, S'_k) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q} \textsc{PowerFail-I}$$

When power is about to fail (regardless of the mode), the rule PowerFail-I updates the checkpointed stack with the current frame F, to be resumed when power is restored. The function $updJitS(S_k, S)$ updates every jit frame in S_k with the corresponding frame in S. The memory inside of the checkpointed context does not need to be updated here as it has already been updated if variables were written (taken care of the assignment rules).

E.4.9 Reboot

When power is restored, we can re-execute the interrupted task (as in Reboot-I) or run an alternative if applicable (as in Reboot-IA.) Both rules pull out the checkpointed stack and memories from K. They clear all discard tasks from Q, meaning that, e.g. interrupts marked as discard, do not persist across power failures, and increment all the reboot counters of atomic regions on the stack by 1. The recovery context K' is updated with the incremented stack frames. We write $N_k^{ipID'} \rightsquigarrow \mathsf{N}$ and $V^{ipID'}$ as shorthand for the respective memories, with *pID* updated to the current rebooted process $ipID'$. For example, $N_k^{ipID'} \rightsquigarrow \mathsf{N}$ means the locations of N with checkpointed locations in N_k are updated with those values and a *pID* of $ipID'$. This is a subtle detail needed for our correctness proof.

$$\frac{\begin{array}{c} K = (N_k, V, S) \qquad (F \rhd S_k) = incRb(S) \\ K' = (N_k, V, F \rhd S_k) \qquad F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ polOf(F) \neq \mathsf{altOf}(\_) \qquad Q = Q_d, Q_k \qquad \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \\ \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \qquad ipidOf(F') = ipID' \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K', N_k^{ipID'} \rightsquigarrow \mathsf{N}, V^{ipID'}, S_k, F, Q_k} \text{ Reboot-I}$$

$$\frac{\begin{array}{c} K = (N_k, V, F \rhd S) \qquad S_k = incRb(S) \qquad K' = (N_k, V, S_k) \\ F = (pID_n, rID_n, ID_n, version_n, \overrightarrow{pr}_n, pcS_n, vPol_n, x \mapsto v, \cdot, \mathsf{altOf}(\_, rPol_n), c_n; \mathsf{ret}(res)) \\ Q = Q_d, Q_k \qquad \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \qquad \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \\ (\mathtt{Md}_n, K'') = schdFrame(PDecl, (pID_n, ID_n, v, rPol_n), K', \mathsf{N}) \\ \mathsf{N}' = N_k^{pID_n} \rightsquigarrow \mathsf{N} \qquad ipID = iPidOf(pID_n, \mathtt{Md}_n) \\ F' = (pID_n, rID_n, ID_n, version_n, \overrightarrow{pr}_n, pcS_n, vPol_n, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol_n.Inits); c_n; \mathsf{ret}(res)) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K'', \mathsf{N}', V^{pID_n}, S_k, F', Q_k} \text{ Reboot-IA}$$

If the mode is $\mathsf{altOf}(\_, rPol)$ (as in Reboot-IA), we appeal to the *schdFrame* and *initN* functions to prepare for running a new (alternative) task. We use the re-execution policy of the alternate task $rPol_n$ to decide how to update the stack in *schdFrame*, inserting a jit placeholder if the mode is jit, or another alternate task if the mode is next (there can be nestings of alternatives).

E.4.10 Branching

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ pcS = pc_0 \rhd pcS_0 \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow \mathsf{true} \qquad \mathsf{task}(ID, version, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \Psi(ID, version, \mathtt{sMd}) = \Gamma \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad pc = (\sqcup_{x \in \Lambda} \Gamma(x)\downarrow_{\overline{q}}) \sqcup pc_0 \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, c_1) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ If-true-I}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ pcS = pc_0 \rhd pcS_0 \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow \mathsf{false} \qquad \mathsf{task}(ID, version, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \Psi(ID, version, \mathtt{sMd}) = \Gamma \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad pc = (\sqcup_{x \in \Lambda} \Gamma(x)\downarrow_{\overline{q}}) \sqcup pc_0 \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, c_2) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ If-false-I}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ If-end-I}$$

When starting a branch (rules IF-TRUE-I and IF-FALSE-I), we raise the pc to the join of the current pc_0 and the q (triple) of all the variables the branching condition e depends on. We write it as $pc = (\sqcup_{x \in \Lambda} \Gamma(x)\downarrow_{\vec{q}}) \sqcup pc_0$. Our frames come with a stack of pc to handle nested branches. We also the instruction $\mathsf{end}_{\mathsf{if}}$ to E to pop the pc at the end of a branch (rule IF-END-I).

E.4.11 Let

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{let}\ x = e\ \mathsf{in}\ c) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon[x \mapsto v], E, \mathtt{Md}, c) \\ \mathsf{N}, \mathsf{V}, \upsilon \vdash e \Downarrow v \qquad x\ fresh \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ LET-I}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \mathtt{Md}, \mathsf{let}\ x = \mathsf{in}()\ \mathsf{in}\ c\mathsf{V}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon[x \mapsto v], E, \mathtt{Md}, c) \qquad x\ fresh \end{array}}{\Psi, PDecl \vdash \mathsf{in}(inID, v) :: I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{in}(inID)}{\Longrightarrow} I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ LET-IN-I}$$

All the variables written to by let are local volatile variables. To help with unscoping variables at the end of a let, we only add the bindings to υ. In case of a power failure, jit checkpointing would save the entire frame, including υ, to the recovery stack at the low power signal.

E.4.12 Sequencing

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c_1; c_2) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, ([\,]; c_2) \rhd E, \mathtt{Md}, c_1) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ SEQ1-I}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, ([\,]; c) \rhd E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ SEQ2-I}$$

The rules SEQ1-I and SEQ2-I execute sequences of commands $c_1; c_2$ by first running c_1 and pushing c_2 to E (SEQ1-I). When c_1 is finished running, SEQ2-I pulls c_2 from E and runs it.

E.4.13 Critical Sections

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}}) \\ F' = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} \rhd E, \mathtt{Md}, c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ STARTCRIT-I}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} \rhd E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ ENDCRIT-I}$$

Critical sections are analogous in their treatment of priorities to branches and their treatment of pc. To run a critical section (StartCrit-I) pushes $\mathsf{end}_{\mathsf{cr}}$ to E and the section priority pr to the priority stack $\overrightarrow{pr}_c$. At the end of a critical section, EndCrit-I pops both, returning to the normal task priority.

E.5 Fully-Annotated Running Example

event(ID:entry, pr:1, $vPol_e$, $rPol$:immediate,
start$_{\text{atom}}$(aID$_e$, $aPol_e$); start$_{\text{cr}}$(pr:2, $ShAcc$:{x, d, log}); d := diff(prev, x); prev := x; read(log); end$_{\text{cr}}$; end$_{\text{atom}}$)
isr(ID:ISR_IN, pr:2, $vPol_i$, $rPol$:discard, let y := in(); start$_{\text{cr}}$(pr:2, $ShAcc$:{x, d, log}); x := y; log := d; end$_{\text{cr}}$)

where $aPol_e = (\omega{:}\{prev\}, \mu{:}\{d\}, \beta{:}\cdot, \rho{:}\{x, log\})$
$vPol_e = (ShAcc{:}\{x, d, log\}, ShCP{:}\cdot, nIds{:}\cdot)$ $vPol_i = (ShAcc{:}\{x, d, log\}, ShCP{:}\{x\}, nIds{:}\{log\})$

We show the full running example execution. Changes from the previous line are highlighted, while those remaining the same are abbreviated as "…". We write i for interrupt.

I:in(11), IRQ:(i)(ISR_IN, v_i), K:·, N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en1}, Q:·

$\xrightarrow{\text{trigger}(p0)}$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($\cdot, \cdot, F_J(p0)$), N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en2}, Q:· PostEvent-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($\cdot, \cdot, F_J(p0)$), N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en4}, Q:· Seq1-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en4}, Q:· StartAtom-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en5}, Q:· Seq1-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en6}, Q:· StartCrit-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 0, prev \mapsto 5$), V:·, S:·, F_{en7}, Q:· Seq1-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 5$), V:·, S:·, F_{en8}, Q:· N-Assign-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 5$), V:·, S:·, F_{en9}, Q:· Seq2-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 2$), V:·, S:·, F_{en10}, Q:· N-Assign-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 2$), V:·, S:·, F_{en11}, Q:· EndCrit-I

$\Longrightarrow$ I:in(11), IRQ:(i)(ISR_IN, v_i), K:($prev^{ck} \mapsto 5, \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 2$), V:·, S:·, F_{en12}, Q:· Seq2-I

$\xrightarrow{\text{trigger}(p1)}$ I:in(11), IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:F_{en12}, F_{in1}, Q:· TriggerInt-I

$\Longrightarrow$ I:in(11), IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:F_{en12}, F_{in2}, Q:· Seq1-I

$\xrightarrow{\text{in}(11)}$ I:·, IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:F_{en12}, F_{in3}, Q:· Let-In-I

$\Longrightarrow$ I:·, IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:F_{en12}, F_{in4}, Q:· StartCrit-I

$\Longrightarrow$ I:·, IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:F_{en12}, F_{in5}, Q:· Seq1-I

$\Longrightarrow$ I:·, IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 11, d \mapsto 2, prev \mapsto 3$), V:·, S:F_{en12}, F_{in6}, Q:· N-Assign-I

$\Longrightarrow$ I:·, IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 11, d \mapsto 2, prev \mapsto 3$), V:·, S:·, reboot, Q:· PowerFail-I

$\xrightarrow{\text{rb}}$ I:·, IRQ:·, K:($(x^{ck} \mapsto 3, prev^{ck} \mapsto 5), \cdot, F_{\text{en4}}$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 5$), V:·, S:·, F'_{en4}, Q:· Reboot-I

⋮

$\Longrightarrow$ I:·, IRQ:·, K:($\cdot, \cdot, F_J(p0)$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:·, F'_{en12}, Q:· EndCrit-I

$\xrightarrow{\text{endAtom}(p0,aID_e,1)}$ I:·, IRQ:·, K:($\cdot, \cdot, F_J(p0)$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:·, F_{en13}, Q:· EndAtom-I

$\Longrightarrow$ I:·, IRQ:·, K:($\cdot, \cdot, F_J(p0)$), N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:·, F_{en14}, Q:· Seq2-I

$\xrightarrow{\text{complete}(p0)}$ I:·, IRQ:·, K:·, N:($x \mapsto 3, d \mapsto 2, prev \mapsto 3$), V:·, S:·, F_{fin}, Q:· Ret-Halt-I

Frames for entry

$F_{\text{en1}} = (\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \text{post}(\text{entry}, v_e))$

$F_{\text{en2}} = (pID{:}p0, ID{:}\text{entry}, \overrightarrow{pr}{:}1, vPol_e, v{:}inp \mapsto v_e, E{:}\cdot, \texttt{Md}{:}\text{jit},$
$\text{start}_{\text{atom}}(\text{aID}_e, aPol_e); (\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{d} := \text{diff}(\text{prev}, \text{x}); \text{prev} := \text{x}; \text{read}(\text{log}); \text{end}_{\text{cr}}); \text{end}_{\text{atom}}; \text{ret})$

$F_{\text{en3}} = (\ldots, E{:}[]; \text{ret}, \texttt{Md}{:}\text{jit},$
$\text{start}_{\text{atom}}(\text{aID}_e, aPol_e); (\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{d} := \text{diff}(\text{prev}, \text{x}); \text{prev} := \text{x}; \text{read}(\text{log}); \text{end}_{\text{cr}}); \text{end}_{\text{atom}})$

$F_{\text{en4}} = (\ldots, E{:}[]; \text{ret}, \texttt{Md}{:}\text{atom}(\text{aID}_e, aPol_e, NVw{:}\cdot, Vw{:}\cdot, rb{:}0),$
$(\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{d} := \text{diff}(\text{prev}, \text{x}); \text{prev} := \text{x}; ; \text{read}(\text{log}); \text{end}_{\text{cr}}); \text{end}_{\text{atom}})$

$F_{\text{en5}} = (\ldots, E{:}\text{end}_{\text{atom}} \rhd ([]; \text{ret}), \ldots,$
$(\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{d} := \text{diff}(\text{prev}, \text{x}); \text{prev} := \text{x}; \text{read}(\text{log}); \text{end}_{\text{cr}}))$

$F_{\text{en6}} = (\ldots, \overrightarrow{pr}{:}2 \rhd 1, \ldots, E{:}\text{end}_{\text{cr}} \rhd (\text{end}_{\text{atom}}; ([]; \text{ret})), \ldots,$
$\text{d} := \text{diff}(\text{prev}, \text{x}); \text{prev} := \text{x}; \text{read}(\text{log}))$

$F_{\text{en7}} = (\ldots, E{:}prev := x; read(log) \rhd (\text{end}_{\text{cr}} \rhd (\text{end}_{\text{atom}}; ([]; \text{ret}))), \ldots, \text{d} := \text{diff}(\text{prev}, \text{x}))$

$F_{\text{en8}} = (\ldots, E{:}\text{prev} := \text{x}; \text{read}(\text{log}) \rhd (\text{end}_{\text{cr}} \rhd (\text{end}_{\text{atom}}; ([]; \text{ret}))), \texttt{Md}{:}\text{atom}(\text{aID}_e, aPol_e, NVw{:}\{d\}, Vw{:}\cdot, rb{:}0), \text{skip})$

$F_{\text{en9}} = (\ldots, E{:}\text{end}_{\text{cr}} \rhd (\text{end}_{\text{atom}}; ([]; \text{ret})), \ldots, \text{prev} := \text{x}; \text{read}(\text{log}))$

$F_{\text{en10}} = (\ldots, E{:}\text{end}_{\text{cr}} \rhd (\text{end}_{\text{atom}}; ([]; \text{ret})), \texttt{Md}{:}\text{atom}(\text{aID}_e, aPol_e, NVw{:}\{d, \text{prev}\}, Vw{:}\cdot, rb{:}0), \text{skip})$

$F_{\text{en11}} = (\ldots, \overrightarrow{pr}{:}1, E{:}\text{end}_{\text{atom}}; ([]; \text{ret}), \ldots, \text{skip})$

$F_{\text{en12}} = (\ldots, E{:}([]; \text{ret}), \ldots, \text{end}_{\text{atom}})$

$F'_{\text{en4}} = (\ldots, E{:}([]; \text{ret}), \texttt{Md}{:}\text{atom}(\text{aID}_e, aPol_e, NVw{:}\cdot, Vw{:}\cdot, rb{:}1),$
$(\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{d} := \text{diff}(\text{prev}, \text{x}); \text{prev} := \text{x}; \text{read}(\text{log}); \text{end}_{\text{cr}}); \text{end}_{\text{atom}})$

$F'_{\text{en12}} = (\ldots, \text{end}_{\text{atom}})$

$F_{\text{en13}} = (\ldots, \texttt{Md}{:}\text{jit}, \text{skip})$

$F_{\text{en14}} = (\ldots, E{:}\cdot, \texttt{Md}{:}\text{jit}, \text{ret})$

$F_{\text{fin}} = (\cdot, \cdot, \cdot, \cdot, \cdot, \text{skip})$

Frames for ISR_IN

$F_{\text{in1}} = (pID{:}p1, ID{:}\text{ISR\_IN}, \overrightarrow{pr}{:}2, vPol_i, v{:}inp \mapsto v_i, E{:}([]; \text{ret}), \texttt{Md}{:}\text{discard}(\cdot, \cdot),$
$\text{let y} = \text{in}() \text{ in } (\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{x} := \text{y}; \text{log} := \text{d}; \text{end}_{\text{cr}}); \text{ret})$

$F_{\text{in2}} = (\ldots, E{:}([]; \text{ret}), \texttt{Md}{:}\text{discard}(\cdot, \cdot), \text{let y} = \text{in}() \text{ in } (\text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); \text{x} := \text{y}; \text{log} := \text{d}; \text{end}_{\text{cr}}))$

$F_{\text{in3}} = (\ldots, v{:}(inp \mapsto v_i, y \mapsto 11), E{:}([]; \text{ret}), \ldots, \text{start}_{\text{cr}}(pr{:}2, ShAcc{:}\{\text{x}, \text{d}\}); x := y; \text{log} := \text{d}; \text{end}_{\text{cr}})$

$F_{\text{in4}} = (\ldots, \overrightarrow{pr}{:}2 \rhd 2, \ldots, E{:}\text{end}_{\text{cr}} \rhd ([]; \text{ret}), \ldots, x := y; \text{log} := \text{d})$

$F_{\text{in5}} = (\ldots, E{:}([]; \text{log} := \text{d}) \rhd (\text{end}_{\text{cr}} \rhd ([]; \text{ret})), \ldots, x := y)$

$F_{\text{in6}} = (\ldots, \text{skip})$

E.6 Type System

In this section, we define the types and type checking rules of our system.

E.6.1 Types

access qualifier	$qAcc$	::=	$\texttt{RD} \mid \texttt{WT} \mid \texttt{CK}$
id qualifier	qID	::=	$\mathsf{Id}(qAcc) \mid \mathsf{Id} \mid \mathsf{Nid}$
type qualifier	q	::=	$\langle qID, qID, qID \rangle$
Typing context	Γ	::=	$\cdot \mid \Gamma, x : \mathsf{int}@q$
pc	pc	::=	$\mathsf{Id} \mid \mathsf{Nid}$
writes	$MFWts$	::=	$\cdot \mid x, MFWts$
dependent variables	Λ	::=	$\cdot \mid x, \Lambda$

Our system uses typing contexts Γ to check each task separately. Each entry in Γ has the form $x : \mathsf{int}@q$ where variable x is always of type int and is annotated with a *type qualifier* q indicating whether a variable's idempotence (Id) or nonidempotence (Nid).

Because tasks can be preempted nondeterministically at any point while running a given task, it is important to reason about variable idempotence with respect to *local* usage within a task and also *global* usage by other tasks in the system. To capture this, type qualifiers are a triple of *idempotence qualifiers* $\langle qID, qID, qID \rangle$, where the first component represents the local (non-)idempotence of x, and the second and third components represent how x is used by tasks of lower and higher priority, respectively. When checking atomic regions, local variable qualifiers that are Id come with an interior *access qualifier* which indicates whether a variable is read-only (ρ), written ($\texttt{WT}$), or checkpointed ($\texttt{CK}$) in that region.

In the checking rules, we use the write set *MFWts* to statically track variables that could have been written by an atomic region, and Λ to track the set of variables an expression depends on which is needed for checking branches. Expressions have simple type t of int or $\texttt{bool}$.

Intermediate definitions. We define the order of accesses as $\mathsf{Id}(\_) \sqsubseteq \mathsf{Nid}$. We define the join ($\sqcup$) over qualifiers q below:

$$\frac{qID'_c = qID^1_c \sqcup qID^2_c \qquad qID'_l = qID^1_l \sqcup qID^2_l \qquad qID'_h = qID^1_h \sqcup qID^2_h}{\langle qID^1_c, qID^1_l, qID^1_h \rangle \sqcup \langle qID^2_c, qID^2_l, qID^2_h \rangle = \langle qID'_c, qID'_l, qID'_h \rangle}\ \textsc{q-join}$$

$$\frac{qID'_c = pc \sqcup qID_c \sqcup qID_l \sqcup qID_h}{\langle qID_c, qID_l, qID_h \rangle \sqcup pc = qID'_c}\ \textsc{pc-join}$$

In Q-JOIN, we define the join over qualifiers $\langle qID_c, qID_l, qID_h \rangle$ as the join over their respective components. We say that $\mathsf{Id}(a_1) \sqcup \mathsf{Id}(a_2)$ is undefined if $a_1 \neq a_2$. When checking branches, we need to compute the join of a q triple and a pc (either Id or Nid), which is just the join of all components of q and pc (as in PC-JOIN).

In the next sections, we give fine-grained typing rules for our system, beginning with expressions, then commands, and lastly tasks.

E.6.2 Expression Typing Rules

We define the typing rules for expressions below:

$$\boxed{\Gamma \vdash^{\mathsf{jit}} e : t@q, \Lambda \quad \Gamma \vdash^{\mathsf{discard}} e : t@q, \Lambda \quad \Gamma, \mathit{MFWts} \vdash^{\mathsf{atom}} e : t@q, \Lambda}$$

$$\frac{\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma \vdash^{\mathtt{sMd}} x : t@\langle qID_c, qID_l, qID_h\rangle, \{x\}}\ \text{T-Var-Read1}$$

$$\frac{\Gamma(x) = t@\langle \mathsf{Id}(\mathtt{WT}), qID_l, qID_h\rangle \qquad x \in \mathit{MFWts}}{\Gamma, \mathit{MFWts} \vdash^{\mathsf{atom}} x : t@\langle \mathsf{Id}, qID_l, qID_h\rangle, \{x\}}\ \text{T-Var-Read2}$$

$$\frac{\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \qquad qID_c \neq \mathsf{Id}(\mathtt{WT})}{\Gamma, \mathit{MFWts} \vdash^{\mathsf{atom}} x : t@\langle \overline{qID}_c, qID_l, qID_h\rangle, \{x\}}\ \text{T-Var-Read3}$$

$$\frac{\Gamma \vdash^{\mathtt{sMd}} e_1 : t@q_1, \Lambda_1 \qquad \Gamma \vdash^{\mathtt{sMd}} e_2 : t@q_2, \Lambda_2 \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma \vdash^{\mathtt{sMd}} e_1 \odot e_2 : t@(q_1 \sqcup q_2), \Lambda_1 \cup \Lambda_2}\ \text{T-Binop-Read-DJ}$$

$$\frac{\Gamma, \mathit{MFWts} \vdash^{\mathsf{atom}} e_1 : t@q_1, \Lambda_1 \qquad \Gamma, \mathit{MFWts} \vdash^{\mathsf{atom}} e_2 : t@q_2, \Lambda_2}{\Gamma, \mathit{MFWts} \vdash^{\mathsf{atom}} e_1 \odot e_2 : t@(q_1 \sqcup q_2), \Lambda_1 \cup \Lambda_2}\ \text{T-Binop-Read-A}$$

Our typing judgments are indexed by simple mode $\mathtt{sMd}$, where checking an expression under the mode atom depends on the writes set MFWts and context Γ and modes jit and $\mathsf{discard}$ only depend on Γ. We check that x has simple type t with qualifier q, but where each component is restricted to just Id or Nid.

If the simple mode is either jit or $\mathsf{discard}$, the rule T-Var-Read1 checks a variable x by looking up its type in Γ. The set $\{x\}$ is returned, indicating that x was read.

The rules T-Var-Read2 and T-Var-Read3 check a variable x under the mode atom. The rule T-Var-Read2 says that if the local qualifier of x in Γ is $\mathsf{Id}(\mathtt{WT})$, then x should type check only if x is already in MFWts, meaning that it is a must-first-write that has already been written to and hence is safe to read here. The resulting type of x is the same but where the local qualifier is flattened to Id. Otherwise, if x does not have local qualifier $\mathsf{Id}(\mathtt{WT})$, the rule T-Var-Read3 looks up the type of x in Γ and checks x according to that type with flattened local qualifier $\overline{qID_c}$, defined as $\overline{\mathsf{Id}(\_)} = \mathsf{Id}$, $\overline{\mathsf{Id}} = \mathsf{Id}$, and $\overline{\mathsf{Nid}} = \mathsf{Nid}$.

The rules T-Binop-Read-DJ and T-Binop-Read-A check binary operation $e_1 \odot e_2$. If the mode is jit or $\mathsf{discard}$, the rule T-Binop-Read-J checks e_1 and e_2 separately, respectively computing qualifiers q_1 and q_2 and variable sets Λ_1 and Λ_2. The resulting type of $e_1 \odot e_2$ is the join of qualifiers $q_1 \sqcup q_2$, and the resulting variable set is $\Lambda_1 \cup \Lambda_2$. The rule T-Binop-Read-A is similar, checking $e_1 \odot e_2$ under mode atom.

E.6.3 Command Typing Rules (in mode atom).

Our command typing judgments are also indexed by a simple mode and also the current task priority pr_t which is needed for critical sections. In this section, we give the command typing rules for the mode atom. Note that we do not allow nested atomic regions and posting events from atomic regions, so checking rules for starting an atomic region and posting events are excluded from this set of rules.

$$\boxed{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts'}$$

$$\frac{\Gamma_0, MFWts \vdash^{\mathsf{atom}} e : \mathtt{bool}@q_e, \Lambda_e \qquad \Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}}_{vPol} c_1 \Rightarrow MFWts' \qquad \Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}}_{vPol} c_2 \Rightarrow MFWts' \qquad \Lambda = \Lambda_e}{\Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}}_{vPol} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \Rightarrow MFWts'}\ \text{T-If-Nid-A}$$

$$\frac{\begin{array}{c}\Gamma_0, MFWts \vdash^{\mathsf{atom}} e : \mathtt{bool}@q_e, \Lambda_e \\ \Gamma_0, MFWts, \mathsf{Id} \sqcup q_e \vdash^{\mathsf{atom}}_{vPol} c_1 \Rightarrow MFWts_1 \qquad \Gamma_0, MFWts, \mathsf{Id} \sqcup q_e \vdash^{\mathsf{atom}}_{vPol} c_2 \Rightarrow MFWts_2 \\ W = (MFWts_1 \cup MFWts_2) \setminus (MFWts_1 \cap MFWts_2) \\ \forall x \in W, \Gamma_0(x) = \mathsf{Id}(\mathsf{CK}) \qquad MFWts' = MFWts_1 \cap MFWts_2 \qquad \Lambda = \Lambda_e\end{array}}{\Gamma_0, MFWts, \mathsf{Id} \vdash^{\mathsf{atom}}_{vPol} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \Rightarrow MFWts'}\ \text{T-If-Id-A}$$

The rules T-If-Nid-A and T-If-Id-A check conditionals when the pc is Id or Nid. In a conditional where the pc is Nid (T-If-Nid-A), we check each branch c_1 and c_2 separately as Nid. We assume the resulting writes sets are the same across branches because in Nid branches, the only variables that are allowed to be written are those that are also Nid and we do not update the writes set for Nid assignments (see assignment rules). We check e with type $\mathtt{bool}@q_e$ and read set Λ_e, and check that Λ_e is the same as Λ.

When the pc is Id, the rule T-If-Id-A applies. We first check e under type $\mathtt{bool}@q_e$ with read variables Λ_e. Then, we check branches c_1 and c_2 separately under $\mathsf{Id} \sqcup q_e$, producing write sets $MFWts_1$ and $MFWts_2$ respectively. The resulting writes set $MFWts'$ is the intersection of these two sets. The exclusive-may-write variables W have to be checkpointed.

$$\frac{\begin{array}{c}\Gamma(x) = t@q_x \\ q_x = \langle qID_c, qID_l, qID_h\rangle \qquad qID_c \neq \mathsf{Id}(\mathtt{RD}) \qquad \Gamma \vdash^{\mathsf{atom}} e : t@q_e, \Lambda_e \\ q_e \sqcup pc \sqsubseteq qID_c \qquad MFWts' = \begin{cases} MFWts & \text{if } q_c = \mathsf{Nid} \\ MFWts, x & \text{if } q_c = \mathsf{Id}(\_) \end{cases}\end{array}}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} x := e \Rightarrow MFWts'} \text{ T-Assign-N-A}$$

$$\frac{\Gamma(x) = t@q_x \qquad q_x = \langle \mathsf{Id}, qID_l, qID_h\rangle \qquad \Gamma \vdash^{\mathsf{atom}} e : t@q_e, \Lambda_e \qquad q_e \sqcup pc \sqsubseteq \mathsf{Id}}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} x := e \Rightarrow MFWts} \text{ T-Assign-V-A}$$

$$\frac{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c_1 \Rightarrow MFWts_1 \qquad \Gamma, MFWts_1, pc \vdash^{\mathsf{atom}}_{vPol} c_2 \Rightarrow MFWts_2}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c_1; c_2 \Rightarrow MFWts_2} \text{ T-Sequence-1-A}$$

$$\frac{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts' \qquad \Gamma, MFWts', pc \vdash^{\mathsf{atom}}_{vPol} \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}} \text{ T-Sequence-2-A}$$

$$\frac{\mathsf{dom}(\Gamma\downarrow_{\mathsf{Id}(\mathtt{WT})}) \subseteq MFWts}{\Gamma, MFWts, \mathsf{Id} \vdash^{\mathsf{atom}}_{vPol} \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}} \text{ T-Atom-End-A}$$

The T-Assign-N-A rule types an assignment of expression e to variable x in nonvolatile memory if it is not locally read-only. If x is locally $\mathsf{Id}(\_)$, then we need to collect it into our writes set $MFWts'$, and otherwise we do not. We check that it is okay to assign e of q_e to x of q_x by checking that the join of q_e and the current pc are bounded by qID_c. The rule T-Assign-V-A performs the same check but does not update $MFWts$, as it is for tracking nonvolatile writes only. We also assume that variables in volatile memory are always Id, because they are wiped on re-execution, although this assumption may be too restrictive (see discussion section).

Together, the rules T-Sequence-1-A and T-Sequence-2-A type sequences of commands inside of atomic regions. T-Sequence-2-A types the command inside of an atomic region $c; \mathsf{end}_{\mathsf{atom}}$ as ok given an initial writes set $MFWts$ if c is well-typed. If c is another sequence of commands $c_1; c_2$, we need to appeal to T-Sequence-1-A which first checks c_1 with Γ and $MFWts$, producing updated write set $MFWts_1$, and then it checks c_2 with Γ and write set $MFWts_1$, producing the final writes set $MFWts_2$.

T-Atom-End-A checks that all must-first-write variables (of type $\mathsf{Id}(\mathtt{WT})$) have been written to by the end of an atomic region and that the result argument is fully idempotent.

$$\frac{\begin{array}{c}\Gamma', MFWts, \mathsf{Id} \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts' \\ pr_{\mathsf{c}} = ceilingPrOf(\{pr' \mid \mathsf{tk}(ID, version, pr', vPol, rPol, \lambda x.c) \in PDecl \\ \wedge\ (ShAcc \cap vPol.ShAcc \neq \emptyset)\}) \qquad pr \geq pr_{\mathsf{c}} \qquad ShAcc \subseteq vPol.ShAcc \\ ShAcc \neq \emptyset \qquad \Gamma' = \{x : t@q \mid x \in \mathsf{dom}(\Gamma) \wedge x \notin vPol.ShAcc \setminus ShAcc\}\end{array}}{\Gamma, MFWts, \mathsf{Id} \vdash^{\mathsf{atom}}_{vPol} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \Rightarrow MFWts'} \text{ T-Critical-A}$$

The rule T-Critical-A checks a critical section in atomic mode. It checks the command c, and that the priority pr is high enough. We check that pr is at least as high as the task priority pr_{t}

(which was inserted in the high-level check T-Task) and that pr is at least as high as the priority of other tasks with common shared accesses pr_c, which ensures that c will not be preeempted by these tasks. Additionally, the rule checks that the critical section's *ShAcc* annotation is correct, in that it can only be a subset of the task's *ShAcc* set, and that the only shared variables its code accesses are the ones in *ShAcc*. We enforce the former by checking that $ShAcc \subseteq ShAcc_t$, and the latter by restricting the typing context Γ' to exclude shared variables that appear in $ShAcc_t$ but not in *ShAcc*.

$$\frac{\Gamma, MFWts \vdash^{\mathsf{atom}} e : t@q, \Lambda_e \qquad (\Gamma, x : t@q), MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts'}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} \mathsf{let}\ x = e\ \mathsf{in}\ c \Rightarrow MFWts'}\ \text{T-let-e-A}$$

$$\frac{(\Gamma, x : t@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle), MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts'}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} \mathsf{let}\ x = \mathsf{IN}()\ \mathsf{in}\ c \Rightarrow MFWts'}\ \text{T-let-IN-A}$$

$$\frac{}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} \mathsf{skip} \Rightarrow MFWts}\ \text{T-Skip-A}$$

The rule T-let-e-A for checking the binding of an expression to variable x is standard. We first check $e : t@q$, and then c with $x : t@q$ injected into the context, producing the final output set $MFWts'$. The rule T-let-IN-A is similar, but we restrict the input to be Id, assuming that it is the same every time. The rule T-Skip-A checks a skip command under atomic mode without updating the writes set *MFWts*.

E.6.4 Command typing rules (in mode jit and discard).

The following rules are the same for jit and discard, because commands do not re-execute in these modes. They are similar to the ones for atomic mode, however id qualifiers do not track access modes meaning that we do not need the written variable sets in this judgment.

$\boxed{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} c : \mathsf{ok}}$

$$\frac{\Gamma \vdash^{\mathtt{sMd}} e : \mathtt{bool}@q_e, \Lambda_e \qquad \Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}}_{vPol} c_1 : \mathsf{ok} \qquad \Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}}_{vPol} c_2 : \mathsf{ok} \qquad \Gamma_1, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}}_{vPol} \mathsf{end_{if}} : \mathsf{ok} \qquad \Lambda = \Lambda_e \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc_0 \vdash^{\mathtt{sMd}}_{vPol} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \mathsf{ok}}\ \text{T-If-DJ}$$

$$\frac{\Gamma(x) = t@q_x \qquad \Gamma \vdash^{\mathtt{sMd}} e : t@q_e, \Lambda_e \qquad q_e \sqcup pc \sqsubseteq qID_c \qquad q_x = \langle qID_c, qID_l, qID_h \rangle \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} x := e : \mathsf{ok}}\ \text{T-Assign-DJ}$$

$$\frac{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} c_1 : \mathsf{ok} \qquad \Gamma_1, pc \vdash^{\mathtt{sMd}}_{vPol} c_2 : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} c_1; c_2 : \mathsf{ok}}\ \text{T-Sequence-DJ}$$

$$\frac{\Gamma \vdash^{\mathtt{sMd}} e : t@q, \Lambda_e \qquad (\Gamma, x : t@q), pc \vdash^{\mathtt{sMd}}_{vPol} c : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} \mathsf{let}\ x = e\ \mathsf{in}\ c : \mathsf{ok}}\ \text{T-let-e-DJ}$$

$$\frac{(\Gamma, x : t@\langle \mathsf{ld}, \mathsf{ld}, \mathsf{ld} \rangle), pc \vdash^{\mathtt{sMd}}_{vPol} c : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} \mathsf{let}\ x = \mathsf{IN}()\ \mathsf{in}\ c : \mathsf{ok}}\ \text{T-let-IN-DJ}$$

$$\frac{\begin{array}{c}\Gamma', \mathsf{ld} \vdash^{\mathtt{sMd}}_{vPol} c : \mathsf{ok} \qquad pr \geq pr_{\mathsf{t}} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \\ pr_{\mathsf{c}} = ceilingPrOf(\{pr' \mid \mathsf{tk}(ID, version, pr', vPol, rPol, \lambda x.c) \in PDecl \\ \wedge\ (ShAcc \cap vPol.ShAcc \neq \emptyset)\}) \qquad pr \geq pr_{\mathsf{c}} \\ ShAcc \subseteq vPol.ShAcc \qquad \Gamma' = \{x : t@q \mid x \in \mathsf{dom}(\Gamma) \wedge x \notin vPol.ShAcc \setminus ShAcc\}\end{array}}{\Gamma, \mathsf{ld} \vdash^{\mathtt{sMd}}_{vPol} \mathsf{start_{cr}}(pr, ShAcc); c; \mathsf{end_{cr}} : \mathsf{ok}}\ \text{T-Critical-DJ}$$

The rule T-If-DJ checks conditional branches by first checking e with qualifier q_e, and then c_1 and c_2 separately by raising the initial pc_0 to $pc_0 \sqcup q_e$. The rule T-Assign-DJ checks assignment of e to x. Similar to T-Assign-V-A from above, we check that the join of q_e and the initial pc is bounded by the local qualifier of x (qID_c). The rule T-Sequence-DJ checks a sequenced command $c_1; c_2$ by checking c_1 and c_2 separately. The rule T-let-e-DJ for checking a let binding is similar to T-let-e-A from above.

The rule for checking critical sections T-Critical-DJ is similar to T-Critical-A, but defined here for modes jit and discard.

$$\frac{\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} \mathsf{skip} : \mathsf{ok}}\ \text{T-Skip-DJ}$$

$$\frac{\Gamma \vdash^{\mathsf{jit}}_{vPol} v : t@q_e \qquad q_e \sqcup pc \sqsubseteq \mathsf{ld}}{\Gamma, pc \vdash^{\mathsf{jit}}_{vPol} \mathsf{post}(eID, v) : \mathsf{ok}}\ \text{T-Post-J}$$

$$\frac{\Gamma, pc \vdash^{\mathsf{discard}}_{vPol} c : \mathsf{ok} \qquad \Gamma \vdash^{\mathtt{sMd}} res : t@\langle \mathsf{ld}, \mathsf{ld}, \mathsf{ld}\rangle, \Lambda_e}{\Gamma, pc \vdash^{\mathsf{discard}}_{vPol} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}}\ \text{T-Atom-Begin-D}$$

The rule T-Skip-DJ checks skip commands. The rule T-Post-DJ checks post commands, which we only allow in the mode jit for simulating a continuous execution of posted tasks. We disallow posting events in discard mode because if this task fails, there is no matching continuous execution.

We define a typing rule for the initialization command *initN*(*Inits*), which checks that each variable in *Inits* has a typing in Γ. For atomic regions, these variables are then added to the writes set.

$$\frac{\forall x \in Inits \qquad \Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle}{\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} initN(Inits) \Rightarrow MFWts \cup \{Inits\}}\ \text{T-Init-A}$$

$$\frac{\forall x \in Inits \qquad \Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} initN(Inits) : \mathsf{ok}}\ \text{T-Init-DJ}$$

$$\frac{\begin{array}{l}\omega \cap \rho = \emptyset \qquad \rho \cap \mu = \emptyset \qquad \omega \cap \mu = \emptyset \qquad \omega \cap Inits = \emptyset \\ \forall\, x_k \in \omega \; s.t.\; \Gamma(x_k) = t_k@\langle qID^k_c, qID^k_l, qID^k_h\rangle \,.\, qID^k_l \sqcup qID^k_h \neq \mathsf{Nid} \\ \forall\, x_w \in \mu \cup Inits \; s.t.\; \Gamma(x_w) = t_w@\langle q^w_c, q^w_l, q^w_h\rangle \,.\, q^w_h \neq \mathsf{Nid} \\ vPol = (ShAcc_t, ShCP_t, nIds_t, Inits_t) \\ \Gamma' = mkCtxAtom(\Gamma, \omega, \mu, \beta, \rho, Inits, nIds_t) \\ \Gamma', \cdot, pc \vdash^{\mathsf{atom}}_{vPol} initN(Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \\ \Gamma' \vdash^{\mathsf{atom}} res : t@\langle qID_c, \mathsf{ld}, \mathsf{ld}\rangle, \Lambda_e \qquad \overline{qID_c} = \mathsf{ld}\end{array}}{\Gamma, pc \vdash^{\mathsf{jit}}_{vPol} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}}\ \text{T-Atom-Begin-J}$$

The rule T-Atom-Begin-J checks atomic regions under jit where task re-execution makes checkpointing variables necessary. The rule checks the inner command c under the mode atom. We check that the sets of variables ω, ρ, and μ are disjoint. For every variable in ω, we check that there are no non-idempotent writes from any other tasks, to ensure we do not checkpoint any non-idempotent data. For every variable in μ, we check that there are no non-idempotent writes from any higher-priority tasks, because these could preempt us and write a Nid value the middle of running an atomic region, but variables in μ are expected to remain idempotent after being initialized (see *mkCtxAtom* below). Given an initial context Γ, we compute a context Γ′ and check the inner command c; $\mathsf{end}_{\mathsf{atom}}$ under it.

Below, we define *mkCtxAtom* for computing a context for checking atomic regions according to an intial Γ that is restricted to atomic region and task-level *nIds* variable sets. For every

variable in ω that is Id in Γ, we update the qualifier to $\mathsf{Id}(\mathtt{CK})$, and similarly for ρ. The *Inits* and μ variables low priority qualifiers are set to Id, and local qualifiers are set to $\mathsf{Id}(Wt)$. We assume that the local qualifiers of both are already Id because the task-level *Inits* set would have already reset any Nid qualifiers to Id, so it is just Id here. Γ_v types all variables in β, whose types will all consist of singleton idempotence qualifiers. No access qualifiers are needed for these variables since volatile memory does not persist through power failure. Γ_n types all variables in the task-level set *nIds*, which could also be accessed in the atomic region.

$$\frac{\begin{array}{c}\Gamma_k = \{x : t@\langle \mathsf{Id}(\mathtt{CK}), qID_l, qID_h\rangle \mid \Gamma(x) = t@\langle \mathsf{Id}, qID_l, qID_h\rangle, x \in \omega\} \\ \Gamma_m = \{x : t@\langle \mathsf{Id}(\mathtt{WT}), \mathsf{Id}, qID_h\rangle \mid \Gamma(x) = t@\langle \mathsf{Id}, qID_l, qID_h\rangle, x \in \mu\} \\ \Gamma_r = \{x : t@\langle \mathsf{Id}(\mathtt{RD}), qID_l, qID_h\rangle \mid \Gamma(x) = t@\langle \mathsf{Id}, qID_l, qID_h\rangle, x \in \rho\} \\ \Gamma_i = \{x : t@\langle \mathsf{Id}(\mathtt{WT}), \mathsf{Id}, qID_h\rangle \mid \Gamma(x) = t@\langle \mathsf{Id}, qID_l, qID_h\rangle, x \in \mathit{Inits}\} \\ \Gamma_v = \Gamma \restriction \beta \qquad \Gamma_n = \Gamma \restriction \mathit{nIds}\end{array}}{\mathit{mkCtxAtom}(\Gamma, \omega, \mu, \beta, \rho, \mathit{Inits}, \mathit{nIds}) = \Gamma_k \cup \Gamma_m \cup \Gamma_r \cup \Gamma_i \cup \Gamma_v \cup \Gamma_n}$$

E.6.5 High-level type checking rules

The high-level rules compute a context Ψ for checking *PDecl*.

$$\frac{}{\Psi \vdash \cdot : \mathsf{ok}}\ \text{T-TASK-EMP} \qquad \frac{\Psi \vdash \mathit{tsk} : \mathsf{ok}}{\Psi \vdash \mathit{tsk}, \mathit{TDecl} : \mathsf{ok}}\ \text{T-TASK-IND}$$

$$\frac{\mathit{PDecl} = (\mathsf{post}(\mathit{eID}, v)_{\mathsf{entry}}, \mathit{IDecl}, \mathit{EDecl}) \qquad \Psi \vdash \mathit{IDecl} : \mathsf{ok} \qquad \Psi \vdash \mathit{EDecl} : \mathsf{ok}}{\Psi \vdash \mathit{PDecl} : \mathsf{ok}}\ \text{T-PDecl}$$

$$\frac{\mathit{extractEffect}(\mathit{PDecl}) = \xi \qquad \mathit{mkGlobCtx}(\mathit{PDecl}, \xi) = \Psi \qquad \Psi \vdash \mathit{PDecl} : \mathsf{ok}}{\vdash \mathit{PDecl} : \mathsf{ok}}\ \text{T-PDecl-top}$$

In the rule T-PDecl-top, we use *extractEffect* to gather the sets of variables with unstable or non-idempotent writes, which we call ξ. Using *PDecl* and ξ, *mkGlobCtx* computes a typing context for each task (Ψ), including a type qualifier for each shared variable accessed by the task, and accounting for the unstable/non-idempotent effects of all other tasks. The rules T-PDecl, T-task-ind, and T-task-emp check the *PDecl* under Ψ.

$$\frac{PDecl = (\mathsf{post}(eID, v)_{\mathsf{entry}}, IDecl, EDecl) \quad mkCtx(IDecl, \xi) = \Psi_1 \quad mkCtx(EDecl, \xi) = \Psi_2}{mkGlobCtx(PDecl, \xi) = \Psi_1, \Psi_2}\ \textsc{mkGCtx-Top}$$

$$\frac{\begin{array}{c} t = \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) \\ vPol = (ShAcc, ShCP, nIds, Inits) \quad \mathsf{immediate} \in range(rPol) \\ N_{low} = \cup\{v_{\mathsf{Nid},hi} \mid (ID^0, version^0, pr^0) \mapsto (v_{\mathsf{Nid},leq}, v_{\mathsf{Nid},hi}) \in \xi, pr^0 < pr\} \\ N_{heq} = \cup\{v_{\mathsf{Nid},leq} \mid (ID^0, version^0, pr^0) \mapsto (v_{\mathsf{Nid},leq}, v_{\mathsf{Nid},hi}) \in \xi, pr^0 \geq pr\} \\ \Gamma_i = \{x_i : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, glob(x_i, N_{heq})\rangle \mid x_i \in Inits\} \\ \Gamma_s = \{x_s : \mathsf{int@}\langle \mathsf{Id}, glob(x_s, N_{low}), glob(x_s, N_{heq})\rangle \mid x_s \in ShAcc \setminus (Inits \cup nIds)\} \\ \Gamma_n = \{x_n : \mathsf{int@}\langle \mathsf{Nid}, glob(x_n, N_{low}), glob(x_n, N_{heq})\rangle \mid x_n \in nIds\} \end{array}}{mkCtx((t, TDecl), \xi) = ((ID, version, \mathsf{jit}) \mapsto (\Gamma_s \cup \Gamma_i \cup \Gamma_n)), mkCtx(TDecl, \xi)}\ \textsc{mkCtx-jit}$$

$$\frac{\begin{array}{c} t = \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) \\ vPol = (ShAcc, ShCP, nIds, Inits) \quad \{\mathsf{discard}, \mathsf{next}\} \cap range(rPol) \neq \emptyset \\ N_{low} = \cup\{v_{\mathsf{Nid},hi} \mid (ID^0, version^0, pr^0) \mapsto (v_{\mathsf{Nid},leq}, v_{\mathsf{Nid},hi}) \in \xi, pr^0 < pr\} \\ N_{heq} = \cup\{v_{\mathsf{Nid},leq} \mid (ID^0, version^0, pr^0) \mapsto (v_{\mathsf{Nid},leq}, v_{\mathsf{Nid},hi}) \in \xi, pr^0 \geq pr\} \\ \forall\, x_c \in ShCP, glob(x_c, N_{low}) \sqcup glob(x_c, N_{heq}) = \mathsf{Id} \\ \Gamma_c = \{x_c : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle \mid x_c \in ShCP\} \\ \Gamma_i = \{x_i : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, glob(x_i, N_{heq})\rangle \mid x_i \in Inits\} \\ \Gamma_s = \{x_s : \mathsf{int@}\langle \mathsf{Nid}, glob(x_s, N_{low}), glob(x_s, N_{heq})\rangle \mid x_s \in ShAcc \setminus (Inits \cup ShCP \cup nIds)\} \\ \Gamma_n = \{x_n : \mathsf{int@}\langle \mathsf{Nid}, glob(x_n, N_{low}), glob(x_n, N_{heq})\rangle \mid x_n \in nIds\} \end{array}}{mkCtx((t, TDecl), \xi) = ((ID, version, \mathsf{discard}) \mapsto (\Gamma_s \cup \Gamma_c \cup \Gamma_n \cup \Gamma_i)), mkCtx(TDecl, \xi)}\ \textsc{mkCtx-d}$$

$$\frac{}{mkCtx(\cdot, \xi) = \cdot}\ \textsc{mkCtx-empty} \qquad \frac{qID = \begin{cases} \mathsf{Nid} & \text{if } x \in N \\ \mathsf{Id} & \text{otherwise} \end{cases}}{glob(x, N) = qID}\ \textsc{q-global}$$

The rules mkCtx-jit and mkCtx-discard compute contexts for immediate and discard tasks (which execute under the modes jit and discard) given variable sets *ShAcc*, *ShCP*, *Inits*, and *nIds*. We use the collective $v_{\mathsf{Nid},hi}$ sets from tasks in ξ with strictly lower priority to construct the second qualifier, and we use the collective $v_{\mathsf{Nid},leq}$ sets from higher- or equal-priority tasks to construct the third qualifier. ($v_{\mathsf{Nid},hi}$ indicates unstable/non-idempotent writes *affecting* higher-priority tasks, and $v_{\mathsf{Nid},leq}$ indicates writes *affecting* lower- or equal-priority tasks; thus we collect $v_{\mathsf{Nid},hi}$ from all tasks with priority *lower* than t, and collect $v_{\mathsf{Nid},leq}$ from all tasks with priority *higher* or equal to t).

The key difference between mkCtx-jit and mkCtx-discard is that there is no need for task-level checkpointing in immediate tasks (outside of atomic regions) because the default mode is jit; if there were a power failure, execution would resume from the exact same place before the interruption. So, these tasks do not need to checkpoint shared variables. See *extractEffect* below.

$$\frac{\begin{array}{l}vPol = (ShAcc, ShCP, nIds, Inits) \qquad Inits \cap ShCP = \emptyset \qquad Inits \cap nIds = \emptyset \\ \text{if } range(rPol) \cap \{\mathsf{discard}, \mathsf{alternative}\} \neq \emptyset \text{ then } \Gamma_d = \Psi(ID, version, \mathsf{discard}) \\ (\Gamma_d, x : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle), \mathsf{Id} \vdash^{\mathsf{discard}}_{vPol} initN(Inits); c; \mathsf{ret}(res) : \mathsf{ok} \\ \text{ and } \Gamma_d, x : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle \vdash^{\mathsf{discard}} res : \langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle \\ \text{if } \mathsf{immediate} \in range(rPol) \text{ then } \Gamma_j = \Psi(ID, version, \mathsf{jit}) \\ (\Gamma_j, x : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle), \mathsf{Id} \vdash^{\mathsf{jit}}_{vPol} initN(Inits); c; \mathsf{ret}(res) : \mathsf{ok} \\ \text{ and } \Gamma_j, x : \mathsf{int@}\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle \vdash^{\mathsf{jit}} res : \langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id}\rangle\end{array}}{\Psi \vdash \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) : \mathsf{ok}}$$

T-Task

T-Task is the high-level type checking rule for tasks. Triggered by T-task-ind, it checks a task in the *PDecl* according to the precomputed Ψ. We check the task command under a different context depending on its re-execution behavior. If the re-execution policy is discard or alternative, we type it under discard mode using the task's discard context.

E.6.6 The algorithm for *extractEffect*

For each task in *PDecl*, we compute its effect on other tasks. The *nidEffectToHigherPr*(t) rules compute the set of unstable and non-idempotent writes performed by task t that affect relatively higher-priority tasks, and the *nidEffectToLeqPr*(*t*) rules compute the set of unstable and non-idempotent writes affecting tasks with a priority lower than or equal to t.

$$\frac{}{extractEffect(\emptyset) = \emptyset}\ \textsc{Ext-Empty}$$

$$\frac{\begin{array}{c}\mathsf{t} = \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_c) \\ v_{\mathsf{Nid},leq} = nidEffectToLeqPr(\mathsf{t}) \qquad v_{\mathsf{Nid},hi} = nidEffectToHigherPr(\mathsf{t})\end{array}}{extractEffect(\mathsf{t}, PDecl) = \Big((ID_c, version_c, pr_c) \mapsto (v_{\mathsf{Nid},leq}, v_{\mathsf{Nid},hi})\Big) \cup extractEffect(PDecl)}\ \textsc{Ext-t}$$

The rule Ext-Empty is the base case; if the input is empty, then there remain no more tasks to analyze and we return the empty set. Otherwise, Ext-t analyzes the next task t in *PDecl*, appealing to *nidEffectToHigherPr* and *nidEffectToLeqPr* to compute the sets of unstable and non-idempotent variables ($v_{\mathsf{Nid},leq}$ and $v_{\mathsf{Nid},hi}$).

$$\frac{\mathsf{t} = \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) \quad range(rPol) \in \{\mathsf{discard}, \mathsf{next}\} \quad vPol = (ShAcc, ShCP, nIds, Inits)}{nidEffectToHigherPr(\mathsf{t}) = nIds \cup ShAcc\restriction^{\mathsf{wt}} \cup ShCP}\ \textsc{Compute-Higher-Discard}$$

$$\frac{\begin{array}{c}\mathsf{t} = \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) \quad range(rPol) = \mathsf{jit} \quad vPol = (ShAcc_t, ShCP_t, nIds_t, Inits_t) \\ A = \{aPol \mid \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, aPol) \in \mathsf{atoms}(c)\} \quad W_a = \bigcup_{aPol \in A} (aPol.\omega) \cup (aPol.\mu) \cup (aPol.\beta) \cup (aPol.Inits)\end{array}}{nidEffectToHigherPr(\mathsf{t}) = nIds_t \cup W_a}\ \textsc{Compute-Higher-Immed}$$

The rules Compute-Higher-Discard and Compute-Higher-Immed define the *nidEffectToHigherPr* algorithm, which takes a task and returns a set of variables with non-idempotent and unstable writes affecting higher-priority tasks. If the policy is discard or alternative, then the rule Compute-Higher-Discard applies.

The rule Compute-Higher-Discard computes the set of non-idempotent and unstable writes in discard mode, which consists of (1) the task-level non-idempotent variables (*nIds*), and (2) the shared writes (including the task-level checkpointed variables *ShCP*). We need (2) because a higher-level task could checkpoint an unstable write from t which could be discarded after a power failure. The rule Compute-Higher-Immed computes the set of non-idempotent and unstable writes in jit mode, which consists of (1) task-level *nIds* and (2) the sets of mutable variables for each atomic region, μ, *Inits*, and β. We need (2) because a task could interrupt an atomic region and rely on the unstable writes from its partial execution (then after rebooting, the atomic region will be re-executed, and a future interrupt could read a value written by an earlier statement in the atomic region; this would violate consistency with any continuous execution).

$$\frac{\mathsf{t} = \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) \quad range(rPol) \in \{\mathsf{discard}, \mathsf{next}\} \quad vPol = (ShAcc, ShCP, nIds, Inits)}{nidEffectToLeqPr(\mathsf{t}) = nIds \cup (ShAcc\restriction^{\mathsf{wt}} \setminus ShCP)}\ \textsc{Compute-Leq-Discard}$$

$$\frac{\mathsf{t} = \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c) \quad range(rPol) = \mathsf{jit} \quad vPol = (ShAcc, ShCP, nIds, Inits)}{nidEffectToLeqPr(\mathsf{t}) = nIds}\ \textsc{Compute-Leq-Immed}$$

The rules Compute-Leq-Discard and Compute-Leq-Immed provide a similar algorithm to compute writes affecting tasks with lower or equal priority than t. For discard tasks (Compute-Leq-Discard), the shared checkpoints are not considered Nid. Even if the system loses power before the current task completes, any checkpointed variables will be restored upon reboot; thus no other task can observe an unstable write except by preempting t, which would require a strictly higher priority than t. In Compute-Leq-Immed, the resulting set is just the task-level *nIds* because all other writes would stabilize by the time the task finishes.

E.7 Runtime constructs typing rules

$$\frac{\Psi, PDecl \vdash F : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok} \qquad K = (\mathsf{N}_k, \mathsf{V}_k, S_k) \\ F, K \vdash \mathsf{N}, \mathsf{V} : \mathsf{ok} \qquad \text{if } mdOf(F) = \mathsf{jit} \text{ and } pidOf(F) = pID, \text{ then } S_k = F_j(pID) \rhd S'_k}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}} \text{ T-Config}$$

$$\frac{\Psi, PDecl \vdash K : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q) : \mathsf{ok}} \text{ T-Config-reboot}$$

In T-Config, the last two conditions establish that values are written through to the checkpointed memories in jit mode in every applicable step, i.e., on every write. We prove this condition is upheld for each relevant configuration through our preservation proof, with the key cases being assignment and ending an atomic region.

E.7.1 Frame typing

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts \qquad \Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}}_{vPol} E : \mathsf{ok} \\ \forall NVw' \text{ s.t. } NVw' \supseteq NVw \text{ then } \exists MFWts' \supseteq MFWts \text{ s.t. } \Gamma_0, NVw', pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts' \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-A2A}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}}_{vPol} c : \mathsf{ok} \qquad \Gamma, pc \rhd pcS \vdash^{\mathsf{jit}}_{vPol} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-A2J}$$

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \\ \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathtt{sMd}) \qquad \Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} c : \mathsf{ok} \qquad \Gamma, pc \rhd pcS \vdash^{\mathtt{sMd}}_{vPol} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-Jit-D}$$

$$\frac{}{\Psi, PDecl \vdash F_J(pID) : \mathsf{ok}} \text{ T-Frame-Jit-P}$$

E.7.2 Execution context typing

We type check the execution contexts in a similar manner to typing sequenced commands We check every command c on E under the current mode and the task $vPol$.

$$\boxed{\Gamma, MFWts, pcS \vdash^{\mathsf{atom}}_{vPol} E \Rightarrow MFWts' \qquad \Gamma, pcS \vdash^{\mathtt{sMd}}_{vPol} E : \mathsf{ok}}$$

$$\frac{\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, \mathsf{Id} \vdash^{\mathtt{sMd}}_{vPol} \cdot : \mathsf{ok}}\ \text{T-E-MD}$$

$$\frac{\Gamma, pc \vdash^{\mathtt{sMd}}_{vPol} c : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}}_{vPol} E : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad pcS = pc \rhd pcS'}{\Gamma, pcS \vdash^{\mathtt{sMd}}_{vPol} ([\,]; c) \rhd E : \mathsf{ok}}\ \text{T-E-SEQ-1}$$

$$\frac{\begin{array}{c}\Gamma, MFWts, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts' \\ \Gamma', MFWts', pcS \vdash^{\mathsf{atom}}_{vPol} E : \mathsf{ok} \qquad pcS = pc \rhd pcS'\end{array}}{\Gamma, MFWts, pcS \vdash^{\mathsf{atom}}_{vPol} ([\,]; c) \rhd E : \mathsf{ok}}\ \text{T-E-SEQ-2}$$

$$\frac{\Gamma, pcS' \vdash^{\mathtt{sMd}}_{vPol} E : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad pcS = pc \rhd pcS'}{\Gamma, pcS \vdash^{\mathtt{sMd}}_{vPol} \mathsf{end}_{\mathsf{if}} \rhd E : \mathsf{ok}}\ \text{T-E-IF-END-J}$$

$$\frac{\Gamma, MFWts, pcS' \vdash^{\mathsf{atom}}_{vPol} E : \mathsf{ok} \qquad pcS = pc \rhd pcS'}{\Gamma, MFWts, pcS \vdash^{\mathsf{atom}}_{vPol} \mathsf{end}_{\mathsf{if}} \rhd E : \mathsf{ok}}\ \text{T-E-IF-END-A}$$

$$\frac{\Gamma, pcS \vdash^{\mathtt{sMd}}_{vPol} E : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pcS \vdash^{\mathtt{sMd}}_{vPol} \mathsf{end}_{\mathsf{cr}} \rhd E : \mathsf{ok}}\ \text{T-E-CRITICAL-END-J}$$

$$\frac{\Gamma, MFWts, pcS \vdash^{\mathsf{atom}}_{vPol} E : \mathsf{ok}}{\Gamma, MFWts, pcS \vdash^{\mathsf{atom}}_{vPol} \mathsf{end}_{\mathsf{cr}} \rhd E : \mathsf{ok}}\ \text{T-E-CRITICAL-END-A}$$

E.7.3 Stack typing

$$\frac{}{\Psi, PDecl \vdash \cdot : \mathsf{ok}}\ \text{T-STACK-EMPTY} \qquad \frac{\Psi, PDecl \vdash F : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok}}{\Psi, PDecl \vdash F \rhd S : \mathsf{ok}}\ \text{T-STACK-FULL}$$

E.7.4 Recovery context typing

$$\frac{K = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad \Psi, PDecl \vdash S_k : \mathsf{ok}}{\Psi, PDecl \vdash K : \mathsf{ok}}\ \text{T-K}$$

E.7.5 Memory typing

$$\frac{\begin{array}{c} K = (\mathsf{N}_k, \mathsf{V}_k, S_k) \\ \text{if } \mathit{mdOf}(F) = \mathsf{jit} \text{ and } \mathit{iPidOf}(F) = \mathit{ipID}, \text{ then } \forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k) \ s.t.\ \mathsf{N}(x) = (v, \mathit{ipID}), \\ \text{then } \mathsf{N}_k(x) = (v, \mathit{ipID}) \\ \text{if } \mathit{mdOf}(F) = \mathsf{jit} \text{ and } \mathit{iPidOf}(F) = \mathit{ipID}, \text{ then } \forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k) \ s.t.\ \mathsf{V}(x) = (v, \mathit{ipID}), \\ \text{then } \mathsf{V}_k(x) = (v, \mathit{ipID}) \end{array}}{F, K \vdash \mathsf{N}, \mathsf{V} : \mathsf{ok}} \text{ T-MEM}$$

E.8 Preservation Proofs

Theorem 18 (Preservation for intermittent execution) *If* $\Psi \vdash \mathit{PDecl} : \mathsf{ok}$, $\Psi, \mathit{PDecl} \vdash \Sigma : \mathsf{ok}$, *and* $\Psi, \mathit{PDecl} \vdash \Sigma \Longrightarrow \Sigma'$ $(\exists \Sigma')$, *then* $\Psi, \mathit{PDecl} \vdash \Sigma' : \mathsf{ok}$.

Proof: The proof is by cases on $\Psi, \mathit{PDecl} \vdash \Sigma \Longrightarrow \Sigma'$.

Case StartAtom-I. Let $\Sigma = (I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash \mathit{PDecl} : \mathsf{ok}$, $\Psi, \mathit{PDecl} \vdash \Sigma : \mathsf{ok}$, and the last step is StartAtom-I.

$$\frac{\begin{array}{c} F = (\mathit{pID}, \mathit{rID}, \mathit{ID}, \mathit{version}, \overrightarrow{pr}, \mathit{pcS}, \mathit{vPol}, \upsilon, E, \mathsf{jit}, \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \mathit{aPol}); c; \mathsf{end}_{\mathsf{atom}}) \\ \mathit{aPol} = (\omega, \mu, \beta, \rho, \mathit{Inits}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad K' = (N_k \leftsquigarrow N{\downarrow_\omega}, V_k, F_0 \rhd S_k, \emptyset) \\ \mathit{ipidOf}(F) = \mathit{ipID} \qquad F_0 = (\mathit{pID}, \mathit{rID}, \mathit{ID}, \mathit{version}, \overrightarrow{pr}, \mathit{pcS}, \mathit{vPol}, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, 0, \mathit{aPol}, \emptyset, \emptyset), \mathit{initN}(\mathit{Inits}); c; \mathsf{end}_{\mathsf{atom}}) \end{array}}{\Psi, \mathit{PDecl} \vdash I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, \mathit{IRQ}, K', \mathsf{N}, \mathsf{V}, S, F_0, Q} \text{ StartAtom-I}$$

We need to show that $\Psi, \mathit{PDecl} \vdash I, \mathit{IRQ}, K', \mathsf{N}', \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, \mathit{PDecl} \vdash I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q : \mathsf{ok}$

$$\frac{\begin{array}{c} \Psi, \mathit{PDecl} \vdash F : \mathsf{ok} \qquad \Psi, \mathit{PDecl} \vdash S : \mathsf{ok} \qquad \Psi, \mathit{PDecl} \vdash K : \mathsf{ok} \\ \forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k).N(x) = N_k(x) \text{ and } \forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k).V(x) = V_k(x) \end{array}}{\Psi, \mathit{PDecl} \vdash (I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}} \text{ T-Config}$$

we learn that

(i) $\Psi, \mathit{PDecl} \vdash F : \mathsf{ok}$

(ii) $\Psi, \mathit{PDecl} \vdash S : \mathsf{ok}$

(iii) $\Psi, \mathit{PDecl} \vdash K : \mathsf{ok}$

(iv) $\forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k).N(x) = N_k(x)$ and $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k).V(x) = V_k(x)$ (since $\mathit{mdOf}(F) = \mathsf{jit}$)

By inversion of T-Frame-Jit-D on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \\ \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}}) \qquad \mathtt{sMd} = smodeOf(\mathsf{jit}) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \qquad \Gamma = \Psi(ID, version, \mathsf{jit}) \\ \Gamma, pc \vdash^{\mathsf{jit}} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \qquad \Gamma, pc \rhd pcS \vdash^{\mathsf{jit}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

we learn that

(v) $F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}})$

(vi) $\mathtt{sMd} = smodeOf(\mathsf{jit})$

(vii) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(viii) $\Gamma = \Psi(ID, version, \mathsf{jit})$

(ix) $\Gamma, pc \vdash^{\mathsf{jit}} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}$

(x) $\Gamma, pc \rhd pcS \vdash^{\mathsf{jit}} E : \mathsf{ok}$

By inversion of T-Atom-Begin-J on (ix),

$$\frac{\begin{array}{l} \omega \cap \rho = \emptyset \qquad \rho \cap \mu = \emptyset \\ \omega \cap \mu = \emptyset \qquad \forall\, x_k \in \omega \; s.t. \; \Gamma(x_k) = t_k @ \langle q_c^k, q_l^k, q_h^k \rangle \,.\, q_l^k \sqcup q_h^k \neq \mathsf{Nid} \\ \forall\, x_w \in \mu \; s.t. \; \Gamma(x_w) = t_w @ \langle q_c^w, q_l^w, q_h^w \rangle \,.\, q_h^w \neq \mathsf{Nid} \\ \Gamma' = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \qquad \Gamma', \cdot, pc \vdash^{\mathsf{atom}} c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \end{array}}{\Gamma, pc \vdash^{\mathsf{jit}} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}}\ \text{T-Atom-Begin-J}$$

we learn that

(xi) $\omega \cap \rho = \emptyset$

(xii) $\rho \cap \mu = \emptyset$

(xiii) $\omega \cap \mu = \emptyset$

(xiv) $\forall\, x_k \in \omega \; s.t. \; \Gamma(x_k) = t_k @ \langle q_c^k, q_l^k, q_h^k \rangle \,.\, q_l^k \sqcup q_h^k \neq \mathsf{Nid}$

(xv) $\forall\, x_w \in \mu \; s.t. \; \Gamma(x_w) = t_w @ \langle q_c^w, q_l^w, q_h^w \rangle \,.\, q_h^w \neq \mathsf{Nid}$

(xvi) $\Gamma' = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(xvii) $\Gamma', \cdot, pc \vdash^{\mathsf{atom}} c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}$

By application of T-Frame-A2J to the definition of F', (vii), (viii), (xvi), (xvii), (viii), and (x), we have

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, 0, \omega, \mu, \rho, Inits, \emptyset, \emptyset), initN(Inits); c; \mathsf{end}_{\mathsf{atom}}) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, 0, \omega, \mu, \rho, Inits, \emptyset, \emptyset) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma' = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma', \cdot, pc \vdash^{\mathsf{atom}} c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \qquad \Gamma, pc \rhd pcS \vdash^{\mathsf{jit}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-A2J}$$

By inversion of T-K on (iii), we know that $\Psi, PDecl \vdash S_k : \mathsf{ok}$ (*xviii*). By application of T-Stack-full to (xviii) and $\Psi, PDecl \vdash F' : \mathsf{ok}$, we have that

$$\Psi, PDecl \vdash K' : \mathsf{ok}\ (xix)$$

where $K' = F' \rhd S_k$.

Then, by application of T-Config to $\Psi, PDecl \vdash F' : \mathsf{ok}$, (ii), and (xix), we learn that

$$\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

as desired.

Case EndAtom-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is EndAtom-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw), \mathsf{end}_{\mathsf{atom}}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad F_k = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw), c) \qquad V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \\ N'_k = N_k \leftsquigarrow N{\downarrow_{NVw}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, F_J(pID) \rhd S_k) \qquad \mathsf{N}, \mathsf{V}, \upsilon \vdash res \Downarrow v_r \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{endAtom}(pID, \mathsf{aID}, rb, v_r)}{\Longrightarrow} I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q}\ \text{EndAtom-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q : \mathsf{ok}$,

$$\frac{\Psi, PDecl \vdash F : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}}\ \text{T-Config}$$

we learn that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By inversion of T-K on (iii), and since $K = (\mathsf{N}_k, \mathsf{V}_k, F_k \rhd S_k)$,

$$\frac{K = (\mathsf{N}_k, \mathsf{V}_k, F_k \rhd S_k) \qquad \Psi, PDecl \vdash F_k \rhd S_k : \mathsf{ok}}{\Psi, PDecl \vdash K : \mathsf{ok}}\ \text{T-K}$$

we learn that $\Psi, PDecl \vdash F_k \rhd S_k : \mathsf{ok}$ (iv). By inversion of T-Stack-full on (iv),

$$\frac{\Psi, PDecl \vdash S_k : \mathsf{ok} \qquad \Psi, PDecl \vdash F_k : \mathsf{ok}}{\Psi, PDecl \vdash F_k \rhd S_k : \mathsf{ok}}\ \text{T-Stack-full}$$

we learn that

(v) $\Psi, PDecl \vdash S_k : \mathsf{ok}$

(vi) $\Psi, PDecl \vdash F_k : \mathsf{ok}$

By application of T-Stack-full to (v) and by T-Frame-Jit-P, we learn that

$$\Psi, PDecl \vdash F_J(pID) \rhd S_k : \mathsf{ok}\ (vii).$$

By application of T-K to (vii) and $K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, F_J(pID) \rhd S_k)$, we have that

$$\Psi, PDecl \vdash K' : \mathsf{ok}\ (viii)$$

Now, we need to show that $PDecl \vdash F' : \mathsf{ok}$. By inversion of T-Frame-A2J on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{end}_{\mathsf{atom}}) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \qquad \Gamma, pc \rhd pcS \vdash^{\mathsf{jit}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-A2J}$$

we learn that

(ix) $F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{end}_{\mathsf{atom}})$

(x) $\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$

(xi) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(xii) $\Gamma = \Psi(ID, version, \mathsf{jit})$

(xiii) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(xiv) $\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}$

(xv) $\Gamma, pc \rhd pcS \vdash^{\mathsf{jit}} E : \mathsf{ok}$

By T-Skip, we have that $\Gamma, pc \vdash^{\mathsf{jit}} \mathsf{skip} \Rightarrow \Gamma'$ (xvi).

By application of T-Frame-Jit-D to the definition of F', the definition of $\mathtt{sMd}$ (indicating that $\mathtt{sMd} = \mathsf{jit}$), (xi), (xvi), and (xv)

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip}) \\ \mathtt{sMd} = smodeOf(\mathsf{jit}) \qquad \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathtt{sMd}) \qquad \Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok} \qquad \Gamma, pc \rhd pcS \vdash^{\mathsf{jit}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

we learn that $\Psi, PDecl \vdash F' : \mathsf{ok}$ (xvii).

Now, we additionally must show that the invariants for jit hold for the stepped configuration in order to show that it is well-formed. In particular, since $mdOf(F') = \mathsf{jit}$, we must show that

(xviii) if $pidOf(F) = pID$, then $S'_k = F_j(pID) \rhd S_k$ (where $K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S'_k)$)

(xix) if $iPidOf(F) = ipID$, then $\forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}'_k)$ $s.t.$ $\mathsf{N}(x) = (v, ipID)$, then $N'_k{\downarrow_{ckVarOf(S_k)}}(x) = (v, ipID)$

(xx) if $iPidOf(F) = ipID$, then $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}'_k)$ $s.t.$ $\mathsf{V}(x) = (v, ipID)$, then $\mathsf{V}'_k(x) = (v, ipID)$

Note that (xviii) holds trivially by the definition of $K' = (\ldots, F_J(pID) \rhd S_k)$ and $pidOf(F) = pID$.

Now, we must prove that (xix) holds.

Let $x \in \mathsf{dom}(N) \cap \mathsf{dom}(N_k)$ be arbitrary.

If x was not written during the region's execution, i.e., $x \notin NVw$, then the invariant for x follows by the well-formedness of the inital configuration.

Otherwise, x was written during the region's execution, and hence $x \in \mathsf{dom}(N) \cap NVw$. Therefore, the desired result follows by definition of $N'_k = N_k \leftsquigarrow N{\downarrow_{NVw}}$.

Proving (xx) follows analogous reasoning, relying on definition $V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}}$.

Applying T-Config to (xviii), (xix), (xx), (xvii), (viii), and (ii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

Case N-Assign-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is N-Assign-I.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_0) \in PDecl \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{N}) \qquad K = (N_k, V_k, S_k) \qquad \Psi(ID_c, version_c, \mathtt{Md}_c) = \Gamma \\ \mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \Gamma(x).1 = \mathsf{Id}(\_) \\ \mathtt{Md}_c & \text{otherwise} \end{cases} \qquad K' = (N'_k, V_k, S_k) \\ N'_k = \begin{cases} N_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ N_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], \mathsf{V}, S, F', Q} \text{ N-Assign-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID_c)], \mathsf{V}, S, F', Q : \mathsf{ok}$.
By inversion of T-Config on $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q : \mathsf{ok}$,

$$\frac{\begin{array}{c} \Psi, PDecl \vdash F : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok} \\ \text{if } mdOf(F) = \mathsf{jit} \text{ then } \forall x_0 \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k).N(x_0) = N_k(x_0) \\ \text{if } mdOf(F) = \mathsf{jit} \text{ then } \forall x_0 \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k).V(x_0) = V_k(x_0) \\ \text{if } mdOf(F) = \mathsf{jit} \text{ and } pidOf(F) = pID, \text{ then } S_k = F_j(pID) \rhd S^0_k \end{array}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}} \text{ T-Config}$$

we learn that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

(iv) if $mdOf(F) = \mathsf{jit}$ then $\forall x_0 \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k).N(x_0) = N_k(x_0)$

(v) if $mdOf(F) = \mathsf{jit}$ then $\forall x_0 \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k).V(x_0) = V_k(x_0)$

(vi) if $mdOf(F) = \mathsf{jit}$ and $pidOf(F) = pID$, then $S_k = F_j(pID) \rhd S^0_k$

By inversion of T-K on (iii), and given $K = (\mathsf{N}_k, \mathsf{V}_k, S_k)$,

$$\frac{K = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad \Psi, PDecl \vdash S_k : \mathsf{ok}}{\Psi, PDecl \vdash K : \mathsf{ok}} \text{ T-K}$$

we learn that $\Psi, PDecl \vdash S_k : \mathsf{ok}$.
By application of T-K on $\Psi, PDecl \vdash S_k : \mathsf{ok}$, and given that $K' = (N'_k, V_k, S_k)$, we learn that $\Psi, PDecl \vdash K' : \mathsf{ok}$.
Now, we need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$. We have two cases, each depending on $\mathtt{Md}_c$:

Subcase 1: $\mathtt{Md}_c \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{next}\}$**.** By inversion of T-Frame-Jit-D on (i),

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \\ \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \mathsf{task}(ID_c, version_c, pr, vPol_c, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID_c, version_c, \mathtt{sMd}) \qquad \Gamma, pc \vdash^{\mathtt{sMd}} x := e : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} E_c : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-Jit-D}$$

we learn that

(vii) $F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e)$

(viii) $\mathsf{task}(ID_c, version_c, pr, vPol_c, rPol, \lambda x.c_0) \in PDecl$

(ix) $\Gamma, pc \vdash^{\mathtt{sMd}} x := e : \mathsf{ok}$

(x) $\Gamma, pcS \vdash^{\mathtt{sMd}} E_c : \mathsf{ok}$

By T-Skip-Jit-Discard, we have that $\Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok}$ (xi).

Then, by application of T-Frame-Jit-D to (viii), (xi), (x), and the definition of F':

$$\frac{\begin{array}{c} F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \\ \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \mathsf{task}(ID_c, version_c, pr, vPol_c, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID_c, version_c, \mathtt{sMd}) \qquad \Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} E_c : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok} \quad (xii)} \text{ T-Frame-Jit-D}$$

In the case where $\mathtt{Md}_c = \mathsf{jit}$, we additionally must prove that $\forall x \in \mathsf{dom}(\mathsf{N}[x \mapsto (v, ipID)]) \cap \mathsf{dom}(\mathsf{N}'_k).N[x \mapsto (v, ipID)](x) = N'_k(x)$ and $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k).V(x) = V_k(x)$. The latter comes directly from (iv). The former is established by a combination of (v) and the definition of N'_k in jit mode: $N'_k = N_k[x \mapsto (v, ipID)]$ $(xiii)$.

Lastly, we must show that if $pidOf(F) = pID$, then $S_k = F_j(pID) \triangleright S^0_k$. This condition comes directly from (vi) and the fact that $\mathtt{Md}'_c = \mathsf{jit}$.

Then it follows by application of T-Config to (xii), (xiii), (ii), and $\Psi, PDecl \vdash K' : \mathsf{ok}$, we learn that

$$\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID_c)], \mathsf{V}, S, F', Q : \mathsf{ok}$$

as desired.

Subcase 2: $\mathtt{Md}_c = \mathsf{atom}(...)$**.** By inversion of T-Frame-Atomic on (i),

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathtt{Md}_c = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID_c, version_c, pr, vPol_c, rPol, \lambda x.c_0) \in PDecl \qquad \Gamma = \Psi(ID, version, \mathsf{jit}) \\ \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho) \qquad \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} x := e \Rightarrow MFWts' \\ \Gamma_0, MFWts', pc \rhd pcS \vdash^{\mathsf{atom}} E_c \Rightarrow MFWts'' \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Atomic}$$

we learn that

(vii) $F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e)$

(viii) $\mathsf{task}(ID_c, version_c, pr, vPol_c, rPol, \lambda x.c_0) \in PDecl$

(ix) $\Gamma = \Psi(ID, version, \mathsf{jit})$

(x) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho)$

(xi) $\Gamma_0, MFWts, pc \vdash^{\mathsf{atom}} x := e \Rightarrow MFWts'$

(xii) $\Gamma_0, MFWts', pc \rhd pcS \vdash^{\mathsf{atom}} E_c \Rightarrow MFWts''$

(xiii) $\mathtt{Md}_c = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$

By inversion of T-Assign-N-A on (xi),

$$\frac{\begin{array}{c} \Gamma_0(x) = t@q_x \\ q_x = \langle qID_c, qID_l, qID_h \rangle \qquad qID_c \neq \mathsf{Id}(\mathtt{RD}) \qquad \Gamma_0 \vdash e : t@q_e, \Lambda_e \\ q_e \sqcup pc \sqsubseteq qID_c \qquad MFWts' = \begin{cases} MFWts & \text{if } q_x = \mathsf{Nid} \\ MFWts, x & \text{if } q_x = \mathsf{Id}(\_) \end{cases} \end{array}}{\Gamma_0, MFWts, pc \vdash^{\mathsf{atom}} x := e \Rightarrow MFWts'}\ \text{T-Assign-N-A}$$

we learn that

(xiv) $\Gamma_0(x) = t@q_x$

(xv) $q_x = \langle qID_c, qID_l, qID_h \rangle$

(xvi) $qID_c \neq \mathsf{Id}(\mathtt{RD})$

(xvii) $\Gamma_0 \vdash e : t@q_e, \Lambda_e$

(xviii) $q_e \sqcup pc \sqsubseteq qID_c$

(xix) $MFWts' = \begin{cases} MFWts & \text{if } q_x = \mathsf{Nid} \\ MFWts, x & \text{if } q_x = \mathsf{Id}(\_) \end{cases}$

By T-Skip, we have that $\Gamma, pc \vdash^{\mathsf{atom}} \mathsf{skip} : \mathsf{ok}$ (xx).

By application of T-Frame-Atomic to the definition of F', (xx), (xiii), (xii), (x), (viii), and (ix),

$$\frac{\begin{array}{c} F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ \mathsf{task}(ID_c, version_c, pr, vPol_c, rPol, \lambda x.c_0) \in PDecl \qquad \Gamma = \Psi(ID, version, \mathsf{jit}) \\ \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho) \qquad \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts' \\ \Gamma_0, MFWts', pc \rhd pcS \vdash^{\mathsf{atom}} E_c \Rightarrow MFWts'' \\ \mathtt{Md}'_c = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw \cup \{x\}, Vw) \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \textsc{T-Frame-Atomic}$$

Then by application of T-Config to $\Psi, PDecl \vdash F' : \mathsf{ok}$, (ii), and (v), we learn that

$$\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID_c)], \mathsf{V}, S, F', Q : \mathsf{ok}$$

as desired.

Case V-Assign-I. This case is analogous to the one above, but uses the rule V-Assign-I.

Case TriggerInt-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is TriggerInt-I.

$$\frac{\begin{array}{c} IRQ = \mathsf{interrupt}(inID, iID, v) :: IRQ' \\ F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \\ \mathsf{isr}(iID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \qquad pr > prOf(\overrightarrow{pr}_c) \\ fresh(pID) \qquad (\mathtt{Md}_n, K') = schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N}) \\ ipID = iPidOf(pID, \mathtt{Md}_n) \\ F_n = (pID, inID, iID, 0, pr, \mathsf{Id}, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol_c.Inits); c_i; ret) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{trigger}(inID, pID, \mathtt{Md}_n)}{\Longrightarrow} I, IRQ', K', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q}\ \textsc{TriggerInt-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ', K', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q : \mathsf{ok}$.
By inversion of T-Config on $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q : \mathsf{ok}$,

$$\frac{\Psi, PDecl \vdash F : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}}\ \textsc{T-Config}$$

we learn that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By application of T-Stack-full on (i) and (ii), we have

$$\frac{\Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash F : \mathsf{ok}}{\Psi, PDecl \vdash F \rhd S : \mathsf{ok}\ (iv)}\ \text{T-Stack-full}$$

We now need to show that $\Psi, PDecl \vdash F_n : \mathsf{ok}$.

By $\cdot \vdash PDecl : \mathsf{ok}$, it follows by inversion of T-Task that $Inits \cap ShCP = \emptyset$ and $\Gamma, \mathsf{ld} \vdash^{\mathtt{Md}_n} c_i : \mathsf{ok}\ (iv)$ where $\Gamma = \Psi(ID, version, \mathtt{Md}_n)$.

By the rule T-E-md, we know that $\Gamma, \mathsf{ld} \vdash^{\mathtt{Md}_n}_{vPol} \cdot : \mathsf{ok}\ (v)$.

By definition of *schdFrame*, we know that $\mathtt{Md}_n \neq \mathsf{atom}$. Thus, the rule T-Frame-Jit-D applies. By a combination of sequencing for commands and T-Frame-Jit-D applied to (iv) and (v), we learn that:

$$\frac{\begin{array}{c} F_n = (pID, inID, iID, 0, pr, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol_c.Inits); c_i; ret) \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \\ \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \mathsf{isr}(iID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \\ \Gamma, \mathsf{ld} \vdash^{\mathtt{sMd}} initN(vPol_c.Inits); c_i; ret : \mathsf{ok} \qquad \Gamma, \mathsf{ld} \vdash^{\mathtt{sMd}} \cdot : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F_n : \mathsf{ok}\ (vi)}\ \text{T-Frame-Jit-D}$$

Lastly, we need to show that $\Psi, PDecl \vdash K' : \mathsf{ok}$.

By Lemma 44 applied to $\Psi, PDecl \vdash K : \mathsf{ok}$, we have that

$$\Psi, PDecl \vdash K' : \mathsf{ok}\ (vii)$$

In the case that $\mathtt{Md}_n = \mathsf{jit}$, we must further establish that

(viii) if $mdOf(F_n) = \mathsf{jit}$ and $iPidOf(F_n) = ipID$, then $\forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k)$ *s.t.* $\mathsf{N}(x) = (v, ipID)$, then $\mathsf{N}'_k(x) = (v, ipID)$

(ix) if $mdOf(F_n) = \mathsf{jit}$ and $iPidOf(F_n) = ipID$, then $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k)$ *s.t.* $\mathsf{V}(x) = (v, ipID)$, then $\mathsf{V}'_k(x) = (v, ipID)$

(x) if $mdOf(F_n) = \mathsf{jit}$ and $pidOf(F_n) = pID$, then $S_k = F_j(pID) \rhd S'_k$

Note that (x) holds by definition of *schdFrame*. Next, (viii) and (ix) hold vacuously since *pID* is fresh in this rule, and hence the process has not yet written any values to checkpointed memory. The only nonvolatile variables written by this rule are *Inits*, but $Inits \cap ShCP = \emptyset$.

By application of T-Config to (vii), (vi), and (iv), we obtain

$$\frac{\Psi, PDecl \vdash F_n : \mathsf{ok} \qquad \Psi, PDecl \vdash F \rhd S : \mathsf{ok} \qquad \Psi, PDecl \vdash K' : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q) : \mathsf{ok}}\ \text{T-Config}$$

as desired.

Case PostEvent-I. The proof of this case is similar to the one above, but instead considers events.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, v)) \\ \mathsf{eh}(eID, 0, pr, vPol, rPol, \lambda x.c_e) \in PDecl \qquad pr > prOf(\overrightarrow{pr}_c) \qquad fresh(pID) \\ fresh(reID) \qquad F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \\ (\mathtt{Md}_n, K') = schdFrame(PDecl, (pID, eID, v, rPol), K, \mathsf{N}) \\ ipID = iPidOf(pID, \mathtt{Md}_n) \\ F_n = (pID, reID, eID, 0, pr, \mathsf{ld}, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol.Inits); c_e; ret) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{trigger}(reID, pID, \mathtt{Md}_n)}{\Longrightarrow} I, IRQ, K', \mathsf{N}, \mathsf{V}, F' \rhd S, F_n, Q} \textsc{PostEvent-I}$$

Case DelayInt-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is DelayInt-I.

$$\frac{\begin{array}{c} IRQ = irq :: IRQ' \qquad irq = \mathsf{interrupt}(inID, iID, v) \\ pr \leq prOf(\overrightarrow{pr}_c) \qquad \mathsf{isr}(iID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \\ F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{delay}(inID)}{\Longrightarrow} I, IRQ', K, \mathsf{N}, \mathsf{V}, S, F, enqueue(Q, irq)} \textsc{DelayInt-I}$$

We need to show that $PDecl \vdash I, IRQ', K, \mathsf{N}, \mathsf{V}, S, F, enqueue(Q, irq) : \mathsf{ok}$.
By inversion of T-Config on $PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q : \mathsf{ok}$,

$$\frac{\Psi, PDecl \vdash F : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}} \textsc{T-Config}$$

we learn that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By application of T-Config to (i), (ii), and (iii), we learn that

$$\Psi, PDecl \vdash I, IRQ', K, \mathsf{N}, \mathsf{V}, S, F, enqueue(Q, irq) : \mathsf{ok}$$

as desired.

Case DelayEvent-I. This case is similar to the one above, but instead uses DelayEvent-I.

$$\frac{\begin{array}{c} F = (pID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{post}(eID, v)) \\ \mathsf{eh}(eID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \qquad pr \leq prOf(\overrightarrow{pr}_c) \\ fresh(reID) \qquad F' = (pID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip}) \\ Q' = enqueue(Q, \mathsf{event}(reID, eID, v)) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{delay}(reID)}{\Longrightarrow} I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q'} \text{ DelayEvent-I}$$

Case Ret-S-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is Ret-S-I.

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, ret(res)) \\ finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K' \\ S = F \rhd S' \qquad F = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \\ Q = \mathsf{tk}(rID_n, ID_n, v) :: Q' \qquad prOf(\overrightarrow{pr}_n) \geq pr_q \\ (\mathtt{Md}'_n, K'') = \begin{cases} schdFrame(PDecl, (pID_n, ID_n, v, rPol), K', \mathsf{N}) & \text{if } \mathtt{Md}_n = \mathsf{altOf}(rPol) \\ (\mathtt{Md}_n, K') & \text{else} \end{cases} \\ c'_n = \begin{cases} initN(vPol_n.Inits); c_n; ret(res') & \text{if } \mathtt{Md}_n = \mathsf{altOf}(rPol) \\ c_n & \text{else} \end{cases} \\ ipID = iPidOf(pID_n, \mathtt{Md}'_n) \qquad F' = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}'_n, c'_n) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \qquad o_1 = \mathsf{complete}(pID_c, v_r) \qquad o_2 = \mathsf{trigger}(rID_n, pID_n, \mathtt{Md}_n) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q \overset{o_1, o_2}{\Longrightarrow} I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q} \text{ Ret-S-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q : \mathsf{ok}$.

By inversion of T-Config on $PDecl \vdash \Sigma : \mathsf{ok}$,

$$\frac{\Psi, PDecl \vdash F_c : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q) : \mathsf{ok}} \text{ T-Config}$$

we learn that

(i) $\Psi, PDecl \vdash F_c : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By inversion of T-Stack-Full on (ii), we have

(iv) $\Psi, PDecl \vdash S' : \mathsf{ok}$

(v) $\Psi, PDecl \vdash F : \mathsf{ok}$

We now need to show that $\Psi, PDecl \vdash K'' : \mathsf{ok}$.

By Lemma 45 applied to $\Psi, PDecl \vdash K : \mathsf{ok}$, we have that $\Psi, PDecl \vdash K' : \mathsf{ok}$.
By Lemma 44 applied to $\Psi, PDecl \vdash K' : \mathsf{ok}$, we have that

$$\Psi, PDecl \vdash K'' : \mathsf{ok}\ (vi)$$

Lastly, we need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$.
By definition of *schdFrame*, we know that $\mathtt{Md}'_n \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{next}\}$. In all cases, it follows by inversion of T-Frame-Jit-D on (v)

$$\frac{\begin{array}{c} F = (pID_n, rID_n, ID_n, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \\ \mathsf{task}(ID_n, 0, pr_n, vPol_n, rPol_n, \lambda x.c'_n) \in PDecl \\ \Gamma, pc \vdash^{\mathsf{jit}}_{pr_n} c_n : \mathsf{ok} \qquad \Gamma, pc \rhd pcS_n \vdash^{\mathsf{jit}}_{pr_n} E_n : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

that the following hold for $\Gamma = \Psi(ID_n, 0, \mathtt{Md}_n)$:

(vii) $\Gamma, pc \vdash^{\mathsf{jit}}_{pr_n} c_n : \mathsf{ok}$

(viii) $\Gamma, pc \rhd pcS_n \vdash^{\mathsf{jit}}_{pr_n} E_n : \mathsf{ok}$

By application of T-Frame-Jit-D to the definition of F', (*vii*), (*viii*), and sequencing for commands, we have

$$\Psi, PDecl \vdash F' : \mathsf{ok}\ (ix)$$

Note that in the case where $\mathtt{Md}'_n = \mathsf{jit}$, the jit invariants are established due to similar reasoning to the PostInt-I case.
Thus, by application of T-Config to (iv), (vi) and (ix), we have that

$$\Psi, PDecl \vdash I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q : \mathsf{ok}$$

as desired.

Case Ret-Q-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is Ret-Q-I.

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, ret(res)) \\ finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K' \qquad S = F \rhd S' \\ F = (pID_n, rID_n, ID_n, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \qquad Q = \mathsf{tk}(rID_q, ID_q, v) :: Q' \\ \mathsf{task}(ID_q, 0, pr_q, pcS_q, vPol_q, rPol_q, \lambda x.c_q) \in PDecl \qquad pr_q > prOf(\overrightarrow{pr}_n) \qquad fresh(pID_q) \\ fresh(rID_q) \qquad schdFrame(PDecl, (pID_q, ID_q, v, rPol_q), K', \mathsf{N}) = (\mathtt{Md}_q, K'') \\ ipID = iPidOf(pID_q, \mathtt{Md}_q) \\ F' = (pID_q, rID_q, ID_q, 0, pr_q, pcS_q, vPol_q, x \mapsto v, \cdot, \mathtt{Md}_q, initN(vPol_q.Inits); c_q; ret) \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v_r \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q \overset{\substack{\mathsf{complete}(pID_c, v_r) \\ \mathsf{trigger}(rID_n, pID_n, \mathtt{Md}_n)}}{\Longrightarrow} I, IRQ, K'', \mathsf{N}, \mathsf{V}, S, F', Q'} \text{ Ret-Q-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K'', \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.
By inversion of T-Config on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$,

$$\frac{\Psi, PDecl \vdash F_c : \mathsf{ok} \qquad \Psi, PDecl \vdash S : \mathsf{ok} \qquad \Psi, PDecl \vdash K : \mathsf{ok}}{\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q) : \mathsf{ok}} \text{ T-Config}$$

we learn that

(i) $\Psi, PDecl \vdash F_c : \mathsf{ok}$
(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$
(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

We need to show that $\Psi, PDecl \vdash K'' : \mathsf{ok}$.
By Lemma 45 applied to $\Psi, PDecl \vdash K : \mathsf{ok}$, we have that $\Psi, PDecl \vdash K' : \mathsf{ok}$.
By Lemma 44 applied to $\Psi, PDecl \vdash K' : \mathsf{ok}$, we have that

$$\Psi, PDecl \vdash K'' : \mathsf{ok} \ (iv)$$

Lastly, we need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$.
By definition of *schdFrame*, where $\mathtt{sMd} = smodeOf(\mathtt{Md}_q)$, we know that

$$\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \ (v)$$

By $\cdot \vdash PDecl : \mathsf{ok}$, we know that

$$\Gamma, \mathsf{Id} \vdash^{\mathtt{sMd}}_{vPol_q} c_q : \mathsf{ok} \ (vi)$$

where $\Gamma = \Psi(ID_q, 0, \mathtt{sMd})$. By T-E-md, we know that

$$\Gamma, \mathsf{Id} \vdash^{\mathtt{sMd}}_{vPol_q} \cdot : \mathsf{ok} \ (vii)$$

By application of the rule T-Frame-Jit-D to (*vi*), (*vii*), and (*v*), we have

$$\frac{\begin{array}{c} F' = (pID_q, rID_q, ID_q,\ pr_q, pcS_q, vPol_q, x \mapsto v, \cdot, \mathtt{Md}_q, initN(vPol_q.Inits); c_q; ret) \\ \mathtt{sMd} = smodeOf(\mathtt{Md}_q) \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \\ \mathsf{task}(ID_q, 0, pr_q, \mathsf{ld}, vPol_q, rPol_q, \lambda x.c_q) \in PDecl \qquad \Gamma = \Psi(ID_q, 0, \mathtt{sMd}) \\ \Gamma, \mathsf{ld} \vdash^{\mathtt{sMd}} initN(vPol_q.Inits); c_q; ret : \mathsf{ok} \qquad \Gamma, \mathsf{ld} \vdash^{\mathtt{sMd}} \cdot : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}\ (viii)}\ \text{T-Frame-Jit-D}$$

Note that in the case where $\mathtt{Md}_q = \mathsf{jit}$, the jit invariants are established as in the previous case.

Thus, by application of T-Config to (ii), (iv) and (viii), we have that

$$\Psi, PDecl \vdash I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q : \mathsf{ok}$$

as desired.

Case StartAtom-DA. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is StartAtom-DA.

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}}) \\ \mathtt{Md} \in \{\mathsf{discard}, \mathsf{next}(pID_a)\} \qquad F' = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \text{StartAtom-DA}$$

We want to show that $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, PDecl \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}$, we have that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By inversion of T-Frame-Jit-D on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}}) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad \Gamma = \Psi(ID, version, \mathtt{sMd}) \\ \Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok} \\ \Gamma, pc \rhd pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok} \end{array}}{PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

we learn that

(iv) $\Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, \mathit{Inits}); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}$

(v) $\Gamma, pc \rhd pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok}$

By inversion of T-Start-Atomic on (iv)

$$\frac{\Gamma, pc \vdash^{\mathtt{sMd}} c : \mathsf{ok} \qquad \mathtt{sMd} = \mathsf{discard}}{\Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, \mathit{Inits}); c; \mathsf{end}_{\mathsf{atom}} : \mathsf{ok}}\ \text{T-Atom-Begin-D}$$

we have that $\Gamma, pc \vdash^{\mathtt{sMd}} c : \mathsf{ok}$ (*vi*).

By application of T-Frame-Jit-D to (vi) and (v), we learn that

$$\Psi, \mathit{PDecl} \vdash F' : \mathsf{ok}\ (\mathit{vii})$$

By application of T-Config to (vii), (ii), and (iii), we obtain the desired result:

$$\Psi, \mathit{PDecl} \vdash I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

Case If-true-I. Let $\Sigma = (I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\cdot \vdash \mathit{PDecl} : \mathsf{ok}$, $\Psi, \mathit{PDecl} \vdash \Sigma : \mathsf{ok}$, and the last step is If-true-I.

$$\frac{\begin{array}{c} F = (\mathit{pID}, \mathit{rID}, \mathit{ID}, \mathit{version}, \overrightarrow{pr}, \mathit{pcS}, \mathit{vPol}, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ \mathit{pcS} = pc_0 \rhd \mathit{pcS}_0 \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow \mathsf{true} \qquad \mathsf{task}(\mathit{ID}, \mathit{version}, pr, \mathit{pcS}, \mathit{vPol}, \mathit{rPol}, \lambda x.c) \in \mathit{PDecl} \\ \Psi(\mathit{ID}, \mathit{version}, \mathtt{sMd}) = \Gamma \qquad \mathtt{sMd} = \mathit{smodeOf}(\mathtt{Md}) \qquad pc = (\sqcup_{x \in \Lambda} \Gamma(x)\downarrow_{\overline{q}}) \sqcup pc_0 \\ F' = (\mathit{pID}, \mathit{rID}, \mathit{ID}, \mathit{version}, \overrightarrow{pr}, pc \rhd \mathit{pcS}, \mathit{vPol}, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, c_1) \end{array}}{\Psi, \mathit{PDecl} \vdash I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \text{If-true-I}$$

We want to show that $\Psi, \mathit{PDecl} \vdash I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, \mathit{PDecl} \vdash (I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{ok}$, we have that

(i) $\Psi, \mathit{PDecl} \vdash F : \mathsf{ok}$

(ii) $\Psi, \mathit{PDecl} \vdash S : \mathsf{ok}$

(iii) $\Psi, \mathit{PDecl} \vdash K : \mathsf{ok}$

(iv) if $\mathit{mdOf}(F_n) = \mathsf{jit}$ and $\mathit{iPidOf}(F_n) = \mathit{ipID}$, then $\forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k)$ *s.t.* $\mathsf{N}(x) = (v, \mathit{ipID})$, then $\mathsf{N}'_k(x) = (v, \mathit{ipID})$ and $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k)$ *s.t.* $\mathsf{V}(x) = (v, \mathit{ipID})$, then $\mathsf{V}'_k(x) = (v, \mathit{ipID})$

(v) if $\mathit{mdOf}(F_n) = \mathsf{jit}$ and $\mathit{pidOf}(F_n) = \mathit{pID}$, then $S_k = F_j(\mathit{pID}) \rhd S'_k$

We need to show that $\Psi, \mathit{PDecl} \vdash F' : \mathsf{ok}$.

Let $\mathtt{sMd} = \mathit{smodeOf}(\mathtt{Md})$. The proof proceeds in cases on $\mathtt{sMd}$:

Case $\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}$**.** The rule T-Frame-Jit-D applies. By inversion of T-Frame-Jit-D on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ pcS = pc_0 \rhd pcS_0 \qquad \mathsf{task}(ID, v, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \Gamma = \Psi(ID, version, \mathtt{sMd}) \\ \Gamma, pc_0 \vdash^{\mathtt{sMd}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

we learn that

(vi) $\Gamma, pc_0 \vdash^{\mathtt{Md}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \mathsf{ok}$

(vii) $\Gamma', pcS \vdash^{\mathtt{Md}} E : \mathsf{ok}$

By inversion of T-If-DJ on (vi),

$$\frac{\begin{array}{c} \Gamma \vdash^{\mathtt{Md}} e : \mathtt{bool}@q_e, \Lambda_e \qquad \Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}} c_1 : \mathsf{ok} \qquad \Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}} c_2 : \mathsf{ok} \\ \Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}} \mathsf{end}_{\mathsf{if}} : \mathsf{ok} \qquad \Lambda = \Lambda_e \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \end{array}}{\Gamma, pc_0 \vdash^{\mathtt{sMd}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \mathsf{ok}}\ \text{T-If-DJ}$$

we learn that

(viii) $\Gamma \vdash^{\mathtt{sMd}} e : \mathtt{bool}@q_e, \Lambda_e$

(ix) $\Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}} c_1 : \mathsf{ok}$

(x) $\Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}} c_2 : \mathsf{ok}$

(xi) $\Gamma, pc_0 \sqcup q_e \vdash^{\mathtt{sMd}} \mathsf{end}_{\mathsf{if}} : \mathsf{ok}$

By Lemma 42 applied to (vi), we learn that $q_e = (\sqcup_{x \in \Lambda_e} \Gamma(x)\!\downarrow_{\overline{q}})$, and hence, $pc = pc_0 \sqcup q_e$.

By application of T-E-if-end-J on (vii) and (xi), we learn that

$$\Gamma, pc \rhd pcS \vdash^{\mathtt{sMd}} ([\,]; \mathsf{end}_{\mathsf{if}}) :: E : \mathsf{ok}\ (xii)$$

By application of T-Frame-Jit-D to the definition of F', (ix), and (xii), we learn the following:

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, ([\,]; \mathsf{end}_{\mathsf{if}}) :: E, \mathtt{Md}, c_1) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad \Gamma = \Psi(ID, version, \mathtt{sMd}) \qquad \Gamma, pc \vdash^{\mathtt{sMd}} c_1 \Rightarrow \Gamma_1 \\ \Gamma_1, pc \rhd pcS \vdash^{\mathtt{Md}} ([\,]; \mathsf{end}_{\mathsf{if}}) :: E \Rightarrow \Gamma'' \qquad \mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{next}(\_), \mathsf{altOf}(\_, \_)\} \end{array}}{PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

Additionally, note that if $\texttt{sMd} = \mathsf{jit}$, the jit invariants hold directly from (vi) and (vii).

Case $\texttt{sMd} = \mathsf{atom}$ The rule T-Frame-Atomic applies, and by inversion on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, v, E, \texttt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ \mathsf{task}(ID, v, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \Gamma = \Psi(ID, v, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho) \\ pcS = pc_0 \rhd pcS_0 \qquad \Gamma_0, NVw, pc_0 \vdash^{\mathsf{atom}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \Rightarrow MFWts \\ \Gamma_0, MFWts, pcS \vdash^{\mathsf{atom}} E : \mathsf{ok} \qquad \texttt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Atomic}$$

we learn that

(vi) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho)$

(vii) $\Gamma_0, MFWts, pc_0 \vdash^{\mathsf{atom}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \Rightarrow MFWts'$

(viii) $\Gamma_0, MFWts', pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}$

The proof proceeds by cases on pc:

Subcase $pc_0 = \mathsf{Nid}$. The rule T-If-Nid-A applies. By inversion on (vi),

$$\frac{\begin{array}{c} \Gamma_0, MFWts \vdash^{\mathsf{atom}} e : \texttt{bool}@q_e, \Lambda_e \qquad \Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}} c_1 \Rightarrow MFWts' \\ \Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}} c_2 \Rightarrow MFWts' \qquad \Lambda = \Lambda_e \end{array}}{\Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \Rightarrow MFWts'}\ \text{T-If-Nid-A}$$

we have that

(ix) $\Gamma_0, MFWts \vdash^{\mathsf{atom}} e : \texttt{bool}@q_e, \Lambda_e$

(x) $\Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}}_{pr_{\mathsf{ceil}}} c_1 \Rightarrow MFWts'$

(xi) $\Gamma_0, MFWts, \mathsf{Nid} \vdash^{\mathsf{atom}}_{pr_{\mathsf{ceil}}} c_2 \Rightarrow MFWts'$

By application of T-E-if-end-A on (viii), and since $\mathsf{Nid} = pc_0 \sqcup q_e$ (from $pc_0 = \mathsf{Nid}$)

$$\frac{\Gamma_0, MFWts', pcS_0 \vdash^{\mathsf{atom}} E : \mathsf{ok}}{\Gamma_0, MFWts', \mathsf{Nid} \rhd pcS_0 \vdash^{\mathsf{atom}} \mathsf{end}_{\mathsf{if}} :: E : \mathsf{ok}\ (xviii)}\ \text{T-E-if-end-A}$$

By application of the rule T-Frame-Atomic to (xvii), (iii), (vi), (xiii), and (viii), we obtain that

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end_{if}} :: E, \mathtt{Md}, c_1) \\ \mathsf{task}(ID, v, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho) \\ pcS = pc_0 \rhd pcS_0 \qquad \Gamma_0, NVw, \mathsf{Nid} \vdash^{\mathsf{atom}} c_1 \Rightarrow MFWts' \\ \Gamma_0, MFWts', \mathsf{Nid} \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{if}} :: E : \mathsf{ok} \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-Atomic}$$

Subcase $pc_0 = \mathsf{ld}$. The rule T-If-Id-A applies. By inversion on (viii),

$$\frac{\begin{array}{c} \Gamma_0, MFWts \vdash^{\mathsf{atom}} e : \mathtt{bool}@q_e, \Lambda_e \qquad \Gamma_0, MFWts, \mathsf{ld} \sqcup q_e \vdash^{\mathsf{atom}} c_1 \Rightarrow MFWts_1 \\ \Gamma_0, MFWts, \mathsf{ld} \sqcup q_e \vdash^{\mathsf{atom}} c_2 \Rightarrow MFWts_2 \\ W = (MFWts_1 \cup MFWts_2) \setminus (MFWts_1 \cap MFWts_2) \\ \forall x \in W, \Gamma_0(x) = \mathsf{ld}(\mathtt{CK}) \qquad MFWts' = MFWts_1 \cap MFWts_2 \qquad \Lambda = \Lambda_e \end{array}}{\Gamma_0, MFWts, \mathsf{ld} \vdash^{\mathsf{atom}} \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \Rightarrow MFWts'}\ \text{T-If-Id-A}$$

we have that

(ix) $\Gamma_0, MFWts \vdash^{\mathsf{atom}} e : \mathtt{bool}@q_e, \Lambda_e$

(x) $\Gamma_0, MFWts, \mathsf{ld} \sqcup q_e \vdash^{\mathsf{atom}} c_1 \Rightarrow MFWts_1$

(xi) $\Gamma_0, MFWts, \mathsf{ld} \sqcup q_e \vdash^{\mathsf{atom}} c_2 \Rightarrow MFWts_2$

(xii) $W = (MFWts_1 \cup MFWts_2) \setminus (MFWts_1 \cap MFWts_2)$

(xiii) $\forall x \in W, \Gamma_0(x) = \mathsf{ld}(\mathtt{CK})$

(xiv) $MFWts' = MFWts_1 \cap MFWts_2$

By application of T-E-if-end-A on (viii), (xii) and $\mathsf{ld} = pc_0 \sqcup q_e$

$$\frac{\Gamma_0, MFWts', pcS \vdash^{\mathsf{atom}} E \Rightarrow MFWts''}{\Gamma_0, MFWts', \mathsf{ld} \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{if}} :: E \Rightarrow MFWts''\ (xv)}\ \text{T-E-if-end-A}$$

By application of the rule T-Frame-Atomic to (xv), (vi), (x), (viii), (xiv), and (xv), we obtain that

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, (\mathsf{end_{if}} :: E), \mathtt{Md}, c_1) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, \mathsf{ld} \vdash^{\mathsf{atom}} c_1 \Rightarrow MFWts' \qquad MFWts' = MFWts_1 \cap MFWts_2 \\ \Gamma_0, MFWts', \mathsf{ld} \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{if}} :: E \Rightarrow MFWts'' \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-Atomic}$$

In all cases, we have that $\Psi, PDecl \vdash F' : \mathsf{ok}$ $(*)$. By application of T-Config on (*), (ii), and (iii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

Case If-false-I. This case is symmetric to the one above. It follows the same structure, but where the typing judgments for c_2 are used.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ pcS = pc_0 \rhd pcS_0 \\ \mathsf{N}, \mathsf{V}, \upsilon_c \vdash e \Downarrow \mathsf{false} \qquad \mathsf{task}(ID, version, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \Psi(ID, version, \mathtt{sMd}) = \Gamma \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad pc = (\sqcup_{x \in \Lambda} \Gamma(x)\downarrow_{\overline{q}}) \sqcup pc_0 \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} \rhd E, \mathtt{Md}, c_2) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{If-false-I}$$

Case If-end-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\cdot \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is If-end-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} :: E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{If-end-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we learn that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

We need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$.

The proof proceeds in two cases based on Md:

Case $\mathtt{Md} = \mathsf{atom}$**.** Suppose that $\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$ for some aID, rb, ω, μ, ρ, $Inits$, NVw, Vw. By inversion of T-Frame-Atomic on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} :: E, \mathtt{Md}, \mathsf{skip}) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \qquad \Gamma = \Psi(ID, version, \mathsf{jit}) \\ \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \qquad \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts' \\ \Gamma_0, MFWts', pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end}_{\mathsf{if}} :: E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \textsc{T-Frame-Atomic}$$

we have

(iv) $F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end_{if}} :: E, \mathtt{Md}, \mathsf{skip})$

(v) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(vi) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(vii) $\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts'$

(viii) $\Gamma_0, MFWts', pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{if}} :: E : \mathsf{ok}$

By the rule T-Skip-A, we learn that

$$\Gamma_0, NVw, pc_0 \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts' \ (ix)$$

where $pcS = pc_0 \rhd pcS_0$.

By inversion of T-E-if-end-A on (viii),

$$\frac{\Gamma_0, MFWts', pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}}{\Gamma_0, MFWts', pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{if}} :: E : \mathsf{ok}} \text{ T-E-if-end-A}$$

we learn that

$$\Gamma_0, MFWts', pcS \vdash^{\mathsf{atom}} E : \mathsf{ok} \ (xiii)$$

By application of T-Frame-Atomic on the definition of F', (v), (vi), (ix), (viii), we obtain

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \qquad \Gamma = \Psi(ID, version, \mathsf{jit}) \\ \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \qquad \Gamma_0, NVw, pc_0 \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts \\ \Gamma_0, MFWts, pcS \vdash^{\mathsf{atom}} E : \mathsf{ok} \qquad \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}} \text{ T-Frame-Atomic}$$

as desired.

Case $\mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{alternative}\}$**.** By inversion of T-Frame-Jit-D on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end_{if}} :: E, \mathtt{Md}, \mathsf{skip}) \\ \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \qquad \Gamma = \Psi(ID, version, \mathtt{sMd}) \\ \Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok} \qquad \Gamma', pc \rhd pcS \vdash^{\mathtt{sMd}} \mathsf{end_{if}} :: E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-Jit-D}$$

we learn that

(iv) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(v) $\Gamma, pc \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok}$

(vi) $\Gamma, pc \rhd pcS \vdash^{\mathtt{sMd}} \mathsf{end}_{\mathsf{if}} :: E : \mathsf{ok}$

By application of T-Skip-S, we have

$$\Gamma, pc_0 \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok}\ (vii)$$

where $pcS = pc_0 \rhd pcS_0$.

By inversion of T-E-if-end-J on (vi),

$$\frac{\Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok} \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}}{\Gamma, pc \rhd pcS \vdash^{\mathtt{sMd}} \mathsf{end}_{\mathsf{if}} :: E : \mathsf{ok}}\ \text{T-E-if-end-J}$$

we learn that

$$\Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok}\ (viii)$$

By application of T-Frame-Jit-D on the definition of F', (iv), (viii), (v), we obtain the desired result:

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \\ \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathtt{sMd}) \qquad \Gamma, pc_0 \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

In both cases, we have that $\Psi, PDecl \vdash F' : \mathsf{ok}$ (ix). By application of T-Config to (ix), (ii), and (iii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

Case Let-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\cdot \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is Let-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{let}\ x = e\ \mathsf{in}\ c) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon[x \mapsto v], E, \mathtt{Md}, c) \\ \mathsf{N}, \mathsf{V}, \upsilon \vdash e \Downarrow v \qquad x\ fresh \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \text{Let-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \mathsf{V}', S, F', Q : \mathsf{ok}$.

By inversion of T-Config, we have that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By inversion of T-K on (iii), we learn that $\Psi, PDecl \vdash S_k : \mathsf{ok}$ (*iv*).
By application of T-K to (iv) and $K' = (\mathsf{N}_k, \mathsf{V}'_k, S_k)$, we have that

$$\Psi, PDecl \vdash K' : \mathsf{ok}\ (v)$$

Now, we need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$. For this, the proof proceeds in two cases depending on $\mathtt{Md}$:

Case $\mathtt{Md} = \mathsf{atom}$**.** Suppose that $\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$ for some aID, rb, ω, μ, ρ, $Inits$, NVw, and Vw. By inversion of T-Frame-Atomic on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, v, E, \mathtt{Md}, \mathsf{let}\ x = e\ \mathsf{in}\ c) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \qquad \Gamma = \Psi(ID, version, \mathsf{jit}) \\ \mathsf{sMd} = smodeOf(\mathtt{Md}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{sMd}} \mathsf{let}\ x = e\ \mathsf{in}\ c \Rightarrow MFWts' \\ \Gamma_0, MFWts', pcS \vdash^{\mathsf{sMd}} E : \mathsf{ok} \qquad \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Atomic}$$

we learn that

(vi) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(vii) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(viii) $\Gamma_0, NVw, pc \vdash^{\mathsf{sMd}} \mathsf{let}\ x = e\ \mathsf{in}\ c \Rightarrow MFWts'$

(ix) $\Gamma_0, MFWts', pcS \vdash^{\mathsf{sMd}} E : \mathsf{ok}$

The proof proceeds in cases on e:

Subcase $e \neq \mathsf{IN}()$**.** The rule T-let-e-A applies. By inversion on (viii),

$$\frac{\Gamma_0, NVw \vdash^{\mathsf{atom}} e : t@q, \Lambda_e \qquad (\Gamma_0, x : t@q), MFWts, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'}{\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{let}\ x = e\ \mathsf{in}\ c \Rightarrow MFWts'}\ \text{T-let-e-A}$$

we have that

(x) $\Gamma_0 \vdash^{\mathsf{atom}} e : t@q, \Lambda_e$

(xi) $(\Gamma_0, x : t@q), MFWts, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'$

By application of T-Frame-Atomic on the definition of F', (vi), (vii), (xi), and (ix), we have

$$\Psi, PDecl \vdash F' : \mathsf{ok}$$

Subcase $e = \mathsf{IN}()$**.** The rule T-let-IN-A applies. By inversion on (viii),

$$\frac{(\Gamma_0, x : t@\langle \mathsf{ld}, \mathsf{ld}, \mathsf{ld}\rangle), NVw, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'}{\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{let}\ x = \mathsf{IN}()\ \mathsf{in}\ c \Rightarrow MFWts'}\ \text{T-let-IN-A}$$

we learn that $(\Gamma, x : t@\langle \mathsf{ld}, \mathsf{ld}, \mathsf{ld}\rangle), NVw, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'$ (x).

By application of T-Frame-Atomic to (vi), (vii), (x), (ix), we have

$$\Psi, PDecl \vdash F' : \mathsf{ok}$$

Case $\mathtt{Md} = \mathsf{jit}$ **or** $\mathtt{Md} = \mathsf{discard}$ **or** $\mathtt{Md} = \mathsf{next}$**.** This case is similar to the one above, but instead uses the rules for nonatomic modes. We again obtain that $\Psi, PDecl \vdash F' : \mathsf{ok}$. The jit invariant for volatile memory is obtained by the observation that both V' and V'_k are extended with $x \mapsto (v, pID)$ in the mode jit.

In all cases, we have that $\Psi, PDecl \vdash F' : \mathsf{ok}$ (vi). Thus, by application of T-Config to (vi), (ii), and (iii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \mathsf{V}', S, F', Q : \mathsf{ok}$$

Case Seq1-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\cdot \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is Seq1-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c_1; c_2) \\ F' = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, ([\,]; c_2) :: E, \mathtt{Md}, c_1) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \text{Seq1-I}$$

We want to show that $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

(iv) if $mdOf(F_n) = \mathsf{jit}$ and $iPidOf(F_n) = ipID$, then $\forall x \in \mathsf{dom}(\mathsf{N}) \cap \mathsf{dom}(\mathsf{N}_k)$ $s.t.$ $\mathsf{N}(x) = (v, ipID)$, then $\mathsf{N}'_k(x) = (v, ipID)$ and $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k)$ $s.t.$ $\mathsf{V}(x) = (v, ipID)$, then $\mathsf{V}'_k(x) = (v, ipID)$

(v) if $mdOf(F_n) = \mathsf{jit}$ and $pidOf(F_n) = pID$, then $S_k = F_J(pID) \rhd S^0_k$

We need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$.

The proof proceeds by cases on $\mathtt{Md}$:

Case $\mathtt{Md} = \mathsf{jit}$ **or** $\mathtt{Md} = \mathsf{discard}(\_)$ **or** $\mathtt{Md} = \mathsf{next}(\_)$**.** By inversion of T-Frame-Jit-D on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, v, E, \mathtt{Md}, c_1; c_2) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ pcS = pc_0 \rhd pcS_0 \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \\ \Gamma = \Psi(ID, version, \mathtt{sMd}) \qquad \Gamma, pc_0 \vdash^{\mathtt{Md}} c_1; c_2 : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{Md}} E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Jit-D}$$

we have that

(vi) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(vii) $pcS = pc_0 \rhd pcS_0$

(viii) $\Gamma, pc_0 \vdash^{\mathtt{sMd}} c_1; c_2 : \mathsf{ok}$

(ix) $\Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok}$

By inversion of T-Sequence-S on (viii),

$$\frac{\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \Gamma, pc_0 \vdash^{\mathtt{sMd}} c_1 : \mathsf{ok} \qquad \Gamma, pc_0 \vdash^{\mathtt{sMd}} c_2 : \mathsf{ok}}{\Gamma, pc_0 \vdash^{\mathtt{sMd}} c_1; c_2 : \mathsf{ok}}\ \text{T-Sequence-S}$$

we have that

(x) $\Gamma, pc_0 \vdash^{\mathtt{sMd}} c_1 : \mathsf{ok}$

(xi) $\Gamma, pc_0 \vdash^{\mathtt{sMd}} c_2 : \mathsf{ok}$

By T-E-seq-1 applied to (ix) and (xi), we have

$$\frac{\Gamma, pc_0 \vdash^{\mathtt{sMd}} c_2 : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok} \qquad pcS = pc_0 \rhd pcS_0}{\Gamma, pcS \vdash^{\mathtt{sMd}} ([\,]; c_2) :: E : \mathsf{ok}\ (xii)}\ \text{T-E-seq-1}$$

By application of T-Frame-Jit-D on the definition of F', (vi), (vii), (x), (xii), we have that

$$\Psi, PDecl \vdash F' : \mathsf{ok}$$

The jit invariants are obtained directly from (iv) and (v).

Case $\mathtt{Md} = \mathsf{atom}$**.** This case is similar to the one above, but uses the rules for atomic mode.

In all cases we have $\Psi, PDecl \vdash F' : \mathsf{ok}$ (*iv*). By application of T-Config to (iv), (ii), and (iii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

Case Seq2-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\cdot \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is Seq2-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, ([\,]; c) :: E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \end{array}}{PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{Seq2-I}$$

We want to show that $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion of T-Config on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

We need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$.

The proof proceeds in cases on $\mathtt{Md}$:

Case $\mathtt{Md} = \mathsf{jit}$ **or** $\mathtt{Md} = \mathsf{discard}$ **or** $\mathtt{Md} = \mathsf{next}(\_)$**.** The rule T-Frame-Jit-D applies. By inversion on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, ([\,]; c) :: E, \mathtt{Md}, \mathsf{skip}) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ pcS = pc_0 \rhd pcS_0 \qquad \Gamma, pc_0 \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} ([\,]; c) :: E : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}}\ \textsc{T-Frame-Jit-D}$$

we learn that

(iv) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(v) $pcS = pc_0 \rhd pcS_0$

(vi) $\Gamma, pc_0 \vdash^{\mathtt{sMd}} \mathsf{skip} : \mathsf{ok}$

(vii) $\Gamma, pcS \vdash^{\mathtt{sMd}} ([\,]; c) :: E : \mathsf{ok}$

By inversion of T-E-seq-1

$$\frac{\Gamma, pc_0 \vdash^{\mathtt{sMd}} c : \mathsf{ok} \qquad \Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok} \qquad pcS = pc_0 \rhd pcS_0}{\Gamma, pcS \vdash^{\mathtt{sMd}} ([\,]; c) :: E : \mathsf{ok}}\ \textsc{T-E-seq-1}$$

we learn that

(viii) $\Gamma, pc_0 \vdash^{\mathtt{sMd}} c : \mathsf{ok}$

(ix) $\Gamma, pcS \vdash^{\mathtt{sMd}} E : \mathsf{ok}$

By application of T-Frame-Jit-D on (viii), (ix), (iv), (v), and the definition of F', we have that

$$\Psi, PDecl \vdash F' : \mathsf{ok}$$

The jit invariants hold by reasoning similar to the previous case.

Case $\mathtt{Md} = \mathsf{atom}$**.** Suppose $\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$ for some aID, rb, $\omega, \mu, \rho, Inits, NVw, Vw$. The rule T-Frame-Atomic applies. By inversion on (i),

$$\frac{\begin{array}{c} F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, ([\,]; c) :: E, \mathtt{Md}, \mathsf{skip}) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ pcS = pc_0 \rhd pcS_0 \qquad \Gamma_0, NVw, pc_0 \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts_1 \\ \Gamma_0, MFWts_1, pcS \vdash^{\mathsf{atom}} ([\,]; c) :: E \Rightarrow MFWts'' \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \end{array}}{PDecl \vdash F : \mathsf{ok}}\ \text{T-Frame-Atomic}$$

we have that

(iv) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(v) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(vi) $\Gamma_0, NVw, pc_0 \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts_1$

(vii) $pcS = pc_0 \rhd pcS_0$

(viii) $\Gamma_0, MFWts_1, pcS \vdash^{\mathsf{atom}} ([\,]; c) :: E \Rightarrow MFWts''$

By inversion of T-E-seq-2 on (viii),

$$\frac{\begin{array}{c} \Gamma_0, NVw, pc_0 \vdash^{\mathsf{atom}} c \Rightarrow MFWts' \\ \Gamma_0, MFWts', pcS \vdash^{\mathsf{atom}} E : \mathsf{ok} \qquad pcS = pc_0 \rhd pcS_0 \end{array}}{\Gamma_0, NVw, pcS \vdash^{\mathsf{atom}} ([\,]; c) :: E : \mathsf{ok}}\ \text{T-E-seq-2}$$

we know that

(ix) $\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'$

(x) $\Gamma_0, MFWts', pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}$

By application of T-Frame-Atomic on the definition of F', (iv), (v), (vii), (ix), (x), we have

$$\Psi, PDecl \vdash F' : \mathsf{ok}$$

In all cases, we have that $\Psi, PDecl \vdash F' : \mathsf{ok}$ (*iv*). By application of T-Config on (iv), (ii), (iii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$$

Case StartCrit-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is StartCrit-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}}) \\ F' = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} :: E, \mathtt{Md}, c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ StartCrit-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

We need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$. To do so, we invert on (i). Since the command in F is not an atomic region, either the rule T-Frame-A2A or T-Frame-Jit-D applies:

Case $\mathtt{sMd}(\mathtt{Md}) = \mathsf{atom}$**.** The rule T-Frame-A2A applies:

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}}) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \Rightarrow MFWts \\ \Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E : \mathsf{ok} \\ \forall NVw' \text{ s.t. } NVw' \supseteq NVw, \exists MFWts' \supseteq MFWts \text{ s.t.} \\ \Gamma_0, NVw', pc \vdash^{\mathsf{atom}} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \Rightarrow MFWts' \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-A2A}$$

By inversion of (i), we learn that:

(iv) $F = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}})$

(v) $\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$

(vi) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(vii) $\Gamma = \Psi(ID, version, \mathsf{jit})$

(viii) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(ix) $\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \Rightarrow MFWts$

(x) $\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}$

(xi) $\forall NVw'$ s.t. $NVw' \supseteq NVw, \exists MFWts' \supseteq MFWts$ s.t. $\Gamma_0, NVw', pc \vdash^{\mathsf{atom}} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \Rightarrow MFWts'$

By inversion of T-Critical-A on (ix):

$$\frac{\begin{array}{c}\Gamma_0, NVw, pc \vdash^{\mathsf{atom}}_{vPol} c \Rightarrow MFWts \\ pr_{\mathsf{c}} = ceilingPrOf(\{pr' \mid \mathsf{tk}(ID, version, pr', vPol, rPol, \lambda x.c) \in PDecl \\ \wedge\ (ShAcc \cap vPol.ShAcc \neq \emptyset)\}) \qquad pr \geq pr_{\mathsf{c}} \qquad ShAcc \subseteq vPol.ShAcc \\ ShAcc \neq \emptyset \qquad \Gamma_0 = \{x : t@q \mid x \in \mathsf{dom}(\Gamma) \wedge x \notin vPol.ShAcc \setminus ShAcc\}\end{array}}{\Gamma, NVw, \mathsf{Id} \vdash^{\mathsf{atom}}_{vPol} \mathsf{start}_{\mathsf{cr}}(pr, ShAcc); c; \mathsf{end}_{\mathsf{cr}} \Rightarrow MFWts}\ \text{T-Critical-A}$$

we learn that

(xii) $\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts$

(xiv) $pr_{\mathsf{c}} = ceilingPrOf(\{pr' \mid \mathsf{tk}(ID, version, pr', vPol, rPol, \lambda x.c) \in PDecl \wedge (ShAcc \cap vPol.ShAcc \neq \emptyset)\})$

(xv) $pr \geq pr_{\mathsf{c}}$

By application of T-E-critical-end-A to (x), we have:

$$\frac{\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}}{\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end}_{\mathsf{cr}} :: E : \mathsf{ok}}\ \text{T-E-critical-end-A} \qquad (xvi)$$

Lastly, we prove the invariant $\forall NVw'$ s.t. $NVw' \supseteq NVw$, $\exists MFWts' \supseteq MFWts$ s.t. $\Gamma_0, NVw', pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'$ holds. Let NVw' be arbitrary s.t. $NVw' \supseteq NVw$. Since a similar invariant holds in (xi), pick the same NVw' and invert T-Critical-A on (xi) to learn the desired result. Since NVw' was arbitrary, the result holds for all such NVw'.

By application of T-Frame-A2A to the definition of F', (vi), (vii), (viii), (xii), and (xvi), we have

$$\frac{\begin{array}{c}F' = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} :: E, \mathtt{Md}, c) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts \qquad \Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end}_{\mathsf{cr}} :: E : \mathsf{ok} \\ \forall NVw' \text{ s.t. } NVw' \supseteq NVw,\ \exists MFWts' \supseteq MFWts \text{ s.t. } \Gamma_0, NVw', pc \vdash^{\mathsf{atom}} c \Rightarrow MFWts'\end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-Frame-A2A}$$

Case *smodeOf*(Md) ∈ {jit, discard}. This case is similar, instead using the rule T-Frame-Jit-D.

The desired result follows by application of T-config to $\Psi, PDecl \vdash F' : \mathsf{ok}$, (ii), and (iii).

Case EndCrit-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is EndCrit-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} :: E, \mathtt{Md}, \mathsf{skip}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ EndCrit-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q : \mathsf{ok}$.

By inversion on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

We need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$. To do so, we invert on (i). Since the command in F is not an atomic region, either the rule T-Frame-A2A or T-Frame-Jit-D applies:

Case $\mathtt{sMd}(\mathtt{Md}) = \mathsf{atom}$**.** The rule T-Frame-A2A applies:

$$\frac{\begin{array}{c} E_0 = \mathsf{end}_{\mathsf{cr}} :: E \\ F = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, E_0, \mathtt{Md}, \mathsf{skip}) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts \qquad \Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E_0 : \mathsf{ok} \\ \forall NVw' \text{ s.t. } NVw' \supseteq NVw,\ \exists MFWts' \supseteq MFWts \text{ s.t.} \\ \Gamma_0, NVw', pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts' \end{array}}{\Psi, PDecl \vdash F : \mathsf{ok}} \text{ T-Frame-A2A}$$

By inversion of (i), we learn that:

(iv) $F = (pID, rID, ID, version, pr \rhd \overrightarrow{pr}_c, pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{cr}} :: E, \mathtt{Md}, \mathsf{skip})$

(v) $\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$

(vi) $\mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl$

(vii) $\Gamma = \Psi(ID, version, \mathsf{jit})$

(viii) $\Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits)$

(ix) $\Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts$

(x) $\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{cr}} :: E : \mathsf{ok}$

(xi) $\forall NVw'$ s.t. $NVw' \supseteq NVw$, $\exists MFWts' \supseteq MFWts$ s.t. $\Gamma_0, NVw', pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts'$

By inversion of T-E-CRITICAL-END-A on (x),

$$\frac{\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}}{\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} \mathsf{end_{cr}} :: E : \mathsf{ok}}\ \text{T-E-CRITICAL-END-A}$$

we learn that $\Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E : \mathsf{ok}$ (*xii*).

By application of T-FRAME-A2A to the definition of F', (vi), (vii), (viii), (ix), (xi), and (xii), we have

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}_c, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip}) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw) \\ \mathsf{task}(ID, version, pr, vPol, rPol, \lambda x.c_0) \in PDecl \\ \Gamma = \Psi(ID, version, \mathsf{jit}) \qquad \Gamma_0 = mkCtxAtom(\Gamma, \omega, \mu, \rho, Inits) \\ \Gamma_0, NVw, pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts \qquad \Gamma_0, MFWts, pc \rhd pcS \vdash^{\mathsf{atom}} E : \mathsf{ok} \\ \forall NVw' \text{ s.t. } NVw' \supseteq NVw,\ \exists MFWts' \supseteq MFWts \text{ s.t.} \\ \Gamma_0, NVw', pc \vdash^{\mathsf{atom}} \mathsf{skip} \Rightarrow MFWts' \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}}\ \text{T-FRAME-A2A}$$

Case $\mathtt{sMd}(\mathtt{Md}) \in \{\mathsf{jit}, \mathsf{discard}\}$**.** This case is similar, instead using the rule T-FRAME-JIT-D.

The desired result follows by application of T-CONFIG to $\Psi, PDecl \vdash F' : \mathsf{ok}$, (ii), and (iii).

Case Reboot-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is REBOOT-I.

$$\frac{\begin{array}{c} K = (N_k, V, F \rhd S_k) \qquad F = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ F' = (pID, rID, ID, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}', c) \qquad polOf(F) \neq \mathsf{altOf}(\_) \\ Q = Q_d, Q_k \qquad \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \qquad \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \\ \mathtt{Md}' = \begin{cases} \mathsf{atom}(\mathsf{aID}, rb+1, \omega, \mu, \rho, \emptyset, \emptyset) & \text{if } \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, NVw, Vw) \\ \mathtt{Md} & \text{otherwise} \end{cases} \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K, \mathsf{N} \leftsquigarrow N_k, V, S_k, F', Q_k}\ \text{REBOOT-I}$$

$$\frac{\begin{array}{c} K = (N_k, V, S) \qquad (F \rhd S_k) = incRb(S) \\ K' = (N_k, V, F \rhd S_k) \qquad F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ polOf(F) \neq \mathsf{altOf}(\_) \qquad Q = Q_d, Q_k \qquad \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \\ \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \qquad ipidOf(F') = ipID' \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K', N_k^{ipID'} \rightsquigarrow \mathsf{N}, V^{ipID'}, S_k, F, Q_k}\ \text{REBOOT-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K', N_k^{ipID'} \rightsquigarrow \mathsf{N}, V^{ipID'}, S_k, F, Q_k : \mathsf{ok}$.

By inversion of T-CONFIG on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have

$$\Psi, PDecl \vdash K : \mathsf{ok}\ (i)$$

In particular, we need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$ and $\Psi, PDecl \vdash S : \mathsf{ok}$. By definition of $incRb(S)$, we know that $\Psi, PDecl \vdash F \rhd S_k : \mathsf{ok}$.

By inversion of T-STACK-FULL, we have that

(ii) $\Psi, PDecl \vdash F : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash S_k : \mathsf{ok}$

By application of T-CONFIG on (i), (ii), and (iii), we obtain the desired result:

$$\Psi, PDecl \vdash I, IRQ, K', N_k^{ipID'} \rightsquigarrow \mathsf{N}, V^{ipID'}, S_k, F, Q_k : \mathsf{ok}$$

Case Reboot-IA. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q)$ and suppose that $\Psi, PDecl \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is REBOOT-IA.

$$\frac{\begin{array}{c} K = (N_k, V, F \rhd S) \qquad S_k = incRb(S) \qquad K' = (N_k, V, S_k) \\ F = (pID_n, rID_n, ID_n, version_n, \overrightarrow{pr}_n, pcS_n, vPol_n, x \mapsto v, \cdot, \mathsf{altOf}(\_, rPol_n), c_n; \mathsf{ret}(res)) \\ Q = Q_d, Q_k \qquad \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \qquad \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \\ (\mathtt{Md}_n, K'') = schdFrame(PDecl, (pID_n, ID_n, v, rPol_n), K', \mathsf{N}) \\ \mathsf{N}' = N_k^{pID_n} \rightsquigarrow \mathsf{N} \qquad ipID = iPidOf(pID_n, \mathtt{Md}_n) \\ F' = (pID_n, rID_n, ID_n, version_n, \overrightarrow{pr}_n, pcS_n, vPol_n, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol_n.Inits); c_n; \mathsf{ret}(res)) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K'', \mathsf{N}', V^{pID_n}, S_k, F', Q_k}\ \text{REBOOT-IA}$$

By inversion of T-CONFIG on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have that

$$\Psi, PDecl \vdash K : \mathsf{ok}\ (i)$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K'', \mathsf{N}', V^{pID_n}, S_k, F', Q_k : \mathsf{ok}$.

We will first show that $\Psi, PDecl \vdash K'' : \mathsf{ok}$.

By inversion of T-K on (i), we know that $\Psi, PDecl \vdash F \rhd S : \mathsf{ok}\ (ii)$.

By inversion of T-STACK-FULL on (ii), we have

(iii) $\Psi, PDecl \vdash F : \mathsf{ok}$

(iv) $\Psi, PDecl \vdash S : \mathsf{ok}$

By application of T-K to (iv) and $K' = (\mathsf{N}_k, \mathsf{V}, S_k)$, we have that

$$\Psi, PDecl \vdash K' : \mathsf{ok}$$

By Lemma 44 applied to $PDecl \vdash K' : \mathsf{ok}$, we have

$$\Psi, PDecl \vdash K'' : \mathsf{ok}\ (v)$$

We now need to show that $\Psi, PDecl \vdash F' : \mathsf{ok}$.

Let $\mathtt{sMd} = smodeOf(\mathtt{Md}_n)$. By $\Psi \vdash PDecl : \mathsf{ok}$ and sequencing for commands, we know that $\Gamma, \mathsf{Id} \vdash^{\mathtt{sMd}}_{pr_n} initN(vPol_n.Inits); c_n; \mathsf{ret}(res) : \mathsf{ok}$ (vi).

By T-E-MD, we have $\Gamma, \mathsf{Id} \vdash^{\mathtt{sMd}}_{pr_n} \cdot : \mathsf{ok}$ (vii).

By definition of *schdFrame*, we know that $\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}$. So the rule T-FRAME-JIT-D applies. By application of T-FRAME-JIT-D on the definition of F', (vi), (vii), we have

$$\frac{\begin{array}{c} F' = (pID_n, rID_n, ID_n, \overrightarrow{pr}_n, \mathsf{Id}, vPol_n, x \mapsto v, \cdot, \mathtt{Md}_n, initN(vPol_n.Inits); c_n; \mathsf{ret}(res)) \\ \mathsf{task}(ID_n, version_n, pr_n, vPol_n, rPol_n, \lambda x.c_n) \in PDecl \\ \mathtt{sMd} = smodeOf(\mathtt{Md}_n) \qquad \mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\} \qquad \Gamma = \Psi(ID_n, version_n, \mathtt{sMd}) \\ \Gamma, \mathsf{Id} \vdash^{\mathtt{Md}_n} initN(vPol_n.Inits); c_n; \mathsf{ret}(res) : \mathsf{ok} \qquad \Gamma, \mathsf{Id} \vdash^{\mathtt{Md}_n} \cdot : \mathsf{ok} \end{array}}{\Psi, PDecl \vdash F' : \mathsf{ok}\ (viii)}\ \text{T-FRAME-JIT-D}$$

By application of T-CONFIG on (viii), (iv), and (v), we obtain the final desired result:

$$\Psi, PDecl \vdash I, IRQ, K'', \mathsf{N}', V^{pID_n}, S_k, F', Q_k : \mathsf{ok}$$

Case PowerFail-I. Let $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and suppose that $\Psi \vdash PDecl : \mathsf{ok}$, $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, and the last step is POWERFAIL-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ K = (N_k, V_k, S_k) \qquad S'_k = updJitS(S_k, F \rhd S) \qquad K' = (N_k, V_k, S'_k) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q}\ \text{POWERFAIL-I}$$

We need to show that $\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q : \mathsf{ok}$. In particular, it suffices to show that $\Psi, PDecl \vdash K' : \mathsf{ok}$.

By inversion of T-CONFIG on $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$, we have that

(i) $\Psi, PDecl \vdash F : \mathsf{ok}$

(ii) $\Psi, PDecl \vdash S : \mathsf{ok}$

(iii) $\Psi, PDecl \vdash K : \mathsf{ok}$

By inversion of T-K on (iii), we know that $\Psi, PDecl \vdash S_k : \mathsf{ok}$ (iv).

By Lemma 41 applied to (iv) and (ii), we know that

$$\Psi, PDecl \vdash S'_k : \mathsf{ok}\ (v)$$

By application of T-K on (v), we know that

$$\Psi, PDecl \vdash K' : \mathsf{ok} \; (vi)$$

The desired result follows by application of T-Config-reboot on (vi):

$$\Psi, PDecl \vdash I, IRQ, K', \mathsf{N}, \cdot, \cdot, \mathsf{reboot}, Q : \mathsf{ok}$$

□

Lemma 41 (*updJitS* produces well-formed S_k) *If* $\Psi, PDecl \vdash S_k : \mathsf{ok}$, $\Psi, PDecl \vdash S : \mathsf{ok}$, *and* $S'_k = updJitS(S_k, S)$, *then* $\Psi, PDecl \vdash S'_k : \mathsf{ok}$.

Proof: The proof is by nested induction on the size of S_k and S.

Base case $S_k = \cdot$. There are two subcases:

Subcase $S = \cdot$.

$$\frac{}{updJitS(\cdot, \cdot) = \cdot}$$

In this case, we apply T-Stack-Empty to obtain the desired result:

$$PDecl \vdash \cdot : \mathsf{ok}$$

Subcase $S = F \rhd S' \; (\exists F)$.

$$\frac{modeOf(F) \neq \mathsf{jit}}{updJitS(S_k, F \rhd S') = updJitS(S_k, S')}$$

By inversion of T-Stack-full on $PDecl \vdash S : \mathsf{ok}$ (and since $S = F \rhd S'$),

$$\frac{S = F \rhd S' \qquad PDecl \vdash S' : \mathsf{ok} \qquad PDecl \vdash F : \mathsf{ok}}{PDecl \vdash S : \mathsf{ok}} \; \text{T-Stack-full}$$

we have

(iii) $PDecl \vdash S' : \mathsf{ok}$

(iv) $PDecl \vdash F : \mathsf{ok}$

Since S' is one frame smaller than S, we can apply the inductive hypothesis to (iii) and $PDecl \vdash S_k : \mathsf{ok}$, to establish

$$PDecl \vdash updJitS(S_k, S') : \mathsf{ok}$$

as desired.

Inductive case $S_k = F \rhd S''_k$ $(\exists F)$**.** There are two subcases to consider:

Subcase $S = \cdot$**.**

$$\frac{\mathit{modeOf}(F) \neq \mathsf{jit}}{\mathit{updJitS}(F \rhd S''_k, S) = F \rhd \mathit{updJitS}(S''_k, S)}$$

By inversion of T-Stack-full on $\mathit{PDecl} \vdash S_k : \mathsf{ok}$ (and since $S_k = F \rhd S''_k$),

$$\frac{S_k = F \rhd S''_k \qquad \mathit{PDecl} \vdash S''_k : \mathsf{ok} \qquad \mathit{PDecl} \vdash F : \mathsf{ok}}{\mathit{PDecl} \vdash S_k : \mathsf{ok}}\ \text{T-Stack-full}$$

we have

(iii) $\mathit{PDecl} \vdash S''_k : \mathsf{ok}$

(iv) $\mathit{PDecl} \vdash F : \mathsf{ok}$

We can apply the inductive hypothesis on (iii) and $\mathit{PDecl} \vdash S : \mathsf{ok}$ to learn that

$$\mathit{PDecl} \vdash \mathit{updJitS}(S''_k, S) : \mathsf{ok}\ (v)$$

By application of T-Stack-full to (v) and (iv), we learn that

$$\mathit{PDecl} \vdash F \rhd \mathit{updJitS}(S''_k, S) : \mathsf{ok}$$

Subcase $S = F \rhd S'$ $(\exists F)$**.**

$$\frac{S'''_k = \mathit{updJitS}(S''_k, S') \qquad \mathit{modeOf}(F) = \mathsf{jit} \qquad \mathit{pidOf}(F) = \mathit{pidOf}(F_k)}{\mathit{updJitS}(F_k \rhd S''_k, F \rhd S') = F \rhd S'''_k}$$

By inversion of T-Stack-full on $\mathit{PDecl} \vdash S : \mathsf{ok}$ (and since $S = F \rhd S'$),

$$\frac{S = F \rhd S' \qquad \mathit{PDecl} \vdash S' : \mathsf{ok} \qquad \mathit{PDecl} \vdash F : \mathsf{ok}}{\mathit{PDecl} \vdash S : \mathsf{ok}}\ \text{T-Stack-full}$$

we have

(iii) $\mathit{PDecl} \vdash S' : \mathsf{ok}$

(iv) $\mathit{PDecl} \vdash F : \mathsf{ok}$

By inversion of T-Stack-full on $\mathit{PDecl} \vdash S_k : \mathsf{ok}$ (and since $S_k = F_k \rhd S''_k$),

$$\frac{S_k = F_k \rhd S''_k \qquad \mathit{PDecl} \vdash S''_k : \mathsf{ok} \qquad \mathit{PDecl} \vdash F_k : \mathsf{ok}}{\mathit{PDecl} \vdash S_k : \mathsf{ok}}\ \text{T-Stack-full}$$

we have

(v) $PDecl \vdash S''_k : \mathsf{ok}$

(vi) $PDecl \vdash F_k : \mathsf{ok}$

We can apply the inductive hypothesis on (iii) and (v) to learn that

$$PDecl \vdash updJitS(S''_k, S') : \mathsf{ok}\ (vii)$$

Since $S'''_k = updJitS(S''_k, S')$, it follows by application of T-Stack-full to (vii) and (iv) that

$$PDecl \vdash F \rhd S'''_k$$

which is the desired result.

□

Lemma 42 (Expression qualifiers match Λ for nonatomic mode) *If* $\Gamma \vdash^{\mathtt{Md}} e : t@q_e, \Lambda_e$ *and* $\mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{next}\}$*, then* $q_e = (\sqcup_{x \in \Lambda_e} \Gamma(x)\downarrow_{\overline{q}})$.

Proof: The proof is by structural induction on e.

Case $e = x$**.** Then the last rule in the derivation must be T-Var-Read3. By inversion,

$$\frac{\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \qquad \mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{next}\}}{\Gamma \vdash^{\mathtt{Md}} x : t@\langle qID_c, qID_l, qID_h\rangle, \{x\}}\ \text{T-Var-Read3}$$

we have that

$$\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle\ (i)$$

Hence, $\Gamma(x)\downarrow_q = \langle qID_c, qID_l, qID_h\rangle$, as desired.

Case $e = e_1 \odot e_2$**.** Then the last rule in the derivation must be T-Binop-Read-J. By inversion,

$$\frac{\mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}, \mathsf{next}\} \qquad \Gamma \vdash^{\mathtt{Md}} e_1 : t@q_1, \Lambda_1 \qquad \Gamma \vdash^{\mathtt{Md}} e_2 : t@q_2, \Lambda_2}{\Gamma \vdash^{\mathtt{Md}} e_1 \odot e_2 : t@(q_1 \sqcup q_2), \Lambda_1 \cup \Lambda_2}\ \text{T-Binop-Read-J}$$

we have that

(i) $\Gamma \vdash^{\mathtt{Md}} e_1 : t@q_1, \Lambda_1$

(ii) $\Gamma \vdash^{\mathtt{Md}} e_2 : t@q_2, \Lambda_2$

By the inductive hypothesis applied to (i) and (ii), we learn that

(iii) $(\sqcup_{x\in\Lambda_1}\Gamma(x)\downarrow_q) = q_1$

(iv) $(\sqcup_{x\in\Lambda_2}\Gamma(x)\downarrow_q) = q_2$

Combining (iii) and (iv), we have that

$$\begin{aligned} q_1 \sqcup q_2 &= (\sqcup_{x\in\Lambda_1}\Gamma(x)\downarrow_q) \sqcup (\sqcup_{x\in\Lambda_2}\Gamma(x)\downarrow_q) \\ &= (\sqcup_{x\in\Lambda_1\cup\Lambda_2}\Gamma(x)\downarrow_q) = q_1 \sqcup q_2 \end{aligned}$$

□

Lemma 43 (Expression qualifiers match Λ for atomic mode) *If* $\Gamma, MFWts \vdash^{\mathsf{atom}} e : t@q_e, \Lambda_e$, *then* $q_e = (\sqcup_{x\in\Lambda_e}\Gamma(x)\downarrow_{\overline{q}})$.

Proof: The proof is by structural induction on e.

Case $e = x$**.** There are two subcases, according to qID_c

Case $\boldsymbol{qID_c \neq qID}\mathsf{Id}(\mathtt{WT})$**.** Then the last rule in the derivation must be T-Var-Read1. By inversion,

$$\frac{\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \qquad qID_c \neq qID\mathsf{Id}(\mathtt{WT})}{\Gamma, MFWts \vdash^{\mathsf{atom}} x : t@\langle\overline{qID}_c, qID_l, qID_h\rangle, \{x\}}\ \text{T-Var-Read1}$$

we learn that $\Gamma(x) = t@\langle\overline{qID}_c, qID_l, qID_h\rangle$ (i).

Hence,
$qidtriple\overline{qID}_c qID_l qID_h = (\Gamma(x)\downarrow_{\overline{q}})$.

Case $\boldsymbol{qID_c \neq qID}\mathsf{Id}(\mathtt{WT})$**.**

$$\frac{\Gamma(x) = t@\langle qID_c, qID_l, qID_h\rangle \qquad qID_c = \mathsf{Id}(\mathtt{WT}) \qquad x \in MFWts}{\Gamma, MFWts \vdash^{\mathsf{atom}} x : t@\langle\mathsf{Id}, qID_l, qID_h\rangle, \{x\}}\ \text{T-Var-Read2}$$

Case $e = e_1 \odot e_2$**.** Then the last rule in the derivation must be T-Binop-Read-A. By inversion,

$$\frac{\Gamma, MFWts \vdash^{\mathsf{atom}} e_1 : t@q_1, \Lambda_1 \qquad \Gamma, MFWts \vdash^{\mathsf{atom}} e_2 : t@q_2, \Lambda_2}{\Gamma, MFWts \vdash^{\mathsf{atom}} e_1 \odot e_2 : t@(q_1 \sqcup q_2), \Lambda_1 \cup \Lambda_2}\ \text{T-Binop-Read-A}$$

we learn that

(i) $\Gamma, MFWts \vdash^{\mathsf{atom}} e_1 : t@q_1, \Lambda_1$

(ii) $\Gamma, MFWts \vdash^{\mathsf{atom}} e_2 : t@q_2, \Lambda_2$

By the inductive hypothesis applied to (i) and (ii), we have

(iii) $q_1 = (\sqcup_{x \in \Lambda_1} \Gamma(x) \downarrow_{\overline{q}})$

(iv) $q_2 = (\sqcup_{x \in \Lambda_2} \Gamma(x) \downarrow_{\overline{q}})$

Combining (iii) and (iv), we obtain the desired result:

$$\begin{aligned} q_1 \sqcup q_2 &= (\sqcup_{x \in \Lambda_1} \Gamma(x) \downarrow_{\overline{q}}) \sqcup (\sqcup_{x \in \Lambda_2} \Gamma(x) \downarrow_{\overline{q}}) \\ &= (\sqcup_{x \in (\Lambda_1 \cup \Lambda_2)} \Gamma(x) \downarrow_{\overline{q}}) \end{aligned}$$

□

Lemma 44 (*schdFrame* produces well-formed K) *If*

- $PDecl \vdash K : \mathsf{ok}$ *and*
- $(\mathtt{Md}_n, K') = schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N})$,

then $PDecl \vdash K' : \mathsf{ok}$.

Proof: Let $K = (\mathsf{N}_k, \mathsf{V}_k, S_k)$. By inversion of T-K on $PDecl \vdash K : \mathsf{ok}$,

$$\frac{K = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad PDecl \vdash S_k : \mathsf{ok}}{PDecl \vdash K : \mathsf{ok}} \text{ T-K}$$

we know that $PDecl \vdash S_k : \mathsf{ok}$ (i).

The proof proceeds in cases on $appPol(rPol)$.

Case *appPol*(*rPol*) = immediate. The rule schdFrame-Immediate applies. By inversion,

$$\frac{\begin{array}{c} appPol(rPol) = \mathsf{immediate} \\ K = (N_k, V_k, S_k) \qquad S'_k = (pID, \mathsf{jit}) \rhd S_k \qquad K' = (N_k, V_k, S'_k) \end{array}}{schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N}) = (\mathsf{jit}, K')} \text{ schdFrame-Immediate}$$

we learn that

(ii) $S'_k = (pID, \mathsf{jit}) \rhd S_k$

(iii) $K' = (N_k, V_k, S'_k)$

From T-Frame-Jit-P, we know that $PDecl \vdash (pID, \mathsf{jit}) : \mathsf{ok}$ (iv).

By application of T-Stack-Full to (iv), (i), and (ii), we have that

$$PDecl \vdash S'_k : \mathsf{ok} \ (v)$$

By application of T-K to (v) and (iii), we learn

$$PDecl \vdash K' : \mathsf{ok}$$

Case *appPol*(*rPol*) = discard. The rule schdFrame-Discard applies. By inversion,

$$\frac{\begin{array}{c}appPol(rPol) = \mathsf{discard} \qquad \mathsf{isr}(iID, 0, pr, pr_{\mathsf{ceil}}, \Gamma, vPol, rPol, \lambda x.c_i) \in PDecl \\ vPol = (ShAcc, ShCP, nIds, Inits) \\ K = (N_k, V_k, S_k) \qquad K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}), V_k, S_k)\end{array}}{schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N}) = (\mathsf{discard}, K')}\ \textsc{schdFrame-Discard}$$

we learn that $K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}, V_k, S_k)$ (ii).
By application of T-K to (i) and (ii),

$$PDecl \vdash K' : \mathsf{ok}$$

Case *appPol*(*rPol*) = alternative. The rule schdFrame-Alternative applies. By inversion,

$$\frac{\begin{array}{c}appPol(rPol) = \mathsf{alternative}(iID, 1) \\ \mathsf{task}(ID, 1, pr, pr^a_{\mathsf{ceil}}, \Gamma_a, vPol_a, rPol_a, \lambda x.c_a) \in PDecl \\ fresh(pID_a) \qquad fresh(rID_a) \\ F_a = (pID_a, rID_a, ID, 1, pr, vPol_a, x \mapsto v, \cdot, \mathsf{altOf}(pID, rPol_a), c_a; ret) \\ K = (N_k, V_k, S_k) \qquad S'_k = F_a \rhd S_k \\ K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}, V_k, S'_k) \qquad vPol_a = (ShAcc, ShCP, nIds, Inits)\end{array}}{schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N}) = (\mathsf{next}(pID_a), K')}\ \textsc{schdFrame-Alternative}$$

we learn that

(ii) $\mathsf{task}(ID, 1, pr, pr^a_{\mathsf{ceil}}, \Gamma_a, vPol_a, rPol_a, \lambda x.c_a) \in PDecl$

(iii) $F_a = (pID_a, rID_a, ID, 1, pr, vPol_a, x \mapsto v, \cdot, \mathsf{altOf}(pID, rPol_a), c_a; ret)$

(iv) $S'_k = F_a \rhd S_k$

(v) $K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}, V_k, S'_k)$

By (ii) and the assumption $\cdot \vdash PDecl : \mathsf{ok}$, we have that

$$\Gamma_a, \mathsf{Id} \vdash^{\mathtt{Md}^a}_{pr^a_{\mathsf{ceil}}} c_a; ret \Rightarrow \Gamma'_a\ (\exists \Gamma'_a)\ (vi)$$

where $\mathtt{Md}^a = \mathsf{jit}$ if $appPol(rPol_a) = \mathsf{immediate}$, and $\mathtt{Md}^a = \mathsf{discard}$ otherwise.
By T-E-md, we know that $\Gamma'_a, \mathsf{Id} \vdash^{\mathtt{Md}^a}_{pr^a_{\mathsf{ceil}}} \cdot \Rightarrow \Gamma'_a$ (vii).
By application of T-Frame-Jit-Discard-Alternative to (ii), (iii), (vi), and (vii), we have:

$$\frac{\begin{array}{c}F_a = (pID_a, rID_a, ID, 1, pr, vPol_a, x \mapsto v, \cdot, \mathsf{altOf}(pID, rPol_a), c_a; ret) \\ \mathsf{task}(ID, 1, pr, pr^a_{\mathsf{ceil}}, \Gamma_a, vPol_a, rPol_a, \lambda x.c_a) \in PDecl \\ \Gamma_a, \mathsf{Id} \vdash^{\mathtt{Md}^a}_{pr^a_{\mathsf{ceil}}} c_a; ret \Rightarrow \Gamma'_a \\ \Gamma'_a, \mathsf{Id} \vdash^{\mathtt{Md}^a}_{pr^a_{\mathsf{ceil}}} \cdot \Rightarrow \Gamma'_a \qquad \mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}(\ldots), \mathsf{next}(\ldots)\}\end{array}}{PDecl \vdash F_a : \mathsf{ok}\ (viii)}\ \textsc{T-Frame-Jit-Discard-Alternative}$$

By application of T-Stack-full to (*i*), (*iv*), and (*viii*),

$$\frac{S'_k = F_a \rhd S_k \qquad PDecl \vdash S_k : \mathsf{ok} \qquad PDecl \vdash F_a : \mathsf{ok}}{PDecl \vdash S'_k : \mathsf{ok}\ (xvi)}\ \text{T-Stack-full}$$

By application of T-K to (xvi) and (v), we have

$$\frac{K' = (N_k \leftsquigarrow N{\downarrow_{ShCP}}, V_k, S'_k) \qquad PDecl \vdash S'_k : \mathsf{ok}}{PDecl \vdash K' : \mathsf{ok}}\ \text{T-K}$$

In all cases, we have that $PDecl \vdash K' : \mathsf{ok}$.

□

Lemma 45 (*finalizeK* produces well-formed K) *If $PDecl \vdash K : \mathsf{ok}$ and $finalizeK(K, F, N, V) = K'$, then $PDecl \vdash K' : \mathsf{ok}$.*

Proof: By inversion of T-K on $PDecl \vdash K : \mathsf{ok}$,

$$\frac{K = (\mathsf{N}_k, \mathsf{V}_k, S_k) \qquad PDecl \vdash S_k : \mathsf{ok}}{PDecl \vdash K : \mathsf{ok}}\ \text{T-K}$$

we learn that

(i) $K = (\mathsf{N}_k, \mathsf{V}_k, S_k)$

(ii) $PDecl \vdash S_k : \mathsf{ok}$

The proof proceeds in cases on $\mathtt{Md}_c$:

Case $\mathtt{Md}_c = \mathsf{jit}$. The rule fin-K-jit applies:

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{jit}, \mathsf{ret}) \\ K = (N_k, V_k, S_k) \qquad S_k = F_k \rhd S'_k \qquad K' = (N'_k{\downarrow_{ckVarOf(S'_k)}}, V_k, S'_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{fin-K-jit}$$

By inversion of T-Stack-Full on (ii), and since $S_k = F_k \rhd S'_k$, we learn that $PDecl \vdash S'_k : \mathsf{ok}$ (*iii*).

Thus, by T-K applied to (*iii*) and $K' = (N'_k{\downarrow_{ckVarOf(S'_k)}}, V_k, S'_k)$, we have that

$$PDecl \vdash K' : \mathsf{ok}$$

Case $\mathtt{Md}_c = \mathsf{discard}$. The rule fin-K-discard applies:

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{discard}(ShCP, NVw, Vw), \mathsf{ret}) \\ K = (N_k, V_k, S_k) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{fin-K-discard}$$

From T-K applied to (ii) and $K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)$, we learn that

$$PDecl \vdash K' : \mathsf{ok}$$

Case $\mathtt{Md}_c = \mathsf{next}(\_)$**.** The rule FIN-K-NEXT applies:

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{next}(pID_a, ShCP, NVw, Vw), \mathsf{ret}) \\ K = (N_k, V_k, F_a \rhd S_k) \\ fresh(rID_a) \qquad F_a = (pID_a, rID_a, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \_, \_, \mathsf{altOf}(pID_c, \_), \_) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k) \end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{FIN-K-NEXT}$$

From T-K applied to (ii) and $K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)$, we learn that

$$PDecl \vdash K' : \mathsf{ok}$$

In all cases, we have

$$PDecl \vdash K' : \mathsf{ok}$$

□

Theorem 19 (Preservation for continuously-powered execution) *If* $\cdot \vdash PDecl : \mathsf{ok}$*,* $PDecl \vdash \sigma : \mathsf{ok}$*, and* $PDecl \vdash \sigma \longrightarrow \sigma'$*, then* $PDecl \vdash \sigma' : \mathsf{ok}$*.*

Proof: The proof is analogous to Theorem 18 but uses the rules for continuous execution. □

E.9 Equivalences

E.9.1 Helper functions

Definition of *ePidOf*(T). We define *ePidOf*(T) to be the set of effective instance process identifiers in a trace T. They include successfully completed atomic region executions, completed discard and alternative tasks, and jit processes (because these re-execute until successful). We use (pID, aID, rb) versus pID to distinguish between atomic versus jit regions in immediate tasks. The $\mathsf{end}_{\mathsf{atom}}$ is okay here because when running an atomic region in a discard task, we throw away $\mathsf{end}_{\mathsf{atom}}$ so it does not appear in the execution trace.

$$\begin{aligned} \mathit{ePidOf}(T_I) = \; & \{pID \mid \mathsf{trigger}(pID, \mathtt{Md}), \mathsf{complete}(pID, v_r) \in T_I, \mathit{appPol}(\mathit{rPol}) \in \{\mathsf{discard}, \mathsf{next}\}\} \\ & \cup \{pID \mid \mathsf{trigger}(pID, \mathsf{jit}) \in T_I\} \\ & \cup \{(pID, \mathsf{aID}, rb) \mid \mathsf{endAtom}(pID, \mathsf{aID}, rb, v_r) \in T_I\} \end{aligned}$$

Definition of context lookup. If the mode is atomic, we need to look up the jit context in Ψ and pass it to *mkCtxAtom*, otherwise it is just a normal Ψ lookup.

$$\frac{\mathtt{Md} \in \{\mathsf{jit}, \mathsf{discard}(\_), \mathsf{next}(\_)\}}{\Psi(\mathit{ID}, \mathit{version}, \mathtt{Md}) = \Gamma} \textsc{TypeCtx-DJN}$$

$$\frac{\begin{array}{c}\Psi(\mathit{ID}, \mathit{version}, \mathsf{jit}) = \Gamma \qquad \Gamma' = \mathit{mkCtxAtom}(\Gamma, \omega, \mu, \beta, \rho, \mathit{Inits}, \mathit{nIds}_t) \\ \mathsf{task}(\mathit{ID}, \mathit{version}, \mathit{pr}, \mathit{vPol}, \mathit{rPol}, \lambda x.c_0) \in \mathit{PDecl} \\ \mathit{vPol} = (\mathit{ShAcc}_t, \mathit{ShCP}_t, \mathit{nIds}_t, \mathit{Inits}_t)\end{array}}{\Psi(\mathit{ID}, \mathit{version}, \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \beta, \rho, \mathit{Inits}, \mathit{NVw}, \mathit{Vw})) = \Gamma'} \textsc{TypeCtx-A}$$

$$\frac{\mathtt{Md} = \begin{cases} \mathsf{jit} & \text{if } \mathsf{range}(\mathit{rPol}) = \mathsf{immediate} \\ \mathsf{discard} & \text{if } \mathsf{range}(\mathit{rPol}) \in \{\mathsf{discard}, \mathsf{alternative}\} \end{cases}}{\Psi(\mathit{ID}, \mathit{version}, \mathsf{altOf}(\mathit{rPol})) = \Psi(\mathit{ID}, \mathit{version}, \mathtt{Md})} \textsc{TypeCtx-altOf}$$

Definition of *NVWtsOf*(Md).

$$\frac{}{\mathit{NVWtsOf}(\mathsf{jit}) = \cdot}$$

$$\frac{\mathtt{Md} \in \{\mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \beta, \rho, \mathit{Inits}, \mathit{NVw}, \mathit{Vw}), \mathsf{discard}(\mathit{NVw}, \mathit{Vw}), \mathsf{next}(\mathit{pID}, \mathit{NVw}, \mathit{Vw})\}}{\mathit{NVWtsOf}(\mathtt{Md}) = \mathit{NVw}}$$

Definition of *VWtsOf*(Md).

$$\frac{}{\mathit{VWtsOf}(\mathsf{jit}) = \cdot}$$

$$\frac{\mathtt{Md} \in \{\mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \beta, \rho, \mathit{Inits}, \mathit{NVw}, \mathit{Vw}), \mathsf{discard}(\mathit{NVw}, \mathit{Vw}), \mathsf{next}(\mathit{pID}, \mathit{NVw}, \mathit{Vw})\}}{\mathit{VWtsOf}(\mathtt{Md}) = \mathit{Vw}}$$

Definition of *getID*(pID_w).

$$\frac{pID_w = (ID_w, version_w, u_w)}{getID(pID_w) = ID_w, version_w}$$

Definition of *shCPof*(*PDecl*, *pID*) (task-level shared checkpoints only). *shCPof*(*PDecl*, *pID*) returns the checkpointed shared variables for the task *pID* as specified in *PDecl*. In the definition below, we omit *PDecl* for simplicity.

$$\frac{\begin{array}{c} getID(pID) = ID, version \\ tsk(ID, version, pr, vPol, rPol, \lambda x.c) \in PDecl \qquad vPol = (ShAcc, ShCP, nIds, Inits) \end{array}}{shCPof(PDecl, pID) = ShCP}$$

Definition of *localTyOf*(Ψ, *PDecl*, pID_w, x).

$$\frac{\begin{array}{c} getID(pID_w) = ID, version \qquad tsk(ID, version, pr, vPol, rPol, \lambda x.c) \in PDecl \\ appPol(rPol) = \mathtt{Md} \qquad \Psi(ID, version, \mathtt{Md}) = \Gamma \qquad \Gamma(x) = \langle qID_c, qID_l, qID_h \rangle \end{array}}{localTyOf(\Psi, PDecl, pID_w, x) = qID_c}$$

E.9.2 Two trace relation relating intermittent and continuous execution

Extracting effectful stack frames. We extract effectful stack frames and identify delayed ones.

$$\boxed{EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)}$$

$$\frac{}{EPids, EIrqIds, EEvIds \vdash getES(\emptyset, \emptyset) = (\emptyset, \emptyset, \emptyset)}\ \textsc{empty}$$

$$\frac{\begin{array}{c}F_k = (pID_a, \cdots, \mathsf{altOf}(pID), \_) \qquad F = (pID, inID, ID, \cdots, \mathsf{next}(pID_a), \_) \\ pID \notin EPids \qquad EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs) \\ inIDs' = \begin{cases} \{inID\} \cup inIDs & inID \in EIrqIds \\ inIDs & inID \notin EIrqIds \end{cases}\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (S_e, inIDs', reIDs)}\ \textsc{delay-irq}$$

$$\frac{\begin{array}{c}F_k = (pID_a, \cdots, \mathsf{altOf}(pID), \_) \qquad F = (pID, reID, ID, \cdots, \mathsf{next}(pID_a), \_) \\ pID \notin EPids \qquad EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs) \\ reIDs' = \begin{cases} \{reID\} \cup reIDs & reID \in EEvIds \\ reIDs & reID \notin EEvIds \end{cases}\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (S_e, inIDs, reIDs')}\ \textsc{delay-ev}$$

$$\frac{\begin{array}{c}F_k = (pID_a, \cdots, \mathsf{altOf}(pID), \_) \qquad F = (pID, rID, ID, \cdots, \mathsf{next}(pID_a), \_) \\ pID \in EPids \qquad EIrqIds, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (F \rhd S_e, inIDs, reIDs)}\ \textsc{drop-alt}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, \cdots, \mathsf{jit}, \_) \\ F_k = (pID, \cdots, \mathsf{jit}, \_) \qquad EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (F \rhd S_e, inIDs, reIDs)}\ \textsc{jit}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, \cdots, \mathsf{atom}(\mathsf{aID}, \cdots), \_) \\ F_k = (pID, \cdots, \mathsf{atom}(\mathsf{aID}, \cdots), \_) \qquad (pID, \mathsf{aID}) \notin EPids \\ EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (F_k \rhd S_e, inIDs, reIDs)}\ \textsc{atom-fail}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, \cdots, \mathsf{atom}(\mathsf{aID}, \cdots), \_) \\ F_k = (pID, \cdots, \mathsf{atom}(\mathsf{aID}, \cdots), \_) \qquad (pID, \mathsf{aID}) \in EPids \\ EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (F \rhd S_e, inIDs, reIDs)}\ \textsc{atom-complete}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, \cdots, \mathsf{discard}, \_) \\ pID \notin EPids \qquad EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs, reIDs)}\ \textsc{discard-fail}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, \cdots, \mathsf{discard}, \_) \qquad pID \in EPids \\ EPids, EIrqIds, EEvIds \vdash getES(S_k, S) = (S_e, inIDs, reIDs)\end{array}}{EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (F \rhd S_e, inIDs, reIDs)}\ \textsc{discard-complete}$$

Relating nonvolatile memories. To relate nonvolatile memories, we begin by extracting the set of variables X_{nid} that are most recently written to by ineffective processes or are written to by a process where the variable is declared in *nIds*.

$\boxed{\Psi, PDecl, EPids, pc \vdash N \Rightarrow X_{nid}}$

$$\frac{}{\Psi, PDecl, EPids, pc \vdash \cdot \Rightarrow \emptyset}\ \text{N-Empty}$$

$$\frac{pID_w \notin EPids \qquad \Psi, PDecl, EPids, pc \vdash N \Rightarrow X_{nid}}{\Psi, PDecl, EPids, pc \vdash N, x \mapsto (v, ipID_w) \Rightarrow X_{nid} \cup \{x\}}\ \text{N-Wt-Nid}$$

$$\frac{pID_w \in EPids \qquad \Psi, PDecl, EPids, pc \vdash N \Rightarrow X_{nid} \qquad localTyOf(\Psi, PDecl, pID_w, x) \sqcup pc = \mathsf{Nid}}{\Psi, PDecl, EPids, pc \vdash N, x \mapsto (v, ipID_w) \Rightarrow X_{nid} \cup \{x\}}\ \text{N-Wt-Eff-Nid}$$

$$\frac{pID_w \in EPids \qquad \Psi, PDecl, EPids, pc \vdash N \Rightarrow X_{nid} \qquad localTyOf(\Psi, PDecl, pID_w, x) \sqcup pc = \mathsf{Id}}{\Psi, PDecl, EPids, pc \vdash N, x \mapsto (v, ipID_w) \Rightarrow X_{nid}}\ \text{N-Wt-Eff-Id}$$

N-Wt-Nid requires that written variables in ineffective tasks be included in X_{nid}. In these rules, $ipID_w$ identifies the last process that wrote to x. The rules N-Wt-Eff-Nid and N-Wt-Eff-Id means that if the process that last wrote to x is effective, then we add x to X_{nid} depending on the local qualifier of x and the (non-)idempotence of pc. If both the local type and pc are idempotent, then N-Wt-Eff-Id and x is not inserted into the returned set, otherwise, N-Wt-Eff-Nid applies to insert x into the returned set.

With the set X_{nid} in hand for a given intermittent nonvolatile memory N_I, we define a relation between N_I and N_c. We say that N_I and N_c are related if their memories are equivalent up to their non-idempotent and ineffective writes.

$\boxed{\Psi, PDecl, EPids, pc \vdash N_I \approx N_c}$

$$\frac{\Psi, PDecl, EPids, pc \vdash N_I \Rightarrow X_{nid} \qquad N_I \backslash X_{nid} = N_c \backslash X_{nid}}{\Psi, PDecl, EPids, pc \vdash N_I \approx N_c}\ \text{Equiv-N}$$

Relating volatile memories. We define a relation between volatile memories by first extracting ineffective variables, similar to above. Since volatile data does not persist, we assume that no volatile memory location has an Nid local type. Instead, we extract only the variables written by ineffective processes in V (as in V-NEff), skipping variables that are last written by effective processes (as in V-Eff).

$\boxed{EPids \vdash V \Rightarrow X_{ineff}}$

$$\frac{}{EPids \vdash \cdot \Rightarrow \emptyset}\ \text{V-Empty} \qquad \frac{ipID_w \in EPids \qquad EPids \vdash \mathsf{V} \Rightarrow X_{ineff}}{EPids \vdash \mathsf{V}, x \mapsto (v, ipID_w) \Rightarrow X_{ineff}}\ \text{V-Eff}$$

$$\frac{ipID_w \notin EPids \qquad EPids \vdash \mathsf{V} \Rightarrow X_{ineff}}{EPids \vdash \mathsf{V}, x \mapsto (v, ipID_w) \Rightarrow X_{ineff} \cup \{x\}}\ \text{V-NEff}$$

$\boxed{EPids \vdash V_I \approx V_c}$

$$\frac{EPids \vdash V_I \Rightarrow X_{ineff} \qquad V_I \backslash X_{ineff} = V_c \backslash X_{ineff}}{EPids \vdash V_I \approx V_c}\ \text{Equiv-V}$$

Extracting idempotent stack frames. Towards the goal of relating the intermittent runtime stack with a continuously-powered runtime stack, we define a function for extracting idempotent stack frames from a current runtime stack. If the top *pc* is Id, then Proj-Frame-Id applies to return that frame. Otherwise, the rules Proj-Frame-EndIf-Nid, Proj-Frame-EndCr-Nid-1 and Proj-Frame-cont-Nid apply to inspect the next frame expected to execute.

$\boxed{S|_{id} = S'}$

$$\frac{}{\cdot|_{id} = \cdot}\ \text{Proj-Stack-empty} \qquad \frac{}{(F \rhd S)|_{id} = F|_{id} \rhd S|_{id}}\ \text{Proj-Stack-ind}$$

$\boxed{F|_{id} = F'}$

$$\frac{F = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Id} \rhd pcS, vPol, \upsilon, E, \mathtt{Md}, c)}{F|_{id} = F}\ \text{Proj-Frame-Id}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Nid} \rhd pcS, vPol, \upsilon, \mathsf{end_{if}} \rhd E, \mathtt{Md}, c) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip})\end{array}}{F|_{id} = F'|_{id}}\ \text{Proj-Frame-EndIf-Nid}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Nid} \rhd pcS, vPol, \upsilon, C \rhd E, \mathtt{Md}, c) \\ C \in \{([\,]; c'), \mathsf{end_{cr}}\} \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Nid} \rhd pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip})\end{array}}{F|_{id} = F'|_{id}}\ \text{Proj-Frame-Nid-1}$$

$$\frac{\begin{array}{c}F = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Nid} \rhd pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ E = \cdot \text{ or } E = C \rhd E' \text{ where } C \in \{([\,]; c'), \mathsf{end_{cr}}\} \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Nid} \rhd pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip})\end{array}}{F|_{id} = F'|_{id}}\ \text{Proj-Frame-Nid-2}$$

Relating configuration states Σ_j and σ. We write $T|_j \vdash \Sigma_j \approx \sigma$ if all of the following hold

- $\Sigma_j = T{\downarrow_j} = (I_j, IRQ_j, (N_k, V_k, S_k), \mathsf{N}_j, \mathsf{V}_j, S_j, F_j, Q_j)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T|_j)$
- $EIrqIds = eIrqOf(IRQ_j, T)$
- $EEvIds = eEvOf(Q_j, T)$
- $EInpIds = eInpOf(I_j, T)$
- $EPids, EIrqIds, EEvIds \vdash getES(F_j \rhd S_j, S_k) = (S_e, inIDs_d, reIDs_d)$
- $S_e|_{id} = F_c \rhd S_c$ effect stack is same as cts execution, except for Nid branches (including $\mathsf{end_{if}}$)
- $topOfPCS(F_j) = pc$
- $\Psi, PDecl, EPids, pc \vdash N_I \approx N_c$
- $\forall x, N_k(x) \neq N_c(x), N_I(x) = N_c(x)$ and $N_I(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$
- $EPids, pc \vdash V_I \approx V_c$
- $\forall x, V_k(x) \neq V_c(x), V_I(x) = V_c(x)$ and $V_I(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$
- $I_j{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$
- $IRQ_j{\downarrow_{EIrqIds}} \cup T{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$
- $Q_j{\downarrow_{EEvIds}} \cup T{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

In the definition, $T|_j$ takes the prefix of T up to the j^{th} configuration. $T{\downarrow_j}$ takes the j^{th} configuration. We $T{\downarrow_{inIDs_d}}$ and $T{\downarrow_{reIDs_d}}$ to collect the set of interrupt requests and scheduled events from the trace respectively, by their unique ids.

$eNVWtsOf(S_e)$ and $eVWtsOf(V)$ return all locations in nonvolatile memory that are currently written to in tasks that will complete in this power cycle. We define $eInpOf(I, T)$ to be the inputs that are consumed by execution of processes with effective pids in T. We define $eIrqOf(IRQ, T)$ to be the interrupts that are processed by an effective pid in T. We define $eEvOf(Q, T)$ to be the events in the queue Q that are processed by an effective pid in T.

Trace Equivalence. Now we can define trace equivalence $T_I \approx T_C$. Because we need the full intermittent trace for configuration relations, we include an index m in our definition $T_I|_m \approx T_C$ to say that these two traces are equivalent up to length m of T_I. We write $T(\Sigma_l)$ to mean that Σ_l is the last configuration of T. We define $nidIf(\Sigma_m \overset{a}{\Longrightarrow} \Sigma_{m+1})$ to be true if the step is taken by a frame whose top pc is Nid. We define analogous definitions for the continuous definition. We define $inEffective(\Sigma_m \overset{a}{\Longrightarrow} \Sigma_{m+1})$ to be steps taken by a frame, whose $ipID$ indicate that it will experience a power fail, or reboot and powerfail steps.

$\boxed{T_I|_m \approx T_C}$

$$\frac{T_I|_0 \vdash T_I{\downarrow_0} \approx \sigma}{T_I|_0 \approx \sigma}\ \textsc{TEquiv-Base}$$

$$\frac{T_I|_m \approx T_C \qquad \Sigma_{m+1} = T_I{\downarrow_{m+1}} \qquad T_I|_{m+1} = T_I|_m \overset{a}{\Longrightarrow} \Sigma_{m+1} \qquad \mathit{inEffective}(\Sigma_m \overset{a}{\Longrightarrow} \Sigma_{m+1}) \qquad T_I|_{m+1} \vdash \Sigma_{m+1} \approx T_C(\sigma_n)}{T_I|_{m+1} \approx T_C}\ \textsc{TEquiv-InEff}$$

$$\frac{T_I|_m \approx T_C \qquad \Sigma_{m+1} = T_I{\downarrow_{m+1}} \qquad T_I|_{m+1} = T_I|_m \overset{a}{\Longrightarrow} \Sigma_{m+1} \qquad \mathit{Effective}(\Sigma_m \overset{a}{\Longrightarrow} \Sigma_{m+1}) \qquad T_C(\sigma) \overset{a}{\longrightarrow} \sigma' \qquad T_I|_{m+1} \vdash \Sigma_{m+1} \approx \sigma'}{T_I|_{m+1} \approx T_C \overset{a}{\longrightarrow} \sigma'}\ \textsc{TEquiv-Eff}$$

$$\frac{T_I|_m \approx T_C \qquad \Sigma_m = T_I{\downarrow_m} \qquad T_I|_{m+1} = T_I|_m \overset{a}{\Longrightarrow} \Sigma_{m+1} \qquad \mathit{nidIf}(\Sigma_m \overset{a}{\Longrightarrow} \Sigma_{m+1}) \qquad T_I|_{m+1} \vdash \Sigma_{m+1} \approx T_C(\sigma)}{T_I|_{m+1} \approx T_C}\ \textsc{TEquiv-Nid-If1}$$

$$\frac{T_I|_m \approx T_C \qquad \Sigma_m = T_I{\downarrow_m} \qquad T_C(\sigma) \overset{a}{\longrightarrow} \sigma' \qquad \mathit{nidIf}(\sigma \overset{a}{\longrightarrow} \sigma') \qquad T_I|_m \vdash \Sigma_m \approx \sigma'}{T_I|_m \approx T_C \overset{a}{\longrightarrow} \sigma'}\ \textsc{TEquiv-Nid-If2}$$

E.9.3 Well-formedness invariants

Configuration well-formedness. WF-CONF defines the well-formedness of a configuration. A configuration is well-formed if all variables in N_k and V_k are last written by effective processes, the recovery stack that will be restored ($updJitS(S_k, F \rhd S)$) is well-formed, the runtime stack including the current frame is well-formed, and lastly, for all nonvolatile variables that were last written by a process that no longer appears in the runtime stack, if its pid is ineffective then it cannot be checkpointed by that process. We include the judgment $\Psi, PDecl \vdash \Sigma : \mathsf{ok}$ so that invariants from preservation can be used in the proof of correctness.

$\boxed{\Psi, EPids \vdash \Sigma : \mathsf{wf}}$

$$\frac{\begin{array}{c} K = (N_k, V_k, S_k) \qquad \forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids \\ \forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids \\ N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf} \\ N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf} \\ \forall x \in \mathsf{dom}(N) \text{ s.t. } N(x) = (v, ipID),\ x \notin EPids, \text{ and } ipID \notin F \rhd S, x \notin ShCPOf(PDecl, ipID) \\ \Psi, PDecl \vdash \Sigma : \mathsf{ok} \qquad \Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) \end{array}}{\Psi, EPids \vdash \Sigma : \mathsf{wf}}\ \textsc{Wf-Conf}$$

Stack well-formedness. We define the well-formedness of a stack in terms of the well-formedness of each of its frames (WF-S-EMPTY and WF-S-IND).

$\boxed{N, V, \Psi, EPids, S_h \vdash S_l : \mathsf{wf}}$

$$\frac{}{N, V, \Psi, EPids, S \vdash \cdot : \mathsf{wf}}\ \textsc{Wf-S-empty}$$

$$\frac{\begin{array}{c} N, \Psi, EPids, S_h \vdash_n F : \mathsf{wf} \\ V, \Psi, EPids, S_h \vdash_v F : \mathsf{wf} \qquad N, V, \Psi, EPids, S_h @ F \vdash S_l : \mathsf{wf} \end{array}}{N, V, \Psi, EPids, S_h \vdash F \rhd S_l : \mathsf{wf}}\ \textsc{Wf-S-ind}$$

Frame well-formedness for nonvolatile writes. A frame is well-formed w.r.t. $\vdash_n$ if for each variable that process has written (in $NVWtsOf(\mathtt{Md})$), either the last writing process is effective or the last writing process has a greater or equal priority to the current frame (in other words, it preempted the frame and overwrote its value). Additionally, the rule checks that every nonvolatile variable that is fully idempotent is a valid write (as checked by $\vdash validNvWt(S, ipID_c, EPids, ipID, x)$ – see below for definition).

$\boxed{N, \Psi, EPids, S \vdash_n F : \mathsf{wf}}$

$$\frac{\begin{array}{c} F = (pID_c, rID, ID, version, \overrightarrow{pr}, pcS, vPol, v, E, \mathtt{Md}, c) \qquad iPidOf(pID_c, \mathtt{Md}) = ipID_c \\ \forall y \in NVWtsOf(\mathtt{Md}).N(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) \geq PrioOf(pID_c) \\ \Psi(ID, version, \mathtt{Md}) = \Gamma \qquad \forall x \in \mathsf{dom}(N).s.t.\Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id}) \\ \overline{qID_c} = \mathsf{Id} \qquad N(x) = (v, ipID) \qquad \vdash validNvWt(S, ipID_c, EPids, ipID, x) \end{array}}{N, \Psi, EPids, S \vdash_n F : \mathsf{wf}}\ \textsc{Wf-F-N}$$

We check the validity of each write according to $\vdash validNvWt(S, ipID_c, EPids, ipID, x)$, where S is the runtime stack, $ipID_c$ identifies the current process, and $ipID$ is the last writing process of x.

$\boxed{\vdash validNvWt(S, ipID_c, EPids, ipID, x)}$

$$\frac{PrioOf(ipID) \leq PrioOf(ipID_c) \qquad x \in InitOf(ipID_c) \cup shCPof(ipID_c)}{\vdash validNvWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-NvWt-Init}$$

$$\frac{ipID \in EPids \cup \{ipID_c\}}{\vdash validNvWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-NvWt-E}$$

$$\frac{\begin{array}{c} ipID = (pID, \mathsf{aID}, rb) \qquad ipID_c = (pID, \mathsf{aID}, rb_c) \\ mdOf(ipID) = \mathsf{atom}(..., \omega, \mu, Inits) \qquad rb_c > rb \qquad x \in \omega \cup \mu \cup Inits \end{array}}{\vdash validNvWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-NvWt-A}$$

$$\frac{\begin{array}{c} \exists F_h \in S, F_h = (ipID, \cdots, vPol_h, \mathsf{discard}, \cdots), \\ vPol_h = (ShAcc, ShCP, nIds, Inits) \qquad x \in ShCP \end{array}}{\vdash validNvWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-NvWt-ShCP}$$

$$\frac{\begin{array}{c} ipID = (pID_h, \mathsf{aID}, rb) \qquad \exists F_h \in S, F_h = (pID_h, \cdots, vPol_h, \mathtt{Md}_h, \cdots), \\ \mathtt{Md}_h = \mathsf{atom}(\mathsf{aID}, rb_h, \omega, \mu, \rho, Inits, NVw, Vw) \\ rb_n \geq rb \qquad x \in \omega \cup \mu \cup Inits \end{array}}{\vdash validNvWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-NvWt-HA}$$

A nonvolatile write x is valid if it will be overwritten ($PrioOf(ipID) \leq PrioOf(ipID_c)$) by the current process on initialization, or if it is a shared checkpoint of it (as in Valid-NvWt-Init). Valid-Nv-Wt-E says that x is valid if its last writing process is either the current process or is effective. Valid-Nv-Wt-A says that the mutable nonvolatile variables of an atomic region ω, μ, and $Inits$ are valid, as the effects from failed prior executions ($ipID$ whose $rb < rb_c$) will eventually stabilize on the last execution. Valid-NvWt-ShCP asserts the validity of a variable written by a high priority discard task that is checkpointed. Lastly, Valid-NvWt-HA asserts that variables last written by high priority atomic regions are valid, as they will eventually stabilize.

Frame well-formedness for volatile writes. Frame well-formedness w.r.t. $\vdash_v$ is similar, but establishes the validity of all volatile writes since none are reverted to their checkpointed versions on reboot from power failure for undo logging.

$\boxed{V, \Psi, EPids, S \vdash_v F : \mathsf{wf}}$

$$\frac{\begin{array}{c} F = (pID_c, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ \forall y \in VWtsOf(\mathtt{Md}).V(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) \geq PrioOf(pID_c) \\ \forall x \in \mathsf{dom}(V) \qquad V(x) = (v, ipID) \qquad \vdash validVWt(S, ipID_c, EPids, ipID, x) \end{array}}{V, \Psi, EPids, S \vdash_v F : \mathsf{wf}}\ \textsc{Wf-F-V}$$

$\boxed{\vdash validVWt(S, ipID_c, EPids, ipID, x)}$

$$\frac{ipID \in EPids \cup \{ipID_c\}}{\vdash validVWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-VWt-E}$$

$$\frac{\exists F_h \in S, F_h = (pID_h, \cdots, vPol_h, \texttt{Md}_h, \cdots), iPidOf(pID_h, \texttt{Md}_h) = ipID}{\vdash validVWt(S, ipID_c, EPids, ipID, x)}\ \textsc{Valid-VWt}$$

We say a volatile write is valid if it is last written by an effective process or the current process (Valid-VWt-E), or is written by a higher-priority task that preempted the current frame (Valid-VWt).

E.10 Correctness

Lemma 46 (Well-formedness) *Given a trace T, $EPids = ePidOf(T)$ and $\Psi, EPids \vdash \Sigma_j : \mathsf{wf}$ where $0 \leq j \leq |T|$ and $\Psi, PDecl \vdash \Sigma_j \Longrightarrow \Sigma_{j+1}$, then $\Psi, EPids \vdash \Sigma_{j+1} : \mathsf{wf}$.*

Proof: The proof proceeds by cases on the step $\Psi, PDecl \vdash \Sigma_j \Longrightarrow \Sigma_{j+1}$. Below, we highlight the most interesting cases. In every case, we prove that $\Psi, PDecl \vdash \Sigma_{j+1} : \mathsf{ok}$ by preservation and the assumptions $\Psi, PDecl \vdash \Sigma_j \Longrightarrow \Sigma_{j+1}$ and $\Psi, PDecl \vdash \Sigma_j : \mathsf{wf}$, which implies $\Psi, PDecl \vdash \Sigma_j : \mathsf{ok}$.

Case Ret-S-I. Let $\Sigma_j = I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q$ and $\Sigma_{j+1} = I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q$, and suppose the last step was Ret-I,

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, \mathsf{ret}) \\ finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K' \\ S = F \rhd S' \qquad F = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \\ Q = \mathsf{tk}(rID_n, ID_n, v) :: Q' \qquad prOf(\overrightarrow{pr}_n) \geq pr_q \\ (\mathtt{Md}'_n, K'') = \begin{cases} schdFrame(PDecl, (pID_n, ID_n, v, rPol), K', \mathsf{N}) & \text{if } \mathtt{Md}_n = \mathsf{altOf}(rPol) \\ (\mathtt{Md}_n, K') & \text{else} \end{cases} \\ c'_n = \begin{cases} initN; c_n; \mathsf{ret} & \text{if } \mathtt{Md}_n = \mathsf{altOf}(rPol) \\ c_n & \text{else} \end{cases} \\ ipID = iPidOf(pID_n, \mathtt{Md}'_n) \qquad F' = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}'_n, c'_n) \\ o_1 = \mathsf{complete}(pID_c) \qquad o_2 = \mathsf{trigger}(rID_n, pID_n) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q \overset{o_1,o_2}{\Longrightarrow} I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q} \text{ Ret-S-I}$$

By inversion of Wf-Conf on $\Psi, EPids \vdash \Sigma_j : \mathsf{wf}$,

$$\frac{\begin{array}{c} K = (N_k, V_k, S_k) \qquad \forall x \in \mathsf{dom}(N_k). N_k(x) = (v, ipID), ipID \in EPids \\ \forall x \in \mathsf{dom}(V_k). V_k(x) = (v, ipID), ipID \in EPids \\ (N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf} \\ N, V, \Psi, EPids, \cdot \vdash F_c \rhd S : \mathsf{wf} \end{array}}{\Psi, EPids \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F_c, Q) : \mathsf{wf}} \text{ Wf-Conf}$$

we have

(i) $K = (N_k, V_k, S_k)$

(ii) $\forall x \in \mathsf{dom}(N_k). N_k(x) = (v, ipID), ipID \in EPids$

(iii) $\forall x \in \mathsf{dom}(V_k). V_k(x) = (v, ipID), ipID \in EPids$

(iv) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf}$

(v) $N, V, \Psi, EPids, \cdot \vdash F_c \rhd S : \mathsf{wf}$

Where $K'' = (N''_k, V''_k, S''_k)$, we need to show that

(1) $\forall x \in \mathsf{dom}(N_k'').N_k''(x) = (v, ipID), ipID \in EPids$

(2) $\forall x \in \mathsf{dom}(V_k'').V_k''(x) = (v, ipID), ipID \in EPids$

(3) $(N_k'' \rightsquigarrow N), (V_k'' \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k'', F' \rhd S') : \mathsf{wf}$

(4) $N, V, \Psi, EPids, \cdot \vdash F' \rhd S' : \mathsf{wf}$

We first prove (1). We consider two cases, where $N_k(x) = N_k''(x)$ or $N_k(x) \neq N_k''(x)$ for arbitrary $x \in \mathsf{dom}(N_k'')$. In the first case, if $N_k(x) = N_k''(x)$, the result follows from (ii). For the second case, if $N_k(x) \neq N_k''(x)$, we apply Lemma 60 to learn that $\mathsf{N}(x) = (v, ipID), ipID \in EPids$. By Lemma 47, we know that $\mathsf{N}_k'(x) = (v, ipID), ipID \in EPids$. By definition of *schdFrame*, we know that $\mathsf{N}_k''(x) = (v, ipID), ipID \in EPids$ since we can see that $\mathsf{N}_k'(x) = \mathsf{N}_k''(x)$. Since $x \in \mathsf{dom}(N_k'')$ was arbitrary, the desired result holds $\forall x \in \mathsf{dom}(N_k'')$.

We next prove (2). The proof of (2) is analogous. Let $x \in \mathsf{dom}(V_k'')$ be arbitrary. In the case where $V_k(x) = V_k''(x)$, the result follows directly from (iii). Otherwise, we apply Lemma 61 to learn that $\mathsf{V}(x) = (v, ipID), ipID \in EPids$, and subsequently Lemma 48 to learn that $\mathsf{V}_k'(x) = (v, ipID), ipID \in EPids$. By definition of *schdFrame*, we can see that $\mathsf{V}_k'(x) = \mathsf{V}_k''(x)$, and hence $\mathsf{V}_k''(x) = (v, ipID), ipID \in EPids$. Since $x \in \mathsf{dom}(V_k'')$ was arbitrary, the desired result holds $\forall x \in \mathsf{dom}(V_k'')$.

Towards (4), we invert WF-S-IND on (v) to learn that $N, V, \Psi, EPids, F_c \vdash S : \mathsf{wf}$. Then, we apply Lemma 57 to (v) to learn that

$$N, V, \Psi, EPids, \cdot \vdash S : \mathsf{wf}$$

By inversion of WF-S-IND on $N, V, \Psi, EPids, \cdot \vdash S : \mathsf{wf}$, and since $S = F \rhd S'$, we have that

(ix) $N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$

(x) $V, \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$

(xi) $N, V, \Psi, EPids, F \vdash S' : \mathsf{wf}$

By inversion of WF-F-N on (x),

$$\frac{\begin{array}{c} F = (pID_n, rID_n, ID_n, 0, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, E_n, \mathtt{Md}_n, c_n) \\ \forall y \in NVWtsOf(\mathtt{Md}_n).N(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) > PrioOf(pID_n) \\ \Psi(ID_n, 0, \mathtt{Md}_n) = \Gamma \qquad \forall x \in \mathsf{dom}(N).s.t.\Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id}) \\ \overline{qID_c} = \mathsf{Id} \qquad N(x) = (v, ipID) \qquad \vdash validNvWt(\cdot, ipID_n, EPids, ipID, x) \end{array}}{N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}} \text{ WF-F-N}$$

we have

(xii) $\forall y \in NVWtsOf(\mathtt{Md}_n).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) > PrioOf(pID_n)$

(xiii) $\Psi(pID_n, 0, \mathtt{Md}_n) = \Gamma$

(xiv) $\forall x \in \mathsf{dom}(N).s.t.\Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id}), \overline{qID_c} = \mathsf{Id}, N(x) = (v, ipID)$, we have $\vdash validNvWt(\cdot, ipID_n, EPids, ipID, x)$

We now need to show that F' satisifies similar conditions:

- $\forall y \in NVWtsOf(\mathtt{Md}'_n).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) > PrioOf(pID_n)$
- $\Psi(pID_n, 0, \mathtt{Md}'_n) = \Gamma$
- $\forall x \in \mathsf{dom}(N).s.t.\Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id}), \overline{qID_c} = \mathsf{Id}, N(x) = (v, ipID)$, we have $\vdash validNvWt(\cdot, ipID_n, EPids, ipID, x)$

To prove these conditions hold, we need to case on $\mathtt{Md}_n$ to determine the relation to $\mathtt{Md}'_n$ w.r.t. *schdFrame*.

Case $\mathtt{Md}_n = \mathsf{altOf}(\boldsymbol{rPol})$**.** By definition of *schdFrame*, we know that $\mathtt{Md}'_n = \mathsf{jit}$ if $appPol(rPol) = \mathsf{immediate}$, $\mathtt{Md}'_n = \mathsf{discard}$ if $appPol(rPol) = \mathsf{discard}$, and $\mathtt{Md}'_n = \mathsf{next}(\_)$ if $appPol(rPol) = \mathsf{alternative}$. Therefore, by definition of TypeCtx-altOf applied to (xiii), we know that

$$\Gamma = \Psi(ID, version, \mathtt{Md}'_n) = \Gamma$$

Therefore, $\Psi(pID_n, 0, \mathtt{Md}'_n) = \Psi(pID_n, 0, \mathtt{Md}_n) = \Gamma$, and hence all the same conditions from above apply for F'. Thus, we know that

$$N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$$

Case $\mathtt{Md}_n \neq \mathsf{altOf}(\boldsymbol{rPol})$**.** By definition of *schdFrame*, we know that $\mathtt{Md}_n = \mathtt{Md}'_n$. Then, all the other conditions hold from above, and it follows by application of Wf-F-N that

$$N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$$

In both cases, we have that $N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$ (*xvi*).

By analogous reasoning, we use the rule Wf-F-V to learn that $V, \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}$ (*xv*).

Since $\mathtt{Md}'_n = \mathtt{Md}_n$, it follows that $F = F'$, and hence it follows from (xi) that:

$$N, V, \Psi, EPids, F' \vdash S' : \mathsf{wf}$$

Hence it follows by Wf-S-ind

$$N, V, \Psi, EPids, \cdot \vdash F' \rhd S' : \mathsf{wf}$$

To prove (3), we must show that the following holds for every combination of ending and starting task policy.

$$(N''_k \rightsquigarrow N), (V''_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S''_k, F' \rhd S') : \mathsf{wf}$$

To do so, the proof proceeds by a nested casing on the mode of the ending frame ($\texttt{Md}_c$), the mode of the new frame ($\texttt{Md}_n$), and the re-execution policy of the new task (*appPol*(*rPol*), if $\texttt{Md}_n = \mathsf{altOf}(rPol)$). The updates to K in each subcase depend on the policies *finalizeK* (which depends on $\texttt{Md}_c$) and *schdFrame* (which depends on *rPol*), which inform the specifics of the condition we need to prove in each case:

Case $\texttt{Md}_c = \mathsf{jit}$**.** If $\texttt{Md}_c = \mathsf{jit}$, then it follows by definition of the checkpointed stack that $S_k = F_j \rhd S'_k$. By definition of *finalizeK*, we know that

$$finalizeK(K, F_c, \mathsf{N}, \mathsf{V}) = \mathsf{N}_k{\downarrow}_{ckVarof(S'_k)}, \mathsf{V}_k, S'_k$$

The proof now proceeds in two cases depending on $\texttt{Md}_n$, which dictates whether the checkpointed context will be further updated by the *schdFrame* function.

Case $\texttt{Md}_n \neq \mathsf{altOf}(\boldsymbol{rPol})$**.** In this case, the *schdFrame* function is not called. Hence the checkpointed context is not further updated, and $\texttt{Md}_n = \texttt{Md}'_n$. Therefore, we must show

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, F' \rhd S') : \mathsf{wf}$$

From assumption (iv), we know:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf}$$

Since $\texttt{Md}_c = \mathsf{jit}$, it follows an application of Lemma 58 (a),

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S) : \mathsf{wf}$$

Where $S = F \rhd S'$, it is either the case that the mode of F is or is not jit. We case on each of these possibilities:

Subcase $\texttt{Md}_n = \mathsf{jit}$**.** It follows by the invariant established in Theorem 18, which says that the frame on top of the checkpointed stack is always a JIT placeholder when $mdOf(F) = \mathsf{jit}$, that $S'_k = F_j(pID_n) \rhd S''_k$. Hence, it follows by definition of *updJitS* that

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F \rhd updJitS(S''_k, S') : \mathsf{wf}$$

By inversion of WF-S-IND, we learn that

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, F \vdash updJitS(S''_k, S') : \mathsf{wf}$$

By $\texttt{Md} = \texttt{Md}'_n$ in this case, it follows that $F = F'$, and hence:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, F' \vdash updJitS(S''_k, S') : \mathsf{wf}$$

By application of WF-S-IND to the above, (xv), and (xvi), we have

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F' \rhd updJitS(S''_k, S') : \mathsf{wf}$$

By definition of RET-I, $mdOf(F') = \mathsf{jit}$ and $pidOf(F') = pID_n$. Hence it follows by definition of *updJitS*:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, F' \rhd S') : \mathsf{wf}$$

Subcase $\mathtt{Md}_n \neq \mathsf{jit}$**.** Since $\mathtt{Md}_n \neq \mathsf{jit}$, it follows by definition of *updJitS* that $updJitS(S'_k, S) = updJitS(S'_k, S')$ where $S = F \rhd S'$, and hence:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S') : \mathsf{wf}$$

Since $\mathtt{Md}'_n = \mathtt{Md}_n$ in this case, it must be that $\mathtt{Md}'_n \neq \mathsf{jit}$. Hence, it follows by definition of *updJitS*:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, F' \rhd S') : \mathsf{wf}$$

Case $\mathtt{Md}_n = \mathsf{altOf}(\boldsymbol{rPol})$**.** In this case, the semantics appeal to the function *schdFrame* to further update the checkpointed context before running the new frame. Since the effects of *schdFrame* differ according to *rPol*, the proof proceeds by an additional casing on $appPol(rPol)$.

Subcase $\boldsymbol{appPol(rPol)} = \mathsf{immediate}$**.** By SCHDFRAME-IMMEDIATE, we have

$$schdFrame(...) = (\mathsf{jit}, (\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)}, \mathsf{V}_k, F_j(pID_n) \rhd S'_k))$$

We need to show that

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_j(pID_n) \rhd S'_k, F' \rhd S') : \mathsf{wf}$$

Since $mdOf(F_c) = \mathsf{jit}$, it follows by Lemma 58 (a) applied to (iv):

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S) : \mathsf{wf}\ (xvii)$$

Since $mdOf(F) \neq \mathsf{jit}$, it follows by definition of *updJitS* that $updJitS(S'_k, S) = updJitS(S'_k, S')$ (where $S = F \rhd S'$), and hence

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S') : \mathsf{wf}$$

Therefore, it follows by Lemma 54:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F' \rhd updJitS(S'_k, S') : \mathsf{wf}$$

Since $appPol(rPol) = \mathsf{immediate}$, it follows by definition of *schdFrame* that $mdOf(F') = \mathsf{jit}$. Therefore, it follows by definition of *updJitS* that $F' \rhd updJitS(S'_k, S') = updJitS(F_j(pID_n) \rhd S'_k, F' \rhd S')$. Hence,

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_j(pID_n) \rhd S'_k, F' \rhd S') : \mathsf{wf}$$

Subcase *appPol*(*rPol*) = discard. By SCHDFRAME-DISCARD, we have:

$$schdFrame(\ldots) = (\mathsf{discard}, (N_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}), V_k, S'_k)$$

Thus, we must show

$$(N_k{\downarrow}_{ckVarof(S'_k)} \leftsquigarrow N{\downarrow}_{ShCP}) \rightsquigarrow N, (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, F' \rhd S') : \mathsf{wf}$$

Since $mdOf(F_c) = \mathsf{jit}$, it follows by application of Lemma 58 (a) to (iv) that

$$(N_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S) : \mathsf{wf}$$

Since $\mathtt{Md}_n \neq \mathsf{jit}$, we know that $updJitS(S'_k, S) = updJitS(S'_k, S')$ (where $S = F \rhd S'$), and hence

$$(N_k{\downarrow}_{ckVarof(S'_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S') : \mathsf{wf}$$

Since $appPol(rPol) = \mathsf{discard}$, it follows by Lemma 53 (b):

$$(N_k{\downarrow}_{ckVarof(S'_k)} \leftsquigarrow N{\downarrow}_{ShCP}) \rightsquigarrow N, (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, F' \rhd S') : \mathsf{wf}$$

Subcase *rPol* = alternative. The logic for this case combines reasoning from each of the above cases, checkpointing task-level shared variables like discard and pushing an alternate task frame to the checkpointed stack analogous to how *schdFrame* pushes a jit placeholder for immediate tasks.

Case $\mathtt{Md}_c = \mathsf{discard}(\_)$**.** Let $K = (N_k, V_k, S_k)$ and $K' = (N'_k, V'_k, S'_k)$. By the rule FIN-K-DISCARD,

$$\frac{\begin{array}{c}\mathtt{Md} = \mathsf{discard}(ShCP, NVw, Vw) \\ F_c = (pID_c, rID_c, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}, \mathsf{ret}) \qquad K = (N_k, V_k, S_k) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N^1_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N^1_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, V, N) = K'}\ \textsc{fin-k-discard}$$

we know that

(ix) $V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}}$

(x) $N'_k = N^1_k{\downarrow_{ckVarOf(S_k)}}$ (where $N^1_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}}$)

(xi) $S'_k = S_k$

The structure of this case is similar to before; we first case on whether or not $\mathtt{Md}_n = \mathsf{altOf}(rPol)$, with an inner casing on $appPol(rPol)$.

Case $\mathtt{Md}_n \neq \mathsf{altOf}(\boldsymbol{rPol})$**.** We need to show that

$$(N'_k \rightsquigarrow N), ((V_k \leftsquigarrow V{\downarrow_{Vw}}) \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F' \rhd S') : \mathsf{wf}$$

The reasoning is similar to the $\mathtt{Md}_n \neq \mathsf{altOf}(rPol)$ subcase in the proof case above.

Case $\mathtt{Md}_n = \mathsf{altOf}(\boldsymbol{rPol})$**.** The proof proceeds in subcases on $appPol(rPol)$.

Subcase $\boldsymbol{appPol(rPol)} = \mathsf{immediate}$**.** If $appPol(rPol) = \mathsf{immediate}$, then the updates to the checkpointed context by *schdFrame* are: $N''_k = N'_k$, $V''_k = V'_k$, and $S''_k = F_j(pID_n) \rhd S_k$. Therefore, we must show that:

$$(N'_k \rightsquigarrow N), ((V_k \leftsquigarrow V{\downarrow_{Vw}}) \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S''_k, F' \rhd S') : \mathsf{wf}$$

We start with assumption (iv), which says:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf}$$

Since $\mathtt{Md}_c = \mathsf{discard}$, it follows by Lemma 58 (b):

$$(N'_k \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$$

Since $appPol(rPol) = \mathsf{immediate}$, it follows by Lemma 53 (a):

$$(N'_k \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F' \rhd updJitS(S_k, S') : \mathsf{wf}$$

Since $appPol(rPol) = \mathsf{immediate}$, it follows by definition of *schdFrame* that $\mathtt{Md}'_n = \mathsf{jit}$. Therefore, $F' \rhd updJitS(S_k, S') = updJitS(F_j(pID_n) \rhd S_k, F' \rhd S')$ and hence:

$$(N'_k \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_j(pID_n) \rhd S_k, F' \rhd S') : \mathsf{wf}$$

Subcase *appPol*(*rPol*) = discard. According to *schdFrame*, if $appPol(rPol) = \mathsf{discard}$, then $S''_k = S'_k$, $N''_k = \mathsf{N}'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}$, and $V''_k = V'_k$. Hence, we must show that

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}) \rightsquigarrow N), ((V_k \leftsquigarrow V{\downarrow}_{Vw}) \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F' \rhd S') : \mathsf{wf}$$

We begin with (iv), which says:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf}$$

Since $\mathtt{Md}_c = \mathsf{discard}$, it follows by Lemma 58 (b):

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$$

Since $S = F \rhd S'$ by definition, and $\mathtt{Md}_n = \mathsf{altOf}(rPol)$ (i.e., is not jit), it follows by definition of *updJitS* that $updJitS(S_k, S) = updJitS(S_k, S')$. Hence:

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S') : \mathsf{wf}$$

Since $appPol(rPol) = \mathsf{discard}$, it follows by Lemma 53 (b):

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F' \rhd S') : \mathsf{wf}$$

Subcase *appPol*(*rPol*) = alternative. If $appPol(rPol) = \mathsf{alternative}$, then by *schdFrame* we know that $S''_k = F_a \rhd S'_k$, $N''_k = \mathsf{N}'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}$, and $V''_k = V'_k$. Therefore, we must show:

$$\begin{aligned}&((N'_k \leftsquigarrow \mathsf{N}{\downarrow}_{ShCP}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \\ &\vdash updJitS(F_a \rhd S_k, F' \rhd S') : \mathsf{wf}\end{aligned}$$

We begin with (iv), which says:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf}$$

Since $\mathtt{Md}_c = \mathsf{discard}$, by Lemma 58 (b), it follows:

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow_{ShCP}}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$$

Since $S = F \rhd S'$ by definition, and $\mathtt{Md}_n = \mathsf{altOf}(rPol)$ (i.e., is not jit), it follows by *updJitS* that $updJitS(S_k, S) = updJitS(S_k, S')$. Hence:

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow_{ShCP}}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S') : \mathsf{wf}$$

Since $appPol(rPol) = \mathsf{alternative}$, it follows by Lemma 53 (c),

$$((N'_k \leftsquigarrow \mathsf{N}{\downarrow_{ShCP}}) \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_a \rhd S_k, F' \rhd S') : \mathsf{wf}$$

Case $\mathtt{Md}_c = \mathsf{next}(\_)$**.** The proof of this case is similar to the above two cases. After the *finalizeK* function is applied, the checkpointed stack will include one fewer frame as a result of popping the alternate.

The desired result follows by application of Wf-Conf:

$$\Psi, EPids \vdash (I, IRQ, K'', \mathsf{N}, \mathsf{V}, S', F', Q) : \mathsf{wf}$$

Case Ret-Q-I. This case is similar to the one above.

Case EndAtom-I. Let $\Sigma_j = I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q$ and $\Sigma_{j+1} = I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q$. Suppose the last rule is EndAtom-I:

$$\frac{\begin{array}{c}\mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, NVw, Vw) \\ F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{end}_{\mathsf{atom}}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip}) \qquad K = (N_k, V_k, F_k \rhd S_k) \\ F_k = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c; \mathsf{end}_{\mathsf{atom}}) \qquad V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \\ N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, F_J(pID) \rhd S_k)\end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q}\ \textsc{EndAtom-I}$$

By inversion of Wf-Conf on $\Psi, EPids \vdash \Sigma_k : \mathsf{wf}$,

$$\frac{\begin{array}{c}K = (N_k, V_k, F_k \rhd S_k) \qquad \forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids \\ \forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids \\ N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(F_k \rhd S_k, F \rhd S) : \mathsf{wf} \\ N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf}\end{array}}{\Psi, EPids \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{wf}}\ \textsc{Wf-Conf}$$

we have

(i) $K = (N_k, V_k, F_k \rhd S_k)$

(ii) $\forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids$

(iii) $\forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids$

(iv) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_k \rhd S_k, F \rhd S) : \mathsf{wf}$

(v) $N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf}$

Where $K' = (N'_k, V'_k, S'_k)$, we need to show that

(1) $(N'_k{\downarrow}_{ckVarOf(S_k)} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_J(pID) \rhd S_k, F' \rhd S) : \mathsf{wf}$

(2) $N, V, \Psi, EPids, \cdot \vdash F' \rhd S : \mathsf{wf}$

(3) $\forall x \in \mathsf{dom}(N'_k).N'_k(x) = (v, ipID), ipID \in EPids$

(4) $\forall x \in \mathsf{dom}(V'_k).V'_k(x) = (v, ipID), ipID \in EPids$

We first prove (3). We consider two cases, where $N_k(x) = N'_k(x)$ or $N_k(x) \neq N'_k(x)$ for arbitrary $x \in \mathsf{dom}(\mathsf{N}'_k)$. In the first case, if $N_k(x) = N'_k(x)$, the result follows from (ii). For the second case, if $N_k(x) \neq N'_k(x)$, we apply Lemma 60 to learn that $\mathsf{N}(x) = (v, ipID), ipID \in EPids$. Then, since $x \in \mathsf{dom}(\mathsf{N}'_k)$, it must be that $x \in \mathsf{dom}(\mathsf{N}_k)$, and hence it follows by $N'_k = N_k \leftsquigarrow N{\downarrow}_{NVw \cap \mathsf{dom}(N_k)}$ that $\mathsf{N}'_k(x) = (v, ipID)$. Since $x \in \mathsf{dom}(N'_k)$ was arbitrary, the desired result holds $\forall x \in \mathsf{dom}(N'_k)$.

We next prove (4). The proof of (4) is analogous. Let $x \in \mathsf{dom}(V'_k)$ be arbitrary. In the case where $V_k(x) = V'_k(x)$, the result follows directly from (iii). Otherwise, we apply Lemma 61 to learn that $\mathsf{V}(x) = (v, ipID), ipID \in EPids$. Then, it follows by $V'_k = V_k \leftsquigarrow V{\downarrow}_{Vw}$ that $\mathsf{V}'_k(x) = (v, ipID)$. Since $x \in \mathsf{dom}(V'_k)$ was arbitrary, the desired result holds $\forall x \in \mathsf{dom}(V'_k)$.

Towards (2), we need to prove that $N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$ and $V, \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}$. By inversion of Wf-S-ind on (v),

$$\frac{N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf} \qquad V, \Psi, EPids, \cdot \vdash_v F : \mathsf{wf} \qquad N, V, \Psi, EPids, F \vdash S : \mathsf{wf}}{N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf}}\ \textsc{Wf-S-ind}$$

we learn that

(vi) $N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$

(vii) $V, \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$

(viii) $N, V, \Psi, EPids, F \vdash S : \mathsf{wf}$

By inversion of Wf-F-N on (vi),

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ \forall y \in NVWtsOf(\mathtt{Md}).N(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) > PrioOf(pID) \\ \mathtt{Md} = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, NVw, Vw) \\ \Psi(ID, version, \mathtt{Md}) = \Gamma \qquad \forall x \in \mathsf{dom}(N).s.t.\Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id}) \\ \overline{qID_c} = \mathsf{Id} \qquad N(x) = (v, ipID_x) \qquad \vdash validNvWt(\cdot, ipID, EPids, ipID_x, x) \end{array}}{N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}} \text{ WF-F-N}$$

we learn that

(ix) $\forall y \in NVWtsOf(\mathtt{Md}).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) > PrioOf(pID)$

(x) $\Psi(ID, version, \mathsf{atom}(\ldots)) = \Gamma$

(xi) $\forall x \in \mathsf{dom}(N).s.t.\Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id}).\overline{qID_c} = \mathsf{Id}$

(xii) $N(x) = (v, ipID_x)$

(xiii) $\vdash validNvWt(\cdot, ipID, EPids, ipID_x, x)$

In order to prove that $N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$, we must show:

(a) $\forall x \in \mathsf{dom}(N).s.t.\Gamma'(x) = (qID_c, \mathsf{Id}, \mathsf{Id}).\overline{qID_c} = \mathsf{Id}$ where $\Psi(ID, version, \mathsf{jit}) = \Gamma'$ and

(b) $\vdash validNvWt(\cdot, ipID, EPids, ipID_x, x)$ where $N(x) = (v, ipID_x)$

By inversion of TypeCtx-A on (x), we know that $mkCtxAtom(\Gamma', \omega, \mu, \rho, Inits) = \Gamma$ where $\Psi(ID, version, \mathsf{atom}(\ldots)) = \Gamma'$. Since *mkCtxAtom* preserves the local idempotence of all variables in $\omega \cup \mu \cup \rho \cup Inits$, (a) is upheld.

Additionally, we observe that N remains unchanged and hence (b) is upheld from (xii) and (xiii). Noting that $NVWtsOf(\mathsf{jit}) = \cdot$, we can apply WF-F-N to obtain the desired result:

$$\frac{\begin{array}{c} F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, c) \\ \Psi(ID, version, \mathsf{jit}) = \Gamma' \qquad \forall x \in \mathsf{dom}(N).s.t.\Gamma'(x) = (qID_c, \mathsf{Id}, \mathsf{Id}) \\ \overline{qID_c} = \mathsf{Id} \qquad N(x) = (v, ipID) \qquad \vdash validNvWt(\cdot, ipID_c, EPids, ipID, x) \end{array}}{N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}} \text{ WF-F-N}$$

We can use a similar argument to establish that $V, \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}$.

By Lemma 59, we know that $N, V, \Psi, EPids, F' \vdash S : \mathsf{wf}$.

We apply WF-S-IND to learn that:

$$\frac{N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf} \qquad V, \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf} \qquad N, V, \Psi, EPids, F' \vdash S : \mathsf{wf}}{N, V, \Psi, EPids, \cdot \vdash F' \rhd S : \mathsf{wf}} \text{ WF-S-IND}$$

We now need to show that (1) holds:

$$(N'_k{\downarrow_{ckVarOf(S_k)}} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_J(pID) \rhd S_k, F' \rhd S) : \mathsf{wf}$$

We begin with (iv), we states:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_k \rhd S_k, F \rhd S) : \mathsf{wf}$$

Then, since $F = (\ldots, \mathsf{end}_{\mathsf{atom}})$ and $mdOf(F) = mdOf(F_k) = \mathsf{atom}$, it follows by Lemma 52:

$$(N'_k{\downarrow_{ckVarOf(S_k)}} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F \rhd updJitS(S_k, S) : \mathsf{wf}$$

By inversion of Wf-S-ind, we have

$$(N'_k{\downarrow_{ckVarOf(S_k)}} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, F \vdash updJitS(S_k, S) : \mathsf{wf}$$

By Lemma 59, we have

$$(N'_k{\downarrow_{ckVarOf(S_k)}} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, F' \vdash updJitS(S_k, S) : \mathsf{wf}$$

Since $N, \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$ and $V, \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}$ from above, it follows by application of Wf-S-ind that

$$(N'_k{\downarrow_{ckVarOf(S_k)}} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F' \rhd updJitS(S_k, S) : \mathsf{wf}$$

Since $mdOf(F') = \mathsf{jit}$, it follows by definition of $updJitS$ that $F' \rhd updJitS(S_k, S) = updJitS(F_J(pID) \rhd S_k, F' \rhd S)$ where $pidOf(F') = pID$, and hence:

$$(N'_k{\downarrow_{ckVarOf(S_k)}} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(F_J(pID) \rhd S_k, F' \rhd S)$$

The desired result follows by application of Wf-Conf:

$$\Psi, EPids \vdash (I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q) : \mathsf{wf}$$

Case N-Assign-I. Let $\Sigma_j = (I, IRQ, K, N, V, S, F, Q)$ and $\Sigma_{j+1} = (I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], \mathsf{V}, S, F', Q)$. Suppose that the last step was N-Assign-I:

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_0) \in PDecl \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{N}) \qquad K = (N_k, V_k, S_k) \qquad \Psi(ID_c, version_c, \mathtt{Md}_c) = \Gamma \\ \mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \Gamma(x).1 = \mathsf{Id}(\_) \\ \mathtt{Md}_c & \text{otherwise} \end{cases} \qquad K' = (N'_k, V_k, S_k) \\ N'_k = \begin{cases} N_k[x \mapsto (v, ipID_c)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ N_k & \text{else} \end{cases} \qquad ipID_c = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID_c)], \mathsf{V}, S, F', Q} \text{ N-Assign-I}$$

By inversion of WF-CONF on $\Psi, EPids \vdash \Sigma_k : \mathsf{wf}$,

$$\frac{\begin{array}{c} K = (N_k, V_k, S_k) \qquad \forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID_c), ipID_c \in EPids \\ \forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID_c), ipID_c \in EPids \\ N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf} \\ N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf} \end{array}}{\Psi, EPids \vdash (I, IRQ, K, \mathsf{N}, V, S, F, Q) : \mathsf{wf}}\ \textsc{Wf-Conf}$$

we have

(i) $K = (N_k, V_k, S_k)$

(ii) $\forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID_0), ipID_0 \in EPids$

(iii) $\forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID_0), ipID_0 \in EPids$

(iv) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf}$

(v) $N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf}$

We need to show that

(1) $\forall x \in \mathsf{dom}(N'_k).N'_k(x) = (v, ipID_0), ipID_0 \in EPids$

(2) $\forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID_0), ipID_0 \in EPids$

(3) $(N'_k \rightsquigarrow \mathsf{N}[x \mapsto (v, ipID_c)]), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F' \rhd S) : \mathsf{wf}$

(4) $\mathsf{N}[x \mapsto (v, ipID_c)], V, \Psi, EPids, \cdot \vdash F' \rhd S : \mathsf{wf}$

(2) follows directly from (iii) since V_k remains untouched.

Towards (1), we case on $\mathtt{Md}_c$ of frame F'. If $\mathtt{Md}_c \neq \mathsf{jit}$, then $N'_k = N_k$, and the result follows directly from (ii). Otherwise, if $\mathtt{Md}_c = \mathsf{jit}$, then it follows by definition of effective pids that $ipID_c \in EPids$.

Towards (3), we case on $\mathtt{Md}_c$. If $\mathtt{Md}_c = \mathsf{jit}$, then $N(x) = N'_k(x)$. Since all other locations remain unchanged, then the desired result follows by Lemma 55. Otherwise, if $\mathtt{Md}_c \neq \mathsf{jit}$, we know that $N'_k = N_k$ and hence it follows by definition of $\rightsquigarrow$ that $(N_k \rightsquigarrow N) = (N_k \rightsquigarrow N[x \mapsto (v, ipID_c)])$. Then the desired result follows by (iv).

Towards (4), we invert WF-S-IND on (v) to learn that

(vi) $N, V, \Psi, EPids, F \vdash S : \mathsf{wf}$

(vii) $N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$

(viii) $V, \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$

By inversion of WF-F-N on (vii),

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \forall y \in NVWtsOf(\mathtt{Md}_c).N(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) > PrioOf(pID_c) \\ \Psi(ID, version, \mathtt{Md}_c) = \Gamma \qquad \forall x' \in \mathsf{dom}(N).s.t.\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id}) \\ \overline{qID_c} = \mathsf{Id} \qquad N(x') = (v, ipID_0) \qquad \vdash validNvWt(\cdot, ipID_c, EPids, ipID_0, x') \end{array}}{N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}} \text{ WF-F-N}$$

we know that

- $\forall y \in NVWtsOf(\mathtt{Md}_c).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) > PrioOf(pID_c)$
- $\Psi(ID, version, \mathtt{Md}_c) = \Gamma$
- $\forall x' \in \mathsf{dom}(N).s.t.\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id}), \overline{qID_c} = \mathsf{Id}, N(x') = (v_0, ipID_0)$
- $\vdash validNvWt(\cdot, ipID_c, EPids, ipID_0, x')$

In order to establish that $N[x \mapsto (v, ipID_c)], \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$, we must show that

- $\forall y \in NVWtsOf(\mathtt{Md}'_c).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) > PrioOf(pID_c)$
- $\Psi(ID, version, \mathtt{Md}'_c) = \Gamma'$
- $\forall x' \in \mathsf{dom}(N).s.t.\Gamma'(x') = (qID_c, \mathsf{Id}, \mathsf{Id}), \overline{qID_c} = \mathsf{Id}, N[x \mapsto (v, ipID)](x') = (v', ipID')$
- $\vdash validNvWt(F, ipID_c, EPids, ipID, x')$

By definition of *TypeCtx* and $\mathtt{Md}'_c$, we know that $\Gamma = \Gamma'$. We only need to show that $\vdash validNvWt(F, ipID_c, EPids, ipID_c, x)$ in the case where $\mathtt{Md}'_c = \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}]$. If $\mathtt{Md}'_c = \mathsf{jit}$, then by definition of effective pids we know that $pID_c \in EPids$ and by Valid-NvWt-E we are done. If $\mathtt{Md}'_c = \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw)$, then since $x \in NVw$, we know by the first condition that $ipID_c \in EPids$, and hence the desired result follows by Valid-NvWt-E. Lastly, if $\mathtt{Md}_c = \mathsf{discard}$, then it must be that $x \in ShCP$, and hence the desired result follows by Valid-NvWt-ShCP by definition of $vPol_c$.

Hence, we have just shown that

$$N[x \mapsto (v, ipID_c)], \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}\ (ix)$$

We can show $V, \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}\ (x)$ by a similar argument.

By inversion of Wf-S-ind on (v), we learn that

$$N, V, \Psi, EPids, F \vdash S : \mathsf{wf}$$

By Lemma 49, we know that

$$N[x \mapsto (v, ipID_c)], V, \Psi, EPids, F' \vdash S : \mathsf{wf}\ (xi)$$

By application of Wf-S-ind to (ix), (x), and (xi), we have:

$$N[x \mapsto (v, ipID_c)], V, \Psi, EPids, \cdot \vdash F' \rhd S : \mathsf{wf}$$

Now that we have established (1), (2), (3), and (4) hold, the desired result follows by application of Wf-Conf:

$$\Psi, EPids \vdash (I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], V, S, F', Q) : \mathsf{wf}$$

Case PowerFail-I/Reboot-I. Let $\Sigma_j = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$ and $\Sigma_{j+1} = (I, IRQ, K'', N_k \rightsquigarrow \mathsf{N}, V_k, S_0, F_0, Q_k)$. Suppose that the last step was a power failure. We consider a combination of rules PowerFail-I and Reboot-I:

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, c) \\ K = (N_k, V_k, S_k) \qquad S'_k = updJitS(S_k, F \rhd S) \\ K' = (N_k, V_k, S'_k) \qquad (F_0 \rhd S_0) = incRb(S'_k) \qquad K'' = (N_k, V_k, F_0 \rhd S_0) \\ F_0 = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ polOf(F_0) \neq \mathsf{altOf}(\_) \qquad Q = Q_d, Q_k \\ \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \qquad \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K'', N_k \rightsquigarrow \mathsf{N}, V_k, S_0, F_0, Q_k}\ \textsc{PowerFail-I/Reboot-I}$$

By inversion of Wf-Conf on $\Psi, EPids \vdash \Sigma_k : \mathsf{wf}$,

$$\frac{\begin{array}{c} K = (N_k, V_k, S_k) \qquad \forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids \\ \forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids \\ N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf} \qquad N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf} \end{array}}{\Psi, EPids \vdash (I, IRQ, K, \mathsf{N}, V, S, F, Q) : \mathsf{wf}}\ \textsc{Wf-Conf}$$

we have

(i) $K = (N_k, V_k, S_k)$

(ii) $\forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids$

(iii) $\forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids$

(iv) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf}$

(v) $N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf}$

We want to show that

(1) $\forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids$

(2) $\forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids$

(3) $(N_k \rightsquigarrow (N_k \rightsquigarrow N)), (V_k \rightsquigarrow V_k), \Psi, EPids, \cdot \vdash updJitS(F_0 \rhd S_0, F_0 \rhd S_0) : \mathsf{wf}$

(4) $(N_k \rightsquigarrow N), V_k, \Psi, EPids, \cdot \vdash F_0 \rhd S_0 : \mathsf{wf}$

(1) and (2) follow directly from (ii) and (iii).

We now show that (4) holds. Since $S'_k = updJitS(S_k, F \rhd S)$, it follows by definition of S'_k that:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash S'_k : \mathsf{wf}$$

By Lemma 51 and since $(F_0 \rhd S_0) = incRb(S'_k)$, we know that

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F_0 \rhd S_0 : \mathsf{wf}$$

Since $\mathsf{dom}(V) = \mathsf{dom}(V_k)$, we know that $V_k \rightsquigarrow V = V_k$. Noting that $N_k \rightsquigarrow (N_k \rightsquigarrow N) = N_k \rightsquigarrow N$ and $V_k = V_k \rightsquigarrow V_k$, we arrive at the desired result:

$$(N_k \rightsquigarrow (N_k \rightsquigarrow N)), (V_k \rightsquigarrow V_k), \Psi, EPids, \cdot \vdash F_0 \rhd S_0 : \mathsf{wf}$$

Establishing (3) follows from the above result, the definition of *updJitS*, the definition of F, and definition of $\rightsquigarrow$. That is, by definition of *updJitS*, we know that $updJitS(F_0 \rhd S_0, F_0 \rhd S_0) = F_0 \rhd S_0$. By definition, $N_k \rightsquigarrow (N_k \rightsquigarrow N) = N_k \rightsquigarrow N$ and $V_k \rightsquigarrow V_k = V_k$, and hence the desired result follows directly from (4):

$$(N_k \rightsquigarrow (N_k \rightsquigarrow N)), (V_k \rightsquigarrow V_k), \Psi, EPids, \cdot \vdash updJitS(F_0 \rhd S_0, F_0 \rhd S_0) : \mathsf{wf}$$

As we have established (1), (2), (3), and (4) hold, the desired result follows by application of Wf-Config.

Case TriggerInt-I. Let $\Sigma_j = (I, IRQ, K, N, V, S, F, Q)$ and $\Sigma_{j+1} = (I, IRQ', K', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q)$. Suppose the last step is TriggerInt-I:

$$\frac{\begin{array}{c} IRQ = \mathsf{interrupt}(inID, iID, v) :: IRQ' \\ F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, c) \\ \mathsf{isr}(iID, 0, pr, vPol, rPol, \lambda x.c_i) \in PDecl \qquad pr > prOf(\overrightarrow{pr}_c) \\ fresh(pID) \qquad (\mathtt{Md}_n, K') = schdFrame(PDecl, (pID, iID, v, rPol), K, \mathsf{N}) \\ ipID = iPidOf(pID, \mathtt{Md}_n) \qquad vPol = (ShAcc, ShCP, nIds, Inits) \\ F_n = (pID, inID, iID, 0, pr, vPol, x \mapsto v, \cdot, \mathtt{Md}_n, initN; c_i; ret) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{trigger}(inID,pID)}{\Longrightarrow} I, IRQ', K', \mathsf{N}, \mathsf{V}, F \rhd S, F_n, Q} \text{ TriggerInt-I}$$

By inversion on Wf-Conf,

$$\frac{\begin{array}{c} K = (N_k, V_k, S_k) \qquad \forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids \\ \forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids \\ N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf} \\ N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf} \end{array}}{\Psi, EPids \vdash (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q) : \mathsf{wf}} \text{ WF-CONF}$$

we learn that

(i) $K = (N_k, V_k, S_k)$

(ii) $\forall x \in \mathsf{dom}(N_k).N_k(x) = (v, ipID), ipID \in EPids$

(iii) $\forall x \in \mathsf{dom}(V_k).V_k(x) = (v, ipID), ipID \in EPids$

(iv) $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf}$

(v) $N, V, \Psi, EPids, \cdot \vdash F \rhd S : \mathsf{wf}$

Letting $K' = (N'_k, V'_k, S'_k)$, we need to show that

(1) $\forall x \in \mathsf{dom}(N'_k).N'_k(x) = (v, ipID), ipID \in EPids$

(2) $\forall x \in \mathsf{dom}(V'_k).V'_k(x) = (v, ipID), ipID \in EPids$

(3) $N'_k \rightsquigarrow N, V'_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S'_k, F_n \rhd F \rhd S) : \mathsf{wf}$

(4) $N, V, \Psi, EPids, \cdot \vdash F_n \rhd F \rhd S : \mathsf{wf}$

We first prove (1). We consider two cases, where $N_k(x) = N'_k(x)$ or $N_k(x) \neq N'_k(x)$ for arbitrary $x \in \mathsf{dom}(N'_k)$. In the first case, if $N_k(x) = N'_k(x)$, the result follows from (ii). For the second case, if $N_k(x) \neq N'_k(x)$, we apply Lemma 60 to learn that $\mathsf{N}(x) = (v, ipID), ipID \in EPids$. By definition of *schdFrame*, we know that $\mathsf{N}'_k(x) = (v, ipID), ipID \in EPids$ since we can see that $\mathsf{N}_k(x) = \mathsf{N}'_k(x)$. Since $x \in \mathsf{dom}(N'_k)$ was arbitrary, the desired result holds $\forall x \in \mathsf{dom}(N'_k)$.

We next prove (2). The proof of (2) is analogous. Let $x \in \mathsf{dom}(V'_k)$ be arbitrary. In the case where $V_k(x) = V'_k(x)$, the result follows directly from (iii). Otherwise, we apply Lemma 61 to learn that $\mathsf{V}(x) = (v, ipID), ipID \in EPids$. By definition of *schdFrame*, we can see that $\mathsf{V}_k(x) = \mathsf{V}'_k(x)$, and hence $\mathsf{V}'_k(x) = (v, ipID), ipID \in EPids$. Since $x \in \mathsf{dom}(V'_k)$ was arbitrary, the desired result holds $\forall x \in \mathsf{dom}(V'_k)$.

We now proceed to show that (4) holds. By applying Lemma 54 to (v), we arrive at the desired result:

$$N, V, \Psi, EPids, \cdot \vdash F_n \rhd F \rhd S : \mathsf{wf}$$

To show (3), like in the RET-I case, the proof proceeds by cases on *appPol*(*rPol*) which determines what K' is produced by *schdFrame*:

Case *appPol*(*rPol*) = immediate. In this case, $N'_k = N_k$, $V'_k = V_k$, and $S'_k = (pID, \mathsf{jit}) \triangleright S_k$. Hence, we must establish that

$$N'_k \rightsquigarrow N, V'_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS((pID, \mathsf{jit}) \triangleright S_k, F_n \triangleright F \triangleright S) : \mathsf{wf}$$

By Lemma 54 applied to (iv), we know that

$$N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash F_n \triangleright updJitS(S_k, F \triangleright S) : \mathsf{wf}$$

By definition of *schdFrame*, since *appPol*(*rPol*) = immediate, it must be that $\mathtt{Md}_n$ = jit. Therefore, it follows by definition of *updJitS* that $F_n \triangleright updJitS(S_k, F \triangleright S) = updJitS((pID, \mathsf{jit}) \triangleright S_k, F_n \triangleright F \triangleright S)$ where $pidOf(F_n) = pID$, and hence:

$$N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS((pID, \mathsf{jit}) \triangleright S_k, F_n \triangleright F \triangleright S) : \mathsf{wf}$$

Since $N'_k = N_k$ and $V'_k = V_k$, we arrive at the desired result:

$$N'_k \rightsquigarrow N, V'_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS((pID, \mathsf{jit}) \triangleright S_k, F_n \triangleright F \triangleright S) : \mathsf{wf}$$

Case *appPol*(*rPol*) = discard. In this case, we know that $\mathsf{N}'_k = N_k \leftsquigarrow N{\downarrow_{ShCP}}$, $V'_k = V_k$, and $S'_k = S_k$. Hence, we must establish that

$$N'_k \rightsquigarrow N, V'_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, F_n \triangleright F \triangleright S) : \mathsf{wf}$$

The desired result is established by Lemma 53 (b) applied to (iv).

Case *appPol*(*rPol*) = alternative. In this case, we know that $\mathsf{N}'_k = N_k \leftsquigarrow N{\downarrow_{ShCP}}$, $V'_k = V_k$, and $S'_k = F_a \triangleright S_k$. Hence, we must establish that

$$N'_k \rightsquigarrow N, V'_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(F_a \triangleright S_k, F_n \triangleright F \triangleright S) : \mathsf{wf}$$

Since *appPol*(*rPol*) = alternative, the desired result follows by Lemma 53 (c) applied to (iv).

In all cases, we have established that (3) holds. Having shown that (1), (2), (3), and (4) hold, we apply Wf-Config to conclude with the desired result:

$$\Psi, EPids \vdash I, IRQ', K', \mathsf{N}, \mathsf{V}, F \triangleright S, F_n, Q : \mathsf{wf}$$

□

Lemma 47 *If finalizeK*$(K, F, \mathsf{V}, \mathsf{N}) = K'$ *and* $K' = (N_k, V_k, S_k)$*, then* $\forall x \in \mathsf{dom}(N) \cap \mathsf{dom}(N'_k)$ *s.t.* $\mathsf{N}(x) = (v, ipID)$*,* $\mathsf{N}'_k(x) = (v, ipID)$*.*

Proof: The proof is by definition of *finalizeK*. We case on *mdOf*(F). The only possible modes are next, discard and jit (which are the only possible modes at the end of a task execution):

Case *mdOf*(F) = discard The rule FIN-K-DISCARD applies:

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{discard}(NVw, Vw), \mathsf{ret}) \\ K = (N_k, V_k, S_k) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{FIN-K-DISCARD}$$

The desired result follows directly from $N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}}$, as N'_k is updated only with $N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}}$.

Case *mdOf*(F) = next The rule FIN-K-NEXT applies:

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{next}(pID_a, NVw, Vw), \mathsf{ret}) \\ K = (N_k, V_k, F_a \rhd S_k) \\ fresh(rID_a) \qquad F_a = (pID_a, rID_a, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \_, \_, \mathsf{altOf}(pID_c, \_), \_) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{FIN-K-NEXT}$$

Again, the desired result follows from $N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}}$.

Case *mdOf*(F) = jit

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{jit}, \mathsf{ret}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad K' = (N_k{\downarrow_{ckVarOf(S_k)}}, V_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{FIN-K-JIT}$$

Since $\mathsf{N}'_k = N_k{\downarrow_{ckVarOf(S_k)}}$ in this case, we need to show that $\forall x \in \mathsf{dom(N)} \cap \mathsf{dom(N}_k\mathsf{)}$, $N(x) = N_k(x)$. This result follows by the invariant established in Theorem 18.

□

Lemma 48 *If finalizeK*$(K, F, \mathsf{V}, \mathsf{N}) = K'$ *and* $K' = (N_k, V_k, S_k)$*, then* $\forall x \in \mathsf{dom}(V) \cap \mathsf{dom}(V'_k)$ *s.t.* $\mathsf{V}(x) = (v, ipID)$*, then* $\mathsf{V}'_k(x) = (v, ipID)$*.*

Proof: The proof is by definition of *finalizeK*. We case on *mdOf*(F). The only possible modes are next, discard and jit (which are the only possible modes at the end of a task execution):

Case *mdOf*(F) = discard The rule FIN-K-DISCARD applies:

$$\frac{\begin{array}{c}F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{discard}(NVw, Vw), \mathsf{ret}) \\ K = (N_k, V_k, S_k) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k)\end{array}}{finalizeK(K, F_c, N, V) = K'}\ \text{FIN-K-DISCARD}$$

The desired result follows directly from $V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}}$, as V'_k is updated only with $V{\downarrow_{Vw}}$.

Case *mdOf*$(F) = \mathsf{next}$ The rule FIN-K-NEXT applies:

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{next}(pID_a, NVw, Vw), \mathsf{ret}) \\ K = (N_k, V_k, F_a \rhd S_k) \\ fresh(rID_a) \qquad F_a = (pID_a, rID_a, ID_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \_, \_, \mathsf{altOf}(pID_c, \_), \_) \\ V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \qquad N'_k = N_k \leftsquigarrow N{\downarrow_{NVw \cap \mathsf{dom}(N_k)}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, S_k) \end{array}}{finalizeK(K, F_c, N, V) = K'} \text{ FIN-K-NEXT}$$

Again, the desired result follows from $V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}}$.

Case *mdOf*$(F) = \mathsf{jit}$

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathsf{jit}, \mathsf{ret}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad K' = (N_k{\downarrow_{ckVarOf(S_k)}}, V_k, S_k) \end{array}}{finalizeK(K, F_c, N, V) = K'} \text{ FIN-K-JIT}$$

Since $\mathsf{V}'_k = \mathsf{V}_k$ in this case, we need to show that $\forall x \in \mathsf{dom}(\mathsf{V}) \cap \mathsf{dom}(\mathsf{V}_k)$, $V(x) = V_k(x)$. This result follows by the invariant established in Theorem 18.

□

Lemma 49 (Stack well-formedness under nonvolatile writes) *If*

(i) $N, V, \Psi, EPids, F \vdash S : \mathsf{wf}$,
(ii) $F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e)$
(iii) $F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip})$
(iv) $x \in \mathsf{dom}(N)$
(v) $ipID_c = ipidOf(pID_c, \mathtt{Md}_c)$
(vi) $N, V, \upsilon \vdash e \Downarrow v$
(vii) $\mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \overline{\Gamma(x).1} = \mathsf{Id} \\ \mathtt{Md}_c & \text{otherwise} \end{cases}$

then $N[x \mapsto (v, ipID_c)], V, \Psi, EPids, F' \vdash S : \mathsf{wf}$.

Proof: The proof is by induction on the number of frames in S. Inductively, we must re-establish the well-formedness of each frame under the $\vdash_n$ and $\vdash_v$ judgments.

In order to complete the proof inductively, we must prove a generalized result: given $N, V, \Psi, EPids, F@S_h \vdash S_l : \mathsf{wf}$ (i), we must show that $N[x \mapsto (v, ipID_c)], V, \Psi, EPids, F'@S_h \vdash S_l : \mathsf{wf}$ for all S_h, S_l s.t. $S = S_h@S_l$. The proof is then by induction on the size of S_l.

Base case $S_l = \cdot$**.** If $S_1 = \cdot$, then the desired result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \rhd S_0$**.** By inversion of WF-S-IND on (i):

$$\frac{\begin{array}{c} N, \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf} \\ V, \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf} \qquad N, V, \Psi, EPids, F@S_h@F_0 \vdash S_0 : \mathsf{wf} \end{array}}{N, V, \Psi, EPids, F@S_h \vdash F_0 \rhd S_0 : \mathsf{wf}} \text{ WF-S-IND}$$

we learn that

(viii) $N, \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf}$

(ix) $V, \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf}$

(x) $N, V, \Psi, EPids, F@S_h@F_0 \vdash S_0 : \mathsf{wf}$

By application of the IH to (x), we have that

$$N[x \mapsto (v, ipID_c)], V, \Psi, EPids, F'@S_h@F_0 \vdash S_0 : \mathsf{wf}\ (xi)$$

Therefore, to complete the proof, we must show that $N[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash_n F_0 : \mathsf{wf}$ and $V, \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$.

First, $V, \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$ (xii) follows from (ix), since the mode of F_0 (and hence its Vw) remain unchanged, and V remains untouched. To re-establish the validity of each variable w.r.t. the new high stack $F'@S_h$, we choose an arbitrary $x \in \mathsf{dom}(V)$ and case on the rule that derived its *validVWt* judgment. The only interesting case is Valid-VWt. In that case, there exists a frame $F_h \in F@S_h$ that last wrote to x. If $F_h \in S_h$ alone, then we know that $F_h \in F'@S_h$ and can apply the same rule to obtain the desired result. Otherwise, $F_h = F$, but since $ipidOf(F) = ipidOf(F')$, the same conditions apply and we can apply the same rule to establish the desired result. Since $x \in \mathsf{dom}(V)$ was arbitrary, we just established the validity of all such $x \in \mathsf{dom}(V)$. Therefore, the desired result (xii) is established by applying Wf-F-V.

Towards proving that $N[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash_n F_0 : \mathsf{wf}$, we invert Wf-F-N on (viii):

$$\frac{\begin{array}{c} F_0 = (pID_0, rID, ID, version, \overrightarrow{pr}, pcS, vPol, v, E, \mathtt{Md}, c) \\ \forall y \in NVWtsOf(\mathtt{Md}).N(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) \geq PrioOf(pID_0) \\ \Psi(ID, version, \mathtt{Md}) = \Gamma \qquad \forall x' \in \mathsf{dom}(N).s.t.\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id}) \\ \overline{qID_c} = \mathsf{Id} \qquad N(x') = (v', ipID') \qquad \vdash validNvWt(S, ipID_0, EPids, ipID', x') \end{array}}{N, \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf}}\ \textsc{Wf-F-N}$$

we learn that

(xiii) $\forall y \in NVWtsOf(\mathtt{Md}).N(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) \geq PrioOf(pID_0)$

(xiv) if $\forall x' \in \mathsf{dom}(N)\ s.t.\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id}), \overline{qID_c} = \mathsf{Id}$ and $\Psi(ID, version, \mathtt{Md}) = \Gamma$ and $N(x') = (v', ipID')$, then $\vdash validNvWt(F@S_h, ipID_0, EPids, ipID', x')$

We must re-establish similar conditions for the high stack $F'@S_h$ and written variable x:

(a) $\forall y \in NVWtsOf(\mathtt{Md}).N[x \mapsto (v, ipID_c)](y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) \geq PrioOf(pID_0)$

(b) $\forall x' \in \mathsf{dom}(N[x \mapsto (v, ipID_c)])$ s.t. $\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id})$ and $\overline{qID_c} = \mathsf{Id}$, then $\vdash validNvWt(F'@S_h, ipID_0, EPids, ipID_c, x')$

For (a), note that if $x \notin \mathit{NVWtsOf}(\mathtt{Md})$, then the desired result follows directly from (xiii). Otherwise, we must show that $\mathit{ipID}_c \in \mathit{EPids}$ or $\mathit{PrioOf}(\mathit{ipID}_c) \geq \mathit{PrioOf}(\mathit{pID}_0)$. Note that the latter follows due to the well-formedness of the stack.

To prove (b), let such $z \in \mathsf{dom}(N[x \mapsto (v, \mathit{ipID}_c)]) \setminus x$ s.t. $\Gamma(z) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ and $\overline{\mathit{qID}_c} = \mathsf{Id}$ be arbitrary, and case on the deriving rule $\vdash \mathit{validNvWt}(F@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}', x')$. For the cases that do not depend on the high stack $F@S_h$, we invert and reapply the same rule. Otherwise, for Valid-NvWt-HA and Valid-NvWt-ShCP, we note that $\mathit{ipidOf}(F) = \mathit{ipidOf}(F')$, meaning that the variable policies of these processes are the same. Hence the same conditions apply for the high stack $F'@S_h$, and we reapply the same rule to establish the validity of z. Since such z was arbitrary, the result holds for all $z \in \mathsf{dom}(N[x \mapsto (v, \mathit{ipID}_c)]) \setminus x$ s.t. $\Gamma(z) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ and $\overline{\mathit{qID}_c} = \mathsf{Id}$. Now we establish the validity of the newly written x by ipID_c. If $\Gamma(x) \neq (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$ then we are done (vacuously). Otherwise, we case on the mode of F'. If $\mathit{mdOf}(F') = \mathsf{atom}$, then the result follows by Valid-NvWt-HA, since the only writable variables are those in $\omega \cup \mu \cup \mathit{Inits}$. If $\mathit{mdOf}(F') = \mathsf{jit}$, then it follows by construction of EPids that $\mathit{ipID}_c \in \mathit{EPids}$, and hence Valid-Nv-Wt-E applies. Lastly, if $\mathit{mdOf}(F') = \mathsf{discard}$, since $\Gamma(x) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$, then $x \in \mathit{ShCP}$, and hence Valid-NvWt-ShCP applies.

Now, we apply Wf-F-N to (a) and (b) to establish that $N[x \mapsto (v, \mathit{ipID}_c)], \Psi, \mathit{EPids}, F'@S_h \vdash_n F_0 : \mathsf{wf}$ (*xvi*).

Then, the desired result follows by applying Wf-S-ind to (xvi), (xii), and (xi):

$$N[x \mapsto (v, \mathit{ipID}_c)], V, \Psi, \mathit{EPids}, F'@S_h \vdash F_0 \rhd S_0 : \mathsf{wf}$$

In general, we have shown that for all S_h, S_l s.t. $S = S_h@S_l$ the following holds:

$$N[x \mapsto (v, \mathit{ipID}_c)], V, \Psi, \mathit{EPids}, F'@S_h \vdash S_l : \mathsf{wf}$$

Taking $S_h = \cdot$ and $S_l = S$, we establish the original desired result:

$$N[x \mapsto (v, \mathit{ipID}_c)], V, \Psi, \mathit{EPids}, F' \vdash S : \mathsf{wf}$$

□

Lemma 50 (Stack well-formedness under volatile writes) *If*

(i) $N, V, \Psi, \mathit{EPids}, F \vdash S : \mathsf{wf}$,
(ii) $F = (\mathit{pID}_c, \mathit{rID}_c, \mathit{ID}_c, \mathit{version}_c, \overrightarrow{pr}_c, \mathit{pcS}_c, \mathit{vPol}_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e)$
(iii) $F' = (\mathit{pID}_c, \mathit{rID}_c, \mathit{ID}_c, \mathit{version}_c, \overrightarrow{pr}_c, \mathit{pcS}_c, \mathit{vPol}_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip})$
(iv) $x \in \mathsf{dom}(V)$
(v) $\mathit{ipID}_c = \mathit{ipidOf}(\mathit{pID}_c, \mathtt{Md}_c)$
(vi) $N, V, \upsilon \vdash e \Downarrow v$
(vii) $\mathtt{Md}'_c = \mathtt{Md}_c[\mathit{Vw} \leftarrow \mathit{Vw} \cup \{x\}]$

then $N, V[x \mapsto (v, \mathit{ipID}_c)], \Psi, \mathit{EPids}, F' \vdash S : \mathsf{wf}$.

Proof: The proof is by induction on the number of frames in S. Inductively, we must re-establish the well-formedness of each frame under the $\vdash_n$ and $\vdash_v$ judgments.

In order to complete the proof inductively, we must prove a generalized result: given $N, V, \Psi, EPids, F@S_h \vdash S_l : \mathsf{wf}$ (i), we must show that $N[x \mapsto (v, ipID_c)], V, \Psi, EPids, F'@S_h \vdash S_l : \mathsf{wf}$ for all S_h, S_l s.t. $S = S_h@S_l$. The proof is then by induction on the size of S_l.

Base case $S_l = \cdot$. If $S_1 = \cdot$, then the desired result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \rhd S_0$. By inversion of WF-S-IND on (i):

$$\frac{N, \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf} \qquad V, \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf} \qquad N, V, \Psi, EPids, F@S_h@F_0 \vdash S_0 : \mathsf{wf}}{N, V, \Psi, EPids, F@S_h \vdash F_0 \rhd S_0 : \mathsf{wf}}\ \text{WF-S-IND}$$

we learn that

(viii) $N, \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf}$

(ix) $V, \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf}$

(x) $N, V, \Psi, EPids, F@S_h@F_0 \vdash S_0 : \mathsf{wf}$

We need to establish similar conditions for a high stack with F' instead of F.

By application of the IH to (x), we have that

$$N, V[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h@F_0 \vdash S_0 : \mathsf{wf}\ (xi)$$

Therefore, to complete the proof, we must show that $N, \Psi, EPids, F'@S_h \vdash_n F_0 : \mathsf{wf}$ and $V[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$.

First, $N, \Psi, EPids, F'@S_h \vdash_n F_0 : \mathsf{wf}$ (xii) follows from (ix), since the mode of F_0 (and hence its *NVw*) remain unchanged, and N remains untouched. To re-establish the validity of each variable w.r.t. the new high stack $F'@S_h$, we choose an arbitrary $x \in \mathsf{dom}(N)$ s.t. $\Gamma(x) = (q_c, \mathsf{ld}, \mathsf{ld})$ and $\overline{q_c} = \mathsf{ld}$, and case on the rule that derived its *validNVWt* judgment. The interesting cases correspond to the rules that depend on the high stack: VALID-NVWT-SHCP and VALID-NVWT-HA. In the first case, there exists a frame $F_h \in F@S_h$ corresponding to a discard task that last wrote to x, where $x \in ShCP$. If $F_h \in S_h$ alone, then we know that $F_h \in F'@S_h$ and can apply the same rule to obtain the desired result. Otherwise, $F_h = F$, but since $ipidOf(F) = ipidOf(F')$, the same conditions apply and we can apply the same rule to establish the desired result. The case for VALID-NVWT-HA is similar, but where F_h is an atomic region frame and $x \in \omega \cup \mu \cup Inits$. Using similar reasoning, we can show that $F_h \in F'@S_h$ and apply the same rule to obtain the desired result. Since $x \in \mathsf{dom}(V)$ was arbitrary, we just established the validity of all such $x \in \mathsf{dom}(V)$. Therefore, the desired result (xii) is established by applying WF-F-V.

Towards proving that $V[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$, we invert WF-F-V on (viii):

$$\frac{\begin{array}{c} F_0 = (pID_0, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \texttt{Md}, c) \\ \forall y \in VWtsOf(\texttt{Md}).V(y) = (v, ipID_y), ipID_y \in EPids \text{ or } PrioOf(ipID_y) \geq PrioOf(pID_0) \\ \forall x' \in \mathsf{dom}(V) \qquad V(x') = (v', ipID') \qquad \vdash validVWt(S, ipID_0, EPids, ipID', x') \end{array}}{V, \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf}} \text{ WF-F-V}$$

we learn that

(xiii) $\forall y \in VWtsOf(\texttt{Md}).V(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(pID_0)$

(xiv) if $\forall x' \in \mathsf{dom}(V)$ and $V(x') = (v', ipID')$, then $\vdash validVWt(F@S_h, ipID_0, EPids, ipID', x')$

We must re-establish similar conditions for the high stack $F'@S_h$ and written variable x:

(a) $\forall y \in VWtsOf(\texttt{Md}).V[x \mapsto (v, ipID_c)](y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(pID_0)$

(b) if $\forall x' \in \mathsf{dom}(V[x \mapsto (v, ipID_c)])$ and $V[x \mapsto (v, ipID_c)](x') = (v', ipID')$, then $\vdash validVWt(F@S_h, ipID_0, EPids, ipID', x')$

For (a), note that if $x \notin VWtsOf(\texttt{Md})$, then the desired result follows directly from (xiii). Otherwise, we must show that $ipID_c \in EPids$ or $PrioOf(ipID_c) \geq PrioOf(pID_0)$. Note that the latter follows due to the well-formedness of the stack.

To prove (b), pick an arbitrary $z \in \mathsf{dom}(V[x \mapsto (v, ipID_c)]) \setminus x$, and case on the deriving rule $\vdash validVWt(F@S_h, ipID_0, EPids, ipID', x')$. For the cases that do not depend on the high stack $F@S_h$, we invert and reapply the same rule. Otherwise, for Valid-VWt we note that $ipidOf(F) = ipidOf(F')$. Hence the same conditions apply for the high stack $F'@S_h$, and we reapply the same rule to establish the validity of z. Since such z was arbitrary, the result holds for all $z \in \mathsf{dom}(V[x \mapsto (v, ipID_c)]) \setminus x$ s.t. $\Gamma(z) = (q_c, \mathsf{ld}, \mathsf{ld})$ and $\overline{q_c} = \mathsf{ld}$. Now we establish the validity of the newly written x by $ipID_c$. Since $F' \in F'@S_h$ and $ipID(F') = ipID_c$, the desired result follows by Valid-VWt.

Now, we apply WF-F-V to (a) and (b) to establish that $V[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$ (*xvi*).

Then, the desired result follows by applying WF-S-ind to (xvi), (xii), and (xi):

$$N, V[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash F_0 \rhd S_0 : \mathsf{wf}$$

In general, we have shown that for all S_h, S_l s.t. $S = S_h@S_l$ the following holds:

$$N, V[x \mapsto (v, ipID_c)], \Psi, EPids, F'@S_h \vdash S_l : \mathsf{wf}$$

Taking $S_h = \cdot$ and $S_l = S$, we establish the original desired result:

$$N, V[x \mapsto (v, ipID_c)], \Psi, EPids, F' \vdash S : \mathsf{wf}$$

□

Lemma 51 (Stack well-formedness under *incRb*) *If* $N, V, \Psi, EPids, \cdot \vdash S : \mathsf{wf}$ *and* $incRb(S) = S'$, *then* $N, V, \Psi, EPids, \cdot \vdash S' : \mathsf{wf}$.

Proof: We prove a generalized result: if $N, V, \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$ and $incRb(S) = S'$ for all S_h, S_l s.t. $S = S_h@S_l$, then $N, V, \Psi, EPids, S'_h \vdash S'_l : \mathsf{wf}$ where $incRb(S_h) = S'_h$ and $incRb(S_l) = S'_l$. The proof is by induction on the size of S_l.

Base case $S_l = \cdot$**.** If $S = \cdot$, then the result holds trivially by assumption, since $incRb(\cdot) = \cdot$.

Inductive case $S_l = F_0 \rhd S_0$ ($\exists F_0 = (\boldsymbol{pID}_c, \boldsymbol{rID}_c, ID_c, \boldsymbol{version}_c, \overrightarrow{\boldsymbol{pr}}_c, \boldsymbol{pcS}_c, \boldsymbol{vPol}_c, \upsilon_c, E_c, \mathtt{Md}_c, c)$)**.** By inversion of WF-S-IND, we learn that:

(i) $N, \Psi, EPids, S_h \vdash_n F_0 : \mathsf{wf}$

(ii) $V, \Psi, EPids, S_h \vdash_v F_0 : \mathsf{wf}$

(iii) $N, V, \Psi, EPids, S_h@F_0 \vdash S_0 : \mathsf{wf}$

In order to establish the desired result, we must prove the following:

(a) $N, \Psi, EPids, S'_h \vdash_n F'_0 : \mathsf{wf}$ where $F'_0 = incRb(F_0)$

(b) $V, \Psi, EPids, S'_h \vdash_v F'_0 : \mathsf{wf}$

(c) $N, V, \Psi, EPids, S'_h@F'_0 \vdash S'_0 : \mathsf{wf}$ where $S'_0 = incRb(S_0)$

(c) is established by application of the IH to (iii). Now, it remains to show that (a) and (b) hold. It follows by definition of $incRb(F_0)$ that $F'_0 = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, c)$, where $\mathtt{Md}'_c = \mathsf{atom}(\ldots, rb + 1, \ldots)$ if $\mathtt{Md} = \mathsf{atom}(\ldots, rb, \ldots)$ and $\mathtt{Md}'_c = \mathtt{Md}_c$ otherwise. In the latter case, (a) comes directly from (i). In the former case, where $\mathtt{Md}'_c = \mathsf{atom}(\ldots, rb + 1, \ldots)$, we invert WF-F-N on (i) to learn the following:

(iv) $\forall y \in NVWtsOf(\mathtt{Md}_c).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(pID_c)$

(v) if $\Psi(ID, version, \mathtt{Md}_c) = \Gamma$ then $\forall x' \in \mathsf{dom}(N).s.t.\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{qID_c} = \mathsf{Id}$, if $N(x') = (v', ipID')$, then $\vdash validNvWt(S_h@F_0, ipID_c, EPids, ipID', x')$

We need to establish similar properties for $\mathtt{Md}'_c$:

(d) $\forall y \in NVWtsOf(\mathtt{Md}'_c).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(ipID'_c)$

(e) if $\Psi(ID, version, \mathtt{Md}'_c) = \Gamma$ then $\forall x' \in \mathsf{dom}(N).s.t.\Gamma(x') = (qID_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$, if $N(x') = (v', ipID')$, then $\vdash validNvWt(S_h@F'_0, ipID'_c, EPids, ipID', x')$

where $ipID'_c = (pID_c, \mathsf{aID}, rb_c + 1)$ and $ipID_c = (pID_c, \mathsf{aID}, rb_c + 1)$, since we know that $\mathtt{Md}'_c = \mathsf{atom}(\mathsf{aID}, \ldots, rb_c + 1, \ldots)$ and $\mathtt{Md}_c = \mathsf{atom}(\mathsf{aID}, \ldots, rb, \ldots)$.

First, we observe that $NVWtsOf(\mathtt{Md}_c) = NVWtsOf(\mathtt{Md}'_c)$, and hence (d) holds directly by (iv). Towards (e), note that $\Psi(ID, version, \mathtt{Md}'_c) = \Psi(ID, version, \mathtt{Md}_c) = \Gamma$, and let $x' \in \mathsf{dom}(N)$ where $N(x') = (v', ipID')$ be arbitrary. Since (v) holds, the proof proceeds in cases, inverting on each of the possible deriving rules: VALID-NVWT-INIT, VALID-NVWT-E, VALID-NVWT-A, VALID-NVWT-SHCP, and VALID-NVWT-HA. We only show the interesting case, which is VALID-NVWT-A which relies on *rb*.

Case VALID-NVWT-A. If VALID-NVWT-A applies, then it must be the case that $ipID' = (pID_c, \mathsf{aID}, rb')$, $rb_c > rb'$, and $x \in \omega \cup \mu \cup Inits$. Therefore, we know that $rb_c + 1 > rb'$, and hence $\vdash validNvWt(S_h@F'_0, ipID'_c, EPids, ipID', x')$ follows by application of VALID-NVWT-A to the definition of $ipID'_c$.

Case VALID-NVWT-INIT, VALID-NVWT-E, VALID-NVWT-SHCP, VALID-NVWT-HA. These cases do not rely on $ipID_c$, and hence (e) follows directly by an inversion on the given applicable rule, and an application of it while instead taking $ipID'_c$.

We have shown that (e) and (d) hold in all cases, and hence (a) follows by application of WF-F-N. The desired result follows by application of WF-S-IND to (a), (b), and (c).

□

Lemma 52 (*updJitS* well-formedness under end atomic) *If*

(i) $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(F_k \rhd S_k, F \rhd S) : \mathsf{wf}$,

(ii) $F = (\ldots, \mathsf{end}_{\mathsf{atom}})$,

(iii) $mdOf(F) = \mathsf{atom}(\mathsf{aID}, rb_f, \ldots)$,

(iv) $mdOf(F_k) = \mathsf{atom}(\mathsf{aID}, rb_k, \ldots)$ *where* $rb_f \geq rb_k$,

then $(N'_k{\downarrow}_{ckVarOf(S_k)} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F \rhd updJitS(S_k, S) : \mathsf{wf}$ *where* $V'_k = V_k \twoheadleftarrow V{\downarrow}_{Vw}$ *and* $N'_k = (N_k \twoheadleftarrow N{\downarrow}_{NVw \cap \mathsf{dom}(N_k)}){\downarrow}_{ckVarOf(S_k)}$.

Proof: In other words, we need to show that

(a) $(N'_k{\downarrow}_{ckVarOf(S_k)} \rightsquigarrow N), (V'_k \rightsquigarrow V), \Psi, EPids, F \vdash updJitS(S_k, S) : \mathsf{wf}$

(b) $(N'_k{\downarrow}_{ckVarOf(S_k)} \rightsquigarrow N), \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$

(c) $(V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$

By definition of *updJitS* applied to (i), and since from (iii) and (iv) we know that $mdOf(F_k), mdOf(F) \neq \mathsf{jit}$, we know that

$$N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash F_k \rhd updJitS(S_k, S) : \mathsf{wf}$$

By inversion of WF-S-IND, we know that

(v) $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_k \vdash updJitS(S_k, S) : \mathsf{wf}$

(vi) $N_k \rightsquigarrow N, \Psi, EPids, \cdot \vdash_n F_k : \mathsf{wf}$

(vii) $V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash_v F_k : \mathsf{wf}$

To prove (a), we proceed by induction on the size of $updJitS(S_k, S)$, relying on the assumption (v) to re-establish the well-formedness of each frame on the stack w.r.t. $\vdash_n$ and $\vdash_v$. The key observation is that the frame F_k does not write any variables, and hence does not affect the validity of any write (i.e., the validity of each write follows due to reasons that do not involve F_k). Towards (b), observe that $\forall y \in NVWts(mdOf(F)).N(y) = (v, ipID_y)$ we have that $ipID_y \in EPids$ by construction of *EPids* and (ii), i.e., F is a successfully completed atomic region. Now, we must establish that $\forall x \in \mathsf{dom}(N)$ s.t. $\Gamma(x) = (q_c, \mathsf{ld}, \mathsf{ld})$ where $\overline{q_c} = \mathsf{ld}$, that $\vdash validNvWt(\cdot, ipID_f, EPids, ipID, x)$. If $x \in NVw(mdOf(F))$, then it must be that $x \in \omega \cup \mu \cup Inits$. Since $rb_f \geq rb_k$, the result follows by VALID-NVWT-A. Otherwise, the result follows by VALID-NVWT-E, since x is either read-only or written by the preceding jit region. (c) holds by similar reasoning to (b). □

Lemma 53 (*updJitS* well-formedness under trigger) *If*

(i) $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$,

(ii) $K = (N_k, V_k, S_k)$,
(iii) $F_n = (pID_n, rID_n, ID_n, version_n, \overrightarrow{pr}_n, pcS_n, vPol_n, \upsilon_n, \cdot, \mathtt{Md}_n, initN; c_n; \mathsf{ret})$,
(iv) $(\mathtt{Md}_n, K') = schdFrame(PDecl, (pID_n, ID_n, v, rPol_), K, \mathsf{N})$,
(v) $K' = (N'_k, V'_k, S'_k)$,
(vi) $\mathsf{task}(ID_n, version_n, pr_n, vPol_n, rPol_n, \lambda x.c_n) \in PDecl$,
then $N'_k \rightsquigarrow N, V'_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash S' : \mathsf{wf}$ *where*
(a) $S' = F_n \rhd updJitS(S_k, S)$ *if* $appPol(rPol_n) = \mathsf{immediate}$
(b) $S' = updJitS(S_k, F_n \rhd S)$ *if* $appPol(rPol_n) = \mathsf{discard}$
(c) $S' = updJitS(F_a \rhd S_k, F_n \rhd S)$ *if* $appPol(rPol_n) = \mathsf{alternative}$ *and* $mdOf(F_a) = \mathsf{altOf}(pID_n)$

Proof: The proof is by cases on $appPol(rPol_n)$.

Case $appPol(rPol_n) = \mathsf{immediate}$. In particular, we must show that

$$N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash F_n \rhd updJitS(S_k, S) : \mathsf{wf}$$

To this end, it suffices to show that the following hold:

(a1) $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_n \vdash updJitS(S_k, S) : \mathsf{wf}$

(a2) $N_k \rightsquigarrow N, \Psi, EPids, \cdot \vdash_n F_n : \mathsf{wf}$

(a3) $V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash_v F_n : \mathsf{wf}$

Note that (a1) holds by an induction on the size of $updJitS(S_k, S)$. We prove the following general result: assuming $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$ for all S_h, S_l s.t. $updJitS(S_k, S) = S_h @ S_l$, then $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_n @ S_h \vdash S_l : \mathsf{wf}$.

Base case $S_l = \cdot$. The desired result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \rhd S_0$ $(\exists F_0 = (\boldsymbol{pID}_0, ID_0, \boldsymbol{version}_0, \mathtt{Md}_0, \ldots))$. By inversion of WF-S-IND on the assumption, we know that

(vii) $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, S_h @ F_0 \vdash S_0 : \mathsf{wf}$

(viii) $N_k \rightsquigarrow N, \Psi, EPids, S_h \vdash_n F_0 : \mathsf{wf}$

(ix) $V_k \rightsquigarrow V, \Psi, EPids, S_h \vdash_v F_0 : \mathsf{wf}$

By application of the IH to (vii), we establish that $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_n @ S_h @ F_0 \vdash S_0 : \mathsf{wf}$. Now, it remains to show that $N_k \rightsquigarrow N, \Psi, EPids, F_n @ S_h \vdash_n F_0 : \mathsf{wf}$ and $N_k \rightsquigarrow N, \Psi, EPids, F_n @ S_h \vdash_n F_0 : \mathsf{wf}$. Towards the former, we invert WF-F-N on (viii) to learn that

(x) $\forall y \in NVWtsOf(\mathtt{Md}_0).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(pID_c)$

(xi) $\forall x' \in \mathsf{dom}(N)$ s.t. $\Gamma(x') = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\Psi(ID_0, version_0, \mathtt{Md}_0) = \Gamma$, $\overline{q_c} = \mathsf{Id}$, and $N(x') = (v', ipID')$, $\vdash validNvWt(S_h, ipID_0, EPids, ipID', x')$

Let $x \in \mathsf{dom}(N)$ be arbitrary, where $\Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\Psi(ID, version, \mathtt{Md}) = \Gamma$, $\overline{q_c} = \mathsf{Id}$, and $N(x) = (v, ipID)$ be arbitrary. Then, proof proceeds in cases on the rule

which derived $\vdash$ $validNvWt(S_h, ipID_0, EPids, ipID, x)$. The interesting ones, Valid-NvWt-ShCP and Valid-NvWt-HA, are shown below:

Case Valid-NvWt-ShCP. By inversion of Valid-NvWt-ShCP, we know that $\exists F_h \in S_h$ s.t. $F_h = (ipID, \ldots, vPol_h, \mathsf{discard}, \ldots)$ and $vPol_h = (ShAcc, ShCP, nIds, Inits)$ and $x \in ShCP$. Therefore, the same $F_h \in F_n@S_h$. Therefore, it follows by Valid-NvWt-ShCP that $\vdash validNvWt(F_n@S_h, ipID_n, EPids, ipID, x)$.

Case Valid-NvWt-HA. The proof is similar to the above case, but relies on Valid-NvWt-HA.

Case Valid-NvWt-Init, Valid-NvWt-E, Valid-NvWt-A. The proofs of these cases proceeds by an inversion of the applicable rule on
$\vdash validNvWt(S_h, ipID_n, EPids, ipID, x)$, followed by an application of the same rule to obtain
$\vdash validNvWt(F_n@S_h, ipID_n, EPids, ipID, x)$, as these rules are independent of the stack S_h.

In all cases, we have that $\vdash validNvWt(F_n@S_h, ipID_n, EPids, ipID, x)$. Since such x was arbitrary, this result holds $\forall x_0 \in \mathsf{dom}(N)$ s.t. $\Gamma(x_0) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\Psi(ID, version, \mathtt{Md}) = \Gamma$, $\overline{q_c} = \mathsf{Id}$, and $N(x_0) = (v, ipID)$. By application of Wf-F-N to this result and (x), we have that

$$N_k \rightsquigarrow N, \Psi, EPids, F_n@S_h \vdash_n F_0 : \mathsf{wf}\ (xii)$$

We use similar reasoning to obtain that

$$V_k \rightsquigarrow V, \Psi, EPids, F_n@S_h \vdash_v F_0 : \mathsf{wf}\ (xiii)$$

By application of Wf-S-ind to $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_n@S_h@F_0 \vdash S_0 : \mathsf{wf}$, (xii), and (xiii), we establish the desired result:

$$N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_n@S_h \vdash F_0 \rhd S_0 : \mathsf{wf}$$

In general, we proved that $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_n@S_h \vdash S_l : \mathsf{wf}$. Taking $S_h = \cdot$ and $S_l = updJitS(S_k, S)$, we establish (a1).

For (a2), since $NVWtsOf(F_n) = \cdot$, it suffices to show that $\forall x \in \mathsf{dom}(N)\ s.t.\ \Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$ and $N(x) = (v, ipID)$, that $\vdash validNvWt(\cdot, ipID_n, EPids, ipID, x)$. By construction of Ψ, we know that $\Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$ is only the case if either $x \in InitOf(ipID) \cup shCPof(ipID)$ or x was written previously by an effective process. In the former, the desired result follows by application of Valid-NvWt-Init. In the latter, we know that $ipID \in EPids$, and the desired result follows by Valid-NvWt-E. We establish (a3) by similar reasoning, but there, only the Valid-NvWt-E rule can apply since *Inits* and *ShCP* are all nonvolatile.

Case $appPol(rPol_n) = \mathsf{discard}$. Since $\mathtt{Md}_n$ must be $\mathsf{discard}$, it follows by definition of *updJitS* that $updJitS(S_k, S) = updJitS(S_k, F_n \rhd S)$. Therefore, we must show that

$$(N_k \leftsquigarrow N{\downarrow_{ShCP}}) \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$$

This result holds by an induction on the size of $updJitS(S_k, S)$. Using the assumption that $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$, we can re-establish the well-formedness of each frame on the stack w.r.t. $\vdash_n$ and $\vdash_v$.

Case $\boldsymbol{appPol(rPol_n)} = \mathsf{alternative}$**.** Since $\mathtt{Md}_n$ must be next and $mdOf(F_a) \neq \mathsf{jit}$, it follows by definition of *updJitS* that $updJitS(F_a \rhd S_k, F_n \rhd S) = F_a \rhd updJitS(F_a \rhd S_k, S)$. Therefore, we must show that

$$(N_k \leftsquigarrow N{\downarrow_{ShCP}}) \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash F_a \rhd updJitS(S_k, S) : \mathsf{wf}$$

To this end, it suffices to show that the following hold:

(c1) $(N_k \leftsquigarrow N{\downarrow_{ShCP}}) \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, F_a \vdash updJitS(S_k, S) : \mathsf{wf}$

(c2) $(N_k \leftsquigarrow N{\downarrow_{ShCP}}) \rightsquigarrow N, \Psi, EPids, \cdot \vdash_n F_a : \mathsf{wf}$

(c3) $V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash_v F_a : \mathsf{wf}$

Note that (c1) holds by an induction on the size of $updJitS(S_k, S)$. Using the assumption that $N_k \rightsquigarrow N, V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$, we can re-establish the well-formedness of each frame on the stack w.r.t. $\vdash_n$ and $\vdash_v$ by induction on the size of $updJitS(S_k, S)$ (like the discard case). (c2) and (c3) are established in a similar manner to (a2) and (a3) in the $appPol(rPol_n) = \mathsf{immediate}$ case from above.

□

Lemma 54 (Stack well-formedness under trigger) *If*

(i) $N, V, \Psi, EPids, \cdot \vdash S : \mathsf{wf}$,

(ii) $\mathsf{task}(ID_n, version_n, pr_n, vPol_n, rPol_n, \lambda x.c_n) \in PDecl$,

(iii) *fresh* pID_n, rID_n

(iv) $\mathtt{Md}_n = \begin{cases} \mathsf{jit} & \text{if } appPol(rPol_n) = \mathsf{immediate} \\ \mathsf{discard}(\cdot,\cdot) & \text{if } appPol(rPol_n) = \mathsf{discard} \\ \mathsf{next}(pID_a, \cdot, \cdot)\ (\exists pID_a) & \text{else} \end{cases}$

(v) $F_n = (pID_n, rID_n, ID_n, version_n, pr_n, pcS_n, vPol_n, \upsilon_n, \cdot, \mathtt{Md}_n, initN; c_n; \mathsf{ret})$

then $N, V, \Psi, EPids, \cdot \vdash F_n \rhd S : \mathsf{wf}$.

Proof: We must show that

$$N, V, \Psi, EPids, \cdot \vdash F_n \rhd S : \mathsf{wf}$$

To this end, it suffices to show that the following hold:

(a1) $N, V, \Psi, EPids, F_n \vdash S : \mathsf{wf}$

(a2) $N, \Psi, EPids, \cdot \vdash_n F_n : \mathsf{wf}$

(a3) $V, \Psi, EPids, \cdot \vdash_v F_n : \mathsf{wf}$

To prove (a1), must inductively re-establish the well-formedness of every frame on the stack S. We prove the following general result: assuming $N, V, \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$ for all S_h, S_l s.t. $S = S_h @ S_l$, then $N, V, \Psi, EPids, F_n @ S_h \vdash S_l : \mathsf{wf}$.

Base case $S_l = \cdot$. The desired result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \rhd S_0$ $(\exists F_0 = (\boldsymbol{pID}_0, ID_0, \boldsymbol{version}_0, \mathtt{Md}_0, \ldots))$. By inversion of WF-S-IND on the assumption, we know that

(vii) $N, V, \Psi, EPids, S_h @ F_0 \vdash S_0 : \mathsf{wf}$

(viii) $N, \Psi, EPids, S_h \vdash_n F_0 : \mathsf{wf}$

(ix) $V, \Psi, EPids, S_h \vdash_v F_0 : \mathsf{wf}$

We must re-establish similar conditions but for the high stacks $F_n @ S_h$ and $F_n @ S_h @ F_0$. By application of the IH to (vii), we establish that $N, V, \Psi, EPids, F_n @ S_h @ F_0 \vdash S_0 : \mathsf{wf}$. Now, it remains to show that $N, \Psi, EPids, F_n @ S_h \vdash_n F_0 : \mathsf{wf}$ and $V, \Psi, EPids, F_n @ S_h \vdash_v F_0 : \mathsf{wf}$. Towards the former, we invert WF-F-N on (viii) to learn that

(x) $\forall y \in NVWtsOf(\mathtt{Md}_0).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(pID_c)$

(xi) $\forall x' \in \mathsf{dom}(N)$ s.t. $\Gamma(x') = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\Psi(ID_0, version_0, \mathtt{Md}_0) = \Gamma$, $\overline{q_c} = \mathsf{Id}$, and $N(x') = (v', ipID'), \vdash validNvWt(S_h, ipID_0, EPids, ipID', x')$

The crucial next step is that using (xi) we must re-establish the validity of every write given the high stack $F_n @ S_h$. To this end, let $x \in \mathsf{dom}(N)$ be arbitrary, where $\Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\Psi(ID, version, \mathtt{Md}) = \Gamma$, $\overline{q_c} = \mathsf{Id}$, and $N(x) = (v, ipID)$ be arbitrary. Then, the proof proceeds in cases on the rule which derived $\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$. The interesting ones, VALID-NVWT-SHCP and VALID-NVWT-HA, are shown below:

Case VALID-NVWT-SHCP. By inversion of VALID-NVWT-SHCP, we know that $\exists F_h \in S_h$ s.t. $F_h = (ipID, \ldots, vPol_h, \mathsf{discard}, \ldots)$ and $vPol_h = (ShAcc, ShCP, nIds, Inits)$ and $x \in ShCP$. Therefore, the same $F_h \in F_n @ S_h$. Therefore, it follows by VALID-NVWT-SHCP that $\vdash validNvWt(F_n @ S_h, ipID_0, EPids, ipID, x)$.

Case VALID-NVWT-HA. The proof is similar to the above case, but relies on VALID-NVWT-HA.

Case VALID-NVWT-INIT, VALID-NVWT-E, VALID-NVWT-A. The proofs of these cases proceed by an inversion of the applicable rule on $\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$, followed by an application of the same rule to obtain $\vdash validNvWt(F_n @ S_h, ipID_0, EPids, ipID, x)$, as these rules are all independent of the high stack S_h.

In all cases, we have that $\vdash validNvWt(F_n @ S_h, ipID_0, EPids, ipID, x)$. Since such x was arbitrary, this result holds $\forall x' \in \mathsf{dom}(N)$ s.t. $\Gamma(x') = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\Psi(ID, version, \mathtt{Md}) = \Gamma$, $\overline{q_c} = \mathsf{Id}$, and $N(x') = (v', ipID')$. By application of WF-F-N to this result and (x), we have that

$$N, \Psi, EPids, F_n @ S_h \vdash_n F_0 : \mathsf{wf} \; (xii)$$

We use similar reasoning to obtain that

$$V, \Psi, EPids, F_n@S_h \vdash_v F_0 : \mathsf{wf}\ (xiii)$$

By application of WF-S-IND to $N, V, \Psi, EPids, F_n@S_h@F_0 \vdash S_0 : \mathsf{wf}$, (xii), and (xiii), we establish the desired result:

$$N, V, \Psi, EPids, F_n@S_h \vdash F_0 \rhd S_0 : \mathsf{wf}$$

We just proved the general result that $N, V, \Psi, EPids, F_n@S_h \vdash S_l : \mathsf{wf}$. Taking $S_h = \cdot$ and $S_l = S$, we establish (a1).

For (a2), since $NVWtsOf(F_n) = \cdot$, it suffices to show that $\forall x \in \mathsf{dom}(N)\ s.t.\ \Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$ and $N(x) = (v, ipID)$, that $\vdash validNvWt(\cdot, ipID_n, EPids, ipID, x)$. By construction of Ψ, we know that $\Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$ is only the case if either $x \in InitOf(ipID) \cup shCPof(ipID)$ or x was written previously by an effective process. In the former, the desired result follows by application of VALID-NVWT-INIT. In the latter, we know that $ipID \in EPids$, and the desired result follows by VALID-NVWT-E. We establish (a3) by similar reasoning, but there, only the VALID-NVWT-E rule can apply since *Inits* and *ShCP* are all nonvolatile.

The desired result then follows by application of WF-S-IND to (a1), (a2), and (a3). □

Lemma 55 (*updJitS* well-formedness under jit nonvolatile updates) *If*

(i) $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf}$

(ii) $F = (pID, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathsf{jit}, x := e)$

(iii) $F' = (pID, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathsf{jit}, \mathsf{skip})$

(iv) $x \in \mathsf{dom}(N)$

then $(N_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{N}[x \mapsto (v, pID)]), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F' \rhd S) : \mathsf{wf}$ *where* $S_k = F_J(pID) \rhd S'_k$.

Proof: Since $mdOf(F) = mdOf(F') = \mathsf{jit}$, it follows that $updJitS(S_k, F \rhd S) = F \rhd updJitS(S'_k, S)$ and $updJitS(S_k, F' \rhd S) = F' \rhd updJitS(S'_k, S)$. Therefore, it suffices to prove that if

$$(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F \rhd updJitS(S'_k, S) : \mathsf{wf}\ (v)$$

then

$$(N_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{N}[x \mapsto (v, pID)]), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F' \rhd updJitS(S'_k, S) : \mathsf{wf}.$$

By inversion of WF-S-IND on (v), we learn that

(vi) $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, F \vdash updJitS(S'_k, S) : \mathsf{wf}$

(vii) $(N_k \rightsquigarrow \mathsf{N}), \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$

(viii) $(V_k \rightsquigarrow \mathsf{V}), \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$

We need to establish the following conditions:

(a) $(N_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{N}[x \mapsto (v, pID)]), (V_k \rightsquigarrow V), \Psi, EPids, F' \vdash updJitS(S'_k, S) : \mathsf{wf}$

(b) $(N_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{N}[x \mapsto (v, pID)]), \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$

(c) $(V_k \rightsquigarrow \mathsf{V}), \Psi, \mathit{EPids}, \cdot \vdash_v F' : \mathsf{wf}$

Towards proving (b), we invert WF-F-N on (vii) to learn the following:

(ix) $\forall y \in \mathit{NVWtsOf}(\mathsf{jit}).N(y) = (v, \mathit{ipID}_y), \mathit{ipID}_y \in \mathit{EPids}$ or $\mathit{PrioOf}(\mathit{ipID}_y) \geq \mathit{PrioOf}(\mathit{iPidOf}(F_0))$

(x) $\forall z \in \mathsf{dom}(N')$ s.t. $\Gamma(z) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$ and $\Psi(\mathit{ID}_c, \mathit{version}_c, \mathsf{jit}) = \Gamma$ and $N'(z) = (v_z, \mathit{ipID}_z), \vdash \mathit{validNvWt}(\cdot, \mathit{pID}, \mathit{EPids}, \mathit{ipID}_z, z)$ and $N' = N_k \rightsquigarrow \mathsf{N}$

We must establish similar conditions for F'. Noting that $\mathit{NVWtsOf}(\mathsf{jit}) = \cdot$, we establish the analogous first condition for F' vacuously. Towards the second condition, we must show that if $\Gamma(x) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$ and $\Psi(\mathit{ID}_c, \mathit{version}_c, \mathsf{jit})$, that $\vdash \mathit{validNvWt}(\cdot, \mathit{pID}, \mathit{EPids}, \mathit{pID}, x)$.

By definition of EPids, we know that $\mathit{pID} \in \mathit{EPids}$ since pID is a jit process. Therefore, the desired result follows by application of VALID-NVWT-E, and hence it follows from (x) that $\forall z \in \mathsf{dom}(N[x \mapsto (v, \mathit{pID})])$ s.t. $\Gamma(z) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$ and $\Psi(\mathit{ID}_c, \mathit{version}_c, \mathsf{jit}) = \Gamma$ and $N(z) = (v_z, \mathit{ipID}_z), \vdash \mathit{validNvWt}(\cdot, \mathit{pID}, \mathit{EPids}, \mathit{ipID}_z, z)$ (xi). Therefore, it follows by WF-F-N applied to (xi) that

$$(N_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow \mathsf{N}[x \mapsto (v, \mathit{pID})]), \Psi, \mathit{EPids}, \cdot \vdash_n F' : \mathsf{wf}$$

The proof of (c) is similar but easier. Since $V_k \rightsquigarrow V$ does not change between assumption and what needs to be proved, we invert on WF-F-V to learn that every $x \in \mathsf{dom}(V_k \rightsquigarrow V)$ is a valid write. Since $\mathit{pidOf}(F) = \mathit{pidOf}(F') = \mathit{pID}$, a similar condition holds for F', and since $\mathit{Vw}(\mathsf{jit}) = \cdot$, the other condition holds vacuously. We obtain the following desired result by application of WF-F-V to these conditions:

$$(V_k \rightsquigarrow \mathsf{V}), \Psi, \mathit{EPids}, \cdot \vdash_v F' : \mathsf{wf}$$

Lastly, we must show that (a) holds. We prove (a) by re-establishing the well-formedness of every frame on the stack $\mathit{updJitS}(S'_k, S)$.

We prove a generalized result, that if $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, \mathit{EPids}, F@S_h \vdash S_u : \mathsf{wf}$, then $(N_k[x \mapsto (v, \mathit{ipID})] \rightsquigarrow \mathsf{N}[x \mapsto (v, \mathit{ipID})]), (V_k \rightsquigarrow V), \Psi, \mathit{EPids}, F'@S_h \vdash S_u : \mathsf{wf}$ for all S_h, S_u where $\mathit{updJitS}(S'_k, S) = S_h@S_u$. The proof is by induction on the size of S_u.

Base case $S_u = \cdot$. Then the desired result follows by WF-S-EMPTY.

Inductive case $S_u = F_0 \rhd S_0$ $(\exists F_0 = (\ldots, \mathit{ID}_0, \boldsymbol{\mathit{version}}_0, \mathtt{Md}_0, \ldots))$.

By inversion of WF-S-IND on the assumption, we know that

(xii) $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, \mathit{EPids}, F@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(xiii) $(N_k \rightsquigarrow \mathsf{N}), \Psi, \mathit{EPids}, F@S_h \vdash_n F_0 : \mathsf{wf}$

(xiv) $(V_k \rightsquigarrow V), \Psi, \mathit{EPids}, F@S_h \vdash_v F_0 : \mathsf{wf}$

We need to show that the following hold:

(a1) $(N_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow \mathsf{N}[x \mapsto (v, \mathit{pID})]), (V_k \rightsquigarrow V), \Psi, \mathit{EPids}, F'@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(a2) $(N_k[x \mapsto (v, \mathit{ipID})] \rightsquigarrow \mathsf{N}[x \mapsto (v, \mathit{ipID})]), \Psi, \mathit{EPids}, F'@S_h \vdash_n F_0 : \mathsf{wf}$

(a3) $(V_k \rightsquigarrow V), \Psi, \mathit{EPids}, F'@S_h \vdash_v F_0 : \mathsf{wf}$

(a1) follows by the IH applied to (xii).

Towards proving (a2), we invert WF-F-N on (xiii) to learn the following, where $N' = N_k \rightsquigarrow \mathsf{N}$:

(xv) $\forall y \in \mathit{NVWtsOf}(\mathit{modeOf}(F_0)).N'(y) = (v, \mathit{ipID}_y)$,
$\mathit{ipID}_y \in \mathit{EPids}$ or $\mathit{PrioOf}(\mathit{ipID}_y) \geq \mathit{qIDPrioOf}(\mathit{iPidOf}(F_0))$

(xvi) $\forall z \in \mathsf{dom}(N')$ s.t. $\Gamma(z) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$ and $\Psi(\mathit{ID}_0, \mathit{version}_0, \mathtt{Md}_0)$ and $N'(z) = (v_z, \mathit{ipID}_z)$, $\vdash \mathit{validNvWt}(F@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}_z, z)$.

We must establish that similar conditions hold for nonvolatile memory $N_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow \mathsf{N}[x \mapsto (v, \mathit{pID})]$. Since pID is a jit process, it follows by construction of EPids that $\mathit{pID} \in \mathit{EPids}$. Hence, it follows from (xv) that

$$\forall y \in \mathit{NVWtsOf}(\mathit{modeOf}(F_0)).N'(y) = (v, \mathit{ipID}_y), \quad \mathit{ipID}_y \in \mathit{EPids} \text{ or } \mathit{PrioOf}(\mathit{ipID}_y) \geq \mathit{PrioOf}(\mathit{iPidOf}(F_0)) \quad (\mathit{xvii})$$

If $\Gamma(x) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$ and $\Psi(\mathit{ID}_0, \mathit{version}_0, \mathtt{Md}_0)$, then we need to show that $\vdash \mathit{validNvWt}(F'@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{pID}, x)$. Again, by construction we know that $\mathit{pID} \in \mathit{EPids}$ since it is a jit process, and hence, the result follows by Valid-NvWt-E.

To re-establish the validity of all other writes wrt the new high stack $F'@S_h$, we let $z' \in \mathsf{dom}(N')$ be arbitrary s.t. $\Gamma(z') = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$, $\overline{\mathit{qID}_c} = \mathsf{Id}$, and $N'(z') = (v_{z'}, \mathit{ipID}_{z'})$, and we invert on the rule used to derive $\vdash \mathit{validNvWt}(F@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}_z, z')$. In the cases where Valid-NvWt-Init, Valid-NvWt-E, and Valid-NvWt-A (which are independent of the high stack), we simply reapply the same rule to obtain the desired result. Otherwise, if Valid-NvWt-ShCP or Valid-NvWt-HA were applied, we note that the high frame F_h that wrote x must be in S_h alone as the $\mathit{mdOf}(F) = \mathsf{jit}$ (i.e., is not $\mathsf{discard}$ or atom). Therefore, $F_h \in F'@S_h$, and it follows by applying the same rule that $\mathit{validNvWt}(F'@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}_z, z')$. Since such z' was arbitrary, this result holds for all such z'.

Overall, we have just established that this condition holds $\forall z \in \mathsf{dom}(N'')$ *s.t.* $\Gamma(z) = (\mathit{qID}_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{\mathit{qID}_c} = \mathsf{Id}$, $\Psi(\mathit{ID}_0, \mathit{version}_0, \mathtt{Md}_0)$, $N''(z) = (v_z, \mathit{ipID}_z)$, and $N'' = N_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow \mathsf{N}[x \mapsto (v, \mathit{pID})]$.

By applying Wf-F-N to this result and (xv), we retrieve (a2).

The reasoning for (a3) is analogous. By inverting Wf-F-V on (xiv), we learn the following where $V' = V_k \rightsquigarrow V$:

(xviii) $\forall y \in \mathit{VWtsOf}(\mathit{modeOf}(F_0)).V'(y) = (v, \mathit{ipID}_y)$,
$\mathit{ipID}_y \in \mathit{EPids}$ or $\mathit{PrioOf}(\mathit{ipID}_y) \geq \mathit{PrioOf}(\mathit{iPidOf}(F_0))$

(xix) $\forall z \in \mathsf{dom}(V')$ s.t. $V'(z) = (v_z, \mathit{ipID}_z)$, $\vdash \mathit{validVWt}(F@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}_z, z)$.

To re-establish similar conditions for the high stack $F'@S_h$, we again pick an arbitrary z and invert (xix) on the deriving rule Valid-VWt-E or Valid-VWt. In the first case, we apply the same rule to the premise to obtain the desired result. In the latter, we note that if the high frame F_h that last wrote to z is F, then note that $\mathit{pidOf}(F) = \mathit{pidOf}(F')$, i.e., the process of F is the same as F'. Therefore, the condition holds for stack $F'@S_h$. By applying Wf-F-V to this result and (xviii), we establish (a3).

Applying Wf-S-ind to (a1), (a2), and (a3), we have that for all S_h, S_u s.t. $S_h@S_u = \mathit{updJitS}(S'_k, S)$:

$$N_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow N[x \mapsto (v, \mathit{pID})], V_k \rightsquigarrow V, \Psi, \mathit{EPids}, F'@S_h \vdash S_u$$

Taking $S_h = \cdot$ and $S_u = updJitS(S'_k, S)$, we obtain (c):

$$N_k[x \mapsto (v, pID)] \rightsquigarrow N[x \mapsto (v, pID)], V_k \rightsquigarrow V, \Psi, EPids, F' \vdash updJitS(S'_k, S)$$

Therefore, by applying WF-S-IND to (a), (b), (c), we arrive at the desired result.

$$N_k[x \mapsto (v, pID)] \rightsquigarrow N[x \mapsto (v, pID)], V_k \rightsquigarrow V, \Psi, EPids, \cdot \vdash F' \rhd updJitS(S'_k, S)$$

□

Lemma 56 (*updJitS* well-formedness under jit volatile updates) *If*

(i) $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F \rhd S) : \mathsf{wf}$

(ii) $F = (pID, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathsf{jit}, x := e)$

(iii) $F' = (pID, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathsf{jit}, \mathsf{skip})$

(iv) $x \in \mathsf{dom}(V)$

then $(N_k \rightsquigarrow \mathsf{N}), (V_k[x \mapsto (v, ipID)] \rightsquigarrow V[x \mapsto (v, ipID)]), \Psi, EPids, \cdot \vdash updJitS(S_k, F' \rhd S) : \mathsf{wf}$.

Proof: Since $mdOf(F) = mdOf(F') = \mathsf{jit}$, it follows that $updJitS(S_k, F \rhd S) = F \rhd updJitS(S'_k, S)$ and $updJitS(S_k, F' \rhd S) = F' \rhd updJitS(S'_k, S)$. Therefore, it suffices to prove that if

$$(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F \rhd updJitS(S'_k, S) : \mathsf{wf}\ (v)$$

then

$$(N_k \rightsquigarrow \mathsf{N}), (V_k[x \mapsto (v, pID)] \rightsquigarrow V[x \mapsto (v, pID)]), \Psi, EPids, \cdot \vdash F' \rhd updJitS(S'_k, S) : \mathsf{wf}.$$

By inversion of WF-S-IND on (v), we learn that

(vi) $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, F \vdash updJitS(S'_k, S) : \mathsf{wf}$

(vii) $(N_k \rightsquigarrow \mathsf{N}), \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$

(viii) $(V_k \rightsquigarrow \mathsf{V}), \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$

We need to establish the following conditions:

(a) $(N_k \rightsquigarrow \mathsf{N}), (V_k[x \mapsto (v, pID)] \rightsquigarrow V[x \mapsto (v, pID)]), \Psi, EPids, F' \vdash updJitS(S'_k, S) : \mathsf{wf}$

(b) $(N_k \rightsquigarrow \mathsf{N}), \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$

(c) $(V_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{V}[x \mapsto (v, pID)]), \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}$

Towards (c), we invert WF-F-V on (viii) to learn the following:

(ix) $\forall y \in VWtsOf(\mathsf{jit}).N(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(iPidOf(F_0))$

(x) $\forall z \in \mathsf{dom}(V')$ s.t. $V'(z) = (v_z, ipID_z), \vdash validVWt(\cdot, pID, EPids, ipID_z, z)$ and $V' = V_k \rightsquigarrow \mathsf{V}$

We must establish similar conditions for F'. Noting that $VWtsOf(\mathsf{jit}) = \cdot$, we establish the analogous first condition for F' vacuously. Towards the second condition, we must show that $\vdash validVWt(\cdot, pID, EPids, pID, x)$.

By definition of *EPids*, we know that $pID \in EPids$ since pID is a jit process. Therefore, the desired result follows by application of Valid-VWt-E, and hence it follows from (x) that $\forall z \in \mathsf{dom}(V_k[x \mapsto (v, ipID)] \rightsquigarrow \mathsf{V}[x \mapsto (v, ipID)]), \vdash validVWt(\cdot, pID, EPids, ipID_z, z)$ (*xi*). Therefore, it follows by Wf-F-V applied to (xi) that

$$(V_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{V}[x \mapsto (v, pID)]), \Psi, EPids, \cdot \vdash_v F' : \mathsf{wf}$$

The proof of (b) is similar but easier. Since $N_k \rightsquigarrow N$ does not change between assumption and what needs to be proved, we invert on Wf-F-N to learn that every $x \in \mathsf{dom}(N_k \rightsquigarrow N)$ is a valid write. Since $pidOf(F) = pidOf(F') = pID$, a similar condition holds for F', and since $NVw(\text{jit}) = \cdot$, the other condition holds vacuously. We obtain the following desired result by application of Wf-F-N to these conditions:

$$(N_k \rightsquigarrow \mathsf{N}), \Psi, EPids, \cdot \vdash_n F' : \mathsf{wf}$$

Lastly, we must show that (a) holds. We prove (a) by re-establishing the well-formedness of every frame on the stack $updJitS(S'_k, S)$.

We prove a generalized result, that if $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, F@S_h \vdash S_u : \mathsf{wf}$, then $(N_k \rightsquigarrow \mathsf{N}), (V_k[x \mapsto (v, ipID)] \rightsquigarrow V[x \mapsto (v, ipID)]), \Psi, EPids, F'@S_h \vdash S_u : \mathsf{wf}$ for all S_h, S_u s.t. $updJitS(S'_k, S) = S_h@S_u$. The proof is by induction on the size of S_u.

Base case $S_u = \cdot$**.** Then the desired result follows by Wf-S-empty.

Inductive case $S_u = F_0 \rhd S_0$ $(\exists F_0 = (\ldots, ID_0, \textbf{\textit{version}}_0, \mathtt{Md}_0, \ldots))$**.**

By inversion of Wf-S-ind on the assumption, we know that

(xii) $(N_k \rightsquigarrow \mathsf{N}), (V_k \rightsquigarrow V), \Psi, EPids, F@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(xiii) $(N_k \rightsquigarrow \mathsf{N}), \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf}$

(xiv) $(V_k \rightsquigarrow V), \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf}$

We need to show that the following hold:

(a1) $(N_k \rightsquigarrow \mathsf{N}), (V_k[x \mapsto (v, pID)] \rightsquigarrow V[x \mapsto (v, pID)]), \Psi, EPids, F'@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(a2) $(N_k \rightsquigarrow \mathsf{N}), \Psi, EPids, F'@S_h \vdash_n F_0 : \mathsf{wf}$

(a3) $(V_k[x \mapsto (v, pID)] \rightsquigarrow V[x \mapsto (v, pID)]), \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$

(a1) follows by the IH applied to (xii).

Towards proving (a3), we invert Wf-F-V on (xiv) to learn the following, where $V' = V_k \rightsquigarrow \mathsf{V}$:

(xv) $\forall y \in VWtsOf(modeOf(F_0)).N'(y) = (v, ipID_y), ipID_y \in EPids$ or $PrioOf(ipID_y) \geq PrioOf(iPidOf(F_0))$

(xvi) $\forall z \in \mathsf{dom}(V')$ s.t. $V'(z) = (v_z, ipID_z), \vdash validNvWt(F@S_h, ipID_0, EPids, ipID_z, z)$.

We must establish that similar conditions hold for the updated memory $V'' = V_k[x \mapsto (v, pID)] \rightsquigarrow \mathsf{V}[x \mapsto (v, pID)]$. Since pID is a jit process, it follows by construction of *EPids* that $pID \in EPids$. Thus, if $x \in VWtsOf(modeOf(F_0))$, the desired condition would hold. Hence, it follows from (xv) that

$$\forall y \in \mathit{VWtsOf}(\mathit{modeOf}(F_0)).V''(y) = (v, \mathit{ipID}_y),$$
$$\mathit{ipID}_y \in \mathit{EPids} \text{ or } \mathit{PrioOf}(\mathit{ipID}_y) \geq \mathit{PrioOf}(\mathit{iPidOf}(F_0)) \quad (xvii)$$

Now, we must show that $\vdash \mathit{validVWt}(F'@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{pID}, x)$. Again, by construction we know that $\mathit{pID} \in \mathit{EPids}$ since it is a jit process, and hence, the result follows by Valid-VWt-E.

To re-establish the validity of all other writes w.r.t. the new high stack $F'@S_h$, we let $z' \in \mathsf{dom}(V')$ be arbitrary s.t. $V'(z') = (v_{z'}, \mathit{ipID}_{z'})$, and we invert on the rule used to derive $\vdash \mathit{validVWt}(F@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}_z, z')$. In the case where Valid-VWt-E (which is independent of the high stack), we simply reapply the same rule to obtain the desired result. Otherwise, if Valid-VWt were applied, we note that the high frame F_h that wrote x must either be in S_h alone or is F. In the former case, we reapply the same rule directly. In the latter, we observe that $\mathit{pidOf}(F) = \mathit{pidOf}(F') = \mathit{pID}$, allowing $F'@S_h$ to satisfy the same condition. By the same rule, we obtain the desired result. Since such z' was arbitrary, this result holds for all such z'.

Overall, we have just established that this condition holds $\forall z \in \mathsf{dom}(V'')$ *s.t.* $\Gamma(z) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$, $\Psi(\mathit{ID}_0, \mathit{version}_0, \mathtt{Md}_0)$, $V''(z) = (v_z, \mathit{ipID}_z)$, and $V'' = V_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow \mathsf{V}[x \mapsto (v, \mathit{pID})]$.

By applying Wf-F-V to this result and (xv), we retrieve (a3).

The reasoning for (a2) is analogous. By inverting Wf-F-N on (xiv), we learn the following where $N' = N_k \rightsquigarrow N$:

(xviii) $\forall y \in \mathit{NVWtsOf}(\mathit{modeOf}(F_0)).N'(y) = (v, \mathit{ipID}_y), \mathit{ipID}_y \in \mathit{EPids}$ or $\mathit{PrioOf}(\mathit{ipID}_y) \geq \mathit{PrioOf}(\mathit{iPidOf}(F_0))$

(xix) $\forall z \in \mathsf{dom}(N')$ s.t. $\Gamma(z) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$ and $\Psi(\mathit{ID}_0, \mathit{version}_0, \mathtt{Md}_0)$ and $N'(z) = (v_z, \mathit{ipID}_z)$, $\vdash \mathit{validNvWt}(F@S_h, \mathit{ipID}_0, \mathit{EPids}, \mathit{ipID}_z, z)$.

To re-establish similar conditions for the high stack $F'@S_h$, we again pick an arbitrary z and invert (xix) on the deriving rule. If the rule is Valid-NVWt-Init, Valid-NVWt-E, or Valid-NVWt-A, the proof proceeds by simply inverting on the rule and reapplying it to the premises to establish the validity condition for the updated high stack $F'@S_h$ (since these rules are all independent of the high stack). Otherwise, Valid-NVWt-ShCP or Valid-NVWt-HA must have been applied. In each case, we observe that the high frame F_h that last wrote to z must be in S_h alone since $\mathit{mdOf}(F) = \mathsf{jit} \neq \mathsf{discard}, \mathsf{atom}$.Therefore, $F_h \in F'@S_h$ and the same conditions apply. We can apply the same rule to establish the validity of z. Since such z was arbitrary, this result holds for all such z. By applying Wf-F-V to this result and (xviii), we establish (a2).

Applying Wf-S-ind to (a1), (a2), and (a3), we have that $\forall S_h, S_u$ s.t. $S_h@S_u = \mathit{updJitS}(S'_k, S)$:

$$N_k \rightsquigarrow N, V_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow V[x \mapsto (v, \mathit{pID})], \Psi, \mathit{EPids}, F'@S_h \vdash S_u$$

Taking $S_h = \cdot$ and $S_u = \mathit{updJitS}(S'_k, S)$, we obtain (c):

$$N_k \rightsquigarrow N, V_k[x \mapsto (v, \mathit{pID})] \rightsquigarrow V[x \mapsto (v, \mathit{pID})], \Psi, \mathit{EPids}, F' \vdash \mathit{updJitS}(S'_k, S)$$

Therefore, by applying Wf-S-ind to (a), (b), (c), we arrive at the desired result.

$$N_k \rightsquigarrow N, V_k[x \mapsto (v, pID)] \rightsquigarrow V[x \mapsto (v, pID)], \Psi, EPids, \cdot \vdash F' \rhd updJitS(S'_k, S)$$

□

Lemma 57 (Stack well-formedness under ret**)** *If*

(i) $\mathsf{N}, \mathsf{V}, \Psi, EPids, F_c \vdash S : \mathsf{wf}$

(ii) $F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, \mathsf{ret})$

(iii) $finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K'$

then $\mathsf{N}, \mathsf{V}, \Psi, EPids, \cdot \vdash S : \mathsf{wf}$.

Proof: We prove a generalized result, that if $N, V, \Psi, EPids, F_c@S_h \vdash S_l : \mathsf{wf}$ for all $S = S_h@S_l$, then $N, V, \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$ for all S_h. The proof is by induction on the size of S_l.

Base case $S_l = \cdot$**.** If $S = \cdot$, then the desired result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \rhd S_0\ (\exists F_0 = (\ldots, ID_0, \textbf{\textit{version}}_0, \mathtt{Md}_0, \ldots))$**.** If $S_l = F_0 \rhd S_0$, we can invert WF-S-IND on the assumption to learn that:

(i) $N, V, \Psi, EPids, F_c@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(ii) $N, \Psi, EPids, F_c@S_h \vdash_n F_0 : \mathsf{wf}$

(iii) $V, \Psi, EPids, F_c@S_h \vdash_v F_0 : \mathsf{wf}$

We need to show that

(a) $N, V, \Psi, EPids, S_h@F_0 \vdash S_0 : \mathsf{wf}$

(b) $N, \Psi, EPids, S_h \vdash_n F_0 : \mathsf{wf}$

(c) $V, \Psi, EPids, S_h \vdash_v F_0 : \mathsf{wf}$

(a) holds by applying the IH to (i). Towards (b), we invert WF-F-N on (ii) to learn that if $\Psi(ID_0, version_0, \mathtt{Md}_0) = \Gamma$, then $\forall x \in \mathsf{dom}(N)\ s.t.\ \Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$ and $N(x) = (v, ipID)$, $\vdash validNvWt(F_c@S_h, ipID_0, EPids, ipID, x)$. Let such x be arbitrary. The proof proceeds by cases on the deriving rule.

Case VALID-NVWT-INIT. In this case, we know that $PrioOf(ipID) \leq PrioOf(ipID_0)$ and $x \in InitOf(ipID_0) \cup shCPof(ipID_0)$. Then it follows by application of VALID-NVWT-INIT that $\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$.

Case VALID-NVWT-E, VALID-NVWT-A. These cases follow reasoning analogous to the previous case.

Case VALID-NVWT-SHCP. In this case, there exists $F_h \in F_c@S_h$ where $F_h = (ipID_h, \ldots, vPol_h, \mathsf{discard}, \ldots)$ and $vPol_h = (ShAcc, ShCP, nIds, Inits)$ and $x \in ShCP$. If $F_h \in S_h$, then we apply VALIDNVWT-SHCP to obtain the desired result. Otherwise, it must be that $F_h = F_c$. In that case, it follows by definition that F_c if effective, and hence, $ipID \in EPids$. Therefore, the desired result follows by VALID-NVWT-E:

$$\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$$

Case VALID-NVWT-HA. In this case, there exists $F_h \in F_c@S_h$ s.t.
$F_h = (ipID_h, \ldots, vPol_h, \mathtt{Md}_h, \ldots)$ and $\mathtt{Md}_h = \mathsf{atom}(\mathsf{aID}, rb_h, \omega, \mu, \rho, Inits, NVw, Vw)$ and $rb_h \geq rb$ and $x \in \omega \cup \mu \cup Inits$. Note that $F_h = F_c$ is impossible, as our semantics enforce that a frame whose command is ret cannot possibly have the mode atom (the atomic region would have already concluded, ending in the mode jit). Therefore, it must be that $F_h \in S_h$, and hence we apply VALIDNVWT-HA to obtain the desired result.

In all cases, we have established that $\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$. Since x was arbitrary, this result holds for all such x. By application of WF-F-N, we establish (b).

By application of WF-S-IND to (a), (b), and (c), we arrive at the desired result:

$$N, V, \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$$

Overall, we have shown that $N, V, \Psi, EPids, S_h \vdash S : \mathsf{wf}$ holds for all S_h. Taking $S_h = \cdot$, we obtain the desired result:

$$N, V, \Psi, EPids, \cdot \vdash S : \mathsf{wf}$$

□

Lemma 58 (*updJitS* well-formedness under ret) *If*

(i) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, F_c \rhd S) : \mathsf{wf}$
(ii) $F_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \cdot, \mathtt{Md}_c, \mathsf{ret})$
(iii) $finalizeK(K, F_c, \mathsf{V}, \mathsf{N}) = K'$

then $(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash S' : \mathsf{wf}$ *where*

(a) $S' = updJitS(S'_k, S)$ *if* $\mathtt{Md}_c = \mathsf{jit}$ *and* $S_k = F_j(pID_c) \rhd S'_k$
(b) $S' = updJitS(S_k, S)$ *if* $\mathtt{Md}_c = \mathsf{discard}$
(c) $S' = updJitS(S_k, S)$ *if* $\mathtt{Md}_c = \mathsf{next}$

Proof: The proof is by cases on $\mathtt{Md}_c$.

Case $\mathtt{Md}_c = \mathsf{jit}$**.** We need to show that:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S) : \mathsf{wf}$$

By definition of *updJitS*, we know that $updJitS(S_k, F_c \rhd S) = F_c \rhd updJitS(S'_k, S)$. Therefore, we know by assumption (i) that:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash F_c \rhd updJitS(S'_k, S) : \mathsf{wf}$$

By inversion of WF-S-IND on (i), we know that

(iv) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, F_c \vdash updJitS(S'_k, S) : \mathsf{wf}$
(v) $(N_k \rightsquigarrow N), \Psi, EPids, \cdot \vdash_n F_c : \mathsf{wf}$
(vi) $(V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash_v F_c : \mathsf{wf}$

We must establish similar conditions wrt the desired result.

First, we prove a general result: given $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, F_c@S_h \vdash S_l : \mathsf{wf}$ for all stacks S_h, S_l s.t. $updJitS(S'_k, S) = S_h@S_l$, we must show that

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$$

The proof is by induction on the size of S_l.

Base case $S_l = \cdot$**.** The result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \rhd S_0$ $(\exists F_0 = (\boldsymbol{ipID}_0, \ldots))$**.** By inversion of WF-S-IND, we have

(vii) $(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, F_c@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(viii) $(N_k \rightsquigarrow N), \Psi, EPids, F_c@S_h \vdash_n F_0 : \mathsf{wf}$

(ix) $(V_k \rightsquigarrow V), \Psi, EPids, F_c@S_h \vdash_v F_0 : \mathsf{wf}$

By application of the IH to (vii), we have that

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, S_h@F_0 \vdash S_0 : \mathsf{wf}\ (x)$$

Now, we must establish that $(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), \Psi, EPids, S_h \vdash_n F_0 : \mathsf{wf}$. In particular, we must show that $\forall x \in \mathsf{dom}(N).N(x) = (v, ipID)$ and $\Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$, we have

$$\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$$

given that

$$\vdash validNvWt(F_c@S_h, ipID_0, EPids, ipID, x)\ \ (xi)$$

Let such x be arbitrary. The proof is by cases on the deriving rule for (xi). Since $mdOf(F_0) \neq \mathsf{atom}$, VALID-NVWT-HA does not apply. The only interesting case is VALID-NVWT-SHCP, as the others do not rely on $F_c@S_h$. In that case, it must be that either $mdOf(F_h) = \mathsf{discard}$ $\exists F_h \in S_h$ and $x \in ShCP$ or $mdOf(F_c) = \mathsf{discard}$. In the first case, the desired result follows again by application of VALID-NVWT-SHCP. Otherwise, if $mdOf(F_c) = \mathsf{discard}$ and that $ipID = ipID_c$, it follows by definition of F_c and construction of $EPids$ that $ipID_c \in EPids$. Therefore, VALID-NVWT-E applies and we obtain the desired result.

In all cases, we have that $\vdash validNvWt(S_h, ipID_0, EPids, ipID, x)$. Since such x was arbitrary, the result holds $\forall x \in \mathsf{dom}(N).N(x) = (v, ipID)$ and $\Gamma(x) = (q_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{q_c} = \mathsf{Id}$.

By application of WF-F-N to this result, we have that

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), \Psi, EPids, S_h \vdash_n F_0 : \mathsf{wf}\ \ (xii)$$

The reasoning to obtain $(V_k \rightsquigarrow V), \Psi, EPids, S_h \vdash_v F_0 : \mathsf{wf}$ $(xiii)$ is similar.

By application of WF-S-IND to (x), (xii), and (xiii), we obtain the desired result:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, S_h \vdash S_l : \mathsf{wf}$$

Taking $S_h = \cdot$ and $S_l = updJitS(S'_k, S)$, we obtain the desired result:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S'_k, S) : \mathsf{wf}$$

Case $\mathtt{Md}_c \in \{\mathsf{discard}, \mathsf{next}\}$**.** We need to show that:

$$(\mathsf{N}_k{\downarrow}_{ckVarof(S_k)} \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$$

By definition of *updJitS* and since $\mathtt{Md}_c \neq \mathsf{jit}$, we know that $updJitS(S_k, F_c \triangleright S) = updJitS(S_k, S)$. Therefore, we know by assumption (i) that:

$$(N_k \rightsquigarrow N), (V_k \rightsquigarrow V), \Psi, EPids, \cdot \vdash updJitS(S_k, S) : \mathsf{wf}$$

The remainder of the proof is by induction on the size of $updJitS(S_k, S)$. We re-establish the well-formedness of each frame on the stack w.r.t. $\vdash_n$ and $\vdash_v$.

□

Lemma 59 (Stack well-formedness under end atomic to jit) *If* $N, V, \Psi, EPids, F \vdash S : \mathsf{wf}$ *and* $F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{atom}(\dots), \mathsf{end}_{\mathsf{atom}})$*, then* $N, V, \Psi, EPids, F' \vdash S : \mathsf{wf}$ *where* $F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip})$.

Proof: We prove a generalized result, that if $N, V, \Psi, EPids, F@S_h \vdash S_c : \mathsf{wf}$, then $N, V, \Psi, EPids, F'@S_h \vdash S_c : \mathsf{wf}$ for all S_h, S_c s.t. $S = S_h@S_l$. The proof is by induction on the size of S_l.

Base case $S_l = \cdot$**.** If $S_l = \cdot$, then the desired result holds by WF-S-EMPTY.

Inductive case $S_l = F_0 \triangleright S_0\, (\exists F_0 = (\dots, ID_0, \textbf{\textit{version}}_0, \mathtt{Md}_0, \dots))$**.** If $S_c = F_0 \triangleright S_0$, we can invert WF-S-IND on the assupmtion to learn that:

(i) $N, V, \Psi, EPids, F@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(ii) $N, \Psi, EPids, F@S_h \vdash_n F_0 : \mathsf{wf}$

(iii) $V, \Psi, EPids, F@S_h \vdash_v F_0 : \mathsf{wf}$

We need to show that

(a) $N, V, \Psi, EPids, F'@S_h@F_0 \vdash S_0 : \mathsf{wf}$

(b) $N, \Psi, EPids, F'@S_h \vdash_n F_0 : \mathsf{wf}$

(c) $V, \Psi, EPids, F'@S_h \vdash_v F_0 : \mathsf{wf}$

(a) holds by applying the IH to (i). Towards (b), we invert WF-F-N on (ii) to learn that if $\Psi(ID_0, version_0, \mathtt{Md}_0) = \Gamma$, then $\forall x \in \mathsf{dom}(N)\, s.t.\, \Gamma(x) = (qID_c, \mathsf{Id}, \mathsf{Id})$ where $\overline{qID_c} = \mathsf{Id}$ and $N(x) = (v, ipID)$, $\vdash validNvWt(F@S_h, ipID_0, EPids, ipID, x)$. Let such x be arbitrary. The proof proceeds by cases on the deriving rule.

Case VALID-NVWT-INIT. In this case, we know that $PrioOf(ipID) \leq PrioOf(ipID_0)$ and $x \in InitOf(ipID_0) \cup shCPof(ipID_0)$. Then it follows by application of VALID-NVWT-INIT that $\vdash validNvWt(F'@S_h, ipID_0, EPids, ipID, x)$.

Case VALID-NVWT-E, VALID-NVWT-A. These cases follow reasoning analogous to the previous case.

Case VALID-NVWT-SHCP. In this case, there exists $F_h \in F@S_h$ s.t. $F_h = (ipID_h, \ldots, vPol_h, \mathsf{discard}, \ldots)$ where $vPol_h = (ShAcc, ShCP, nIds, Inits)$ and $x \in ShCP$. Since $mdOf(F) \neq \mathsf{discard}$, it must be that $F_h \in S_h$, and trivially, $F_h \in F'@S_h$. Therefore, it follows by applicaiton of VALID-NVWT-SHCP that $\vdash validNvWt(F'@S_h, ipID_0, EPids, ipID, x)$.

Case VALID-NVWT-HA. In this case, there exists $F_h \in F@S_h$ s.t. $F_h = (ipID_h, \ldots, vPol_h, \mathtt{Md}_h, \ldots)$ and $\mathtt{Md}_h = \mathsf{atom}(\mathsf{aID}, rb_h, \omega, \mu, \rho, Inits, NVw, Vw)$ and $rb_h \geq rb$ and $x \in \omega \cup \mu \cup Inits$. If $F_h \in S_h$, then we apply VALIDNVWT-HA to obtain the desired result. Otherwise, it must be the case that $F_h = F$. In other words, x was written by the process corresponding to F. However, since we know that F is a successfully completed atomic region, by definition of F, we know that its process is effective, and hence $ipID_h = iPidOf(F) \in EPids$. Therefore, $\vdash validNvWt(F'@S_h, ipID_0, EPids, ipID, x)$ follows by application of VALID-NVWT-E.

In all cases, we have established that $\vdash validNvWt(F'@S_h, ipID_0, EPids, ipID, x)$. Since x was arbitrary, this result holds for all such x. Since $mdOf(F') = \mathsf{jit}$ and $NVWtsOf(\mathsf{jit}) = \cdot$, (b) follows by application of WF-F-N. (c) is established by similar reasoning, using the analogous rules for VALID-VWT.

By application of WF-S-IND to (a), (b), and (c), we arrive at the desired result:

$$N, V, \Psi, EPids, F'@S_h \vdash S_l : \mathsf{wf}$$

Overall, we have shown that $N, V, \Psi, EPids, F'@S_h \vdash S_l : \mathsf{wf}$ holds for all S_h. Taking $S_h = \cdot$ and $S_l = S$, we obtain the desired result:

$$N, V, \Psi, EPids, F' \vdash S : \mathsf{wf}$$

□

Lemma 60 (NVwts to Commit are ID) *If* $\Psi, EPids \vdash \Sigma : \mathsf{wf}$, $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$, $\Sigma \Longrightarrow \Sigma'$ *via* ENDATOM-I, RET-I, RET-IQ, *then* $\forall x$ *s.t.* $N_k(x) \neq N'_k(x)$, $N(x) = (v, ipID)$, $ipID \in EPids$.

Proof: By the construction of Ψ, all the variables committed to N'_k have type $(\mathsf{ld}, \mathsf{ld}, \mathsf{ld})$ for some process $ipID_{cp}$ where there exists some frame $F_{cp} \in S$ s.t. $ipidOf(F_{cp}) = ipID_{cp}$. Towards contradiction, suppose that that process that wrote x is ineffective, i.e., $ipID \notin EPids$. Then it follows by the well-formedness definition $\vdash_n$ that since x is a valid write, $ipID$ must be a higher priority process than $ipID_{cp}$ on S and that $ipID$ must be higher priority than F. However, every frame in S has lower priority than F by construction. Hence, we have a contradiction. Therefore, it must be the case that $ipID \in EPids$. □

Lemma 61 (Vwts to Commit are ID) *If* $\Psi, EPids \vdash \Sigma : \mathsf{wf}$, $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$, $\Sigma \Longrightarrow \Sigma'$ *via* EndAtom-I, Ret-I, Ret-IQ, *then* $\forall x$ *s.t.* $V_k(x) \neq V'_k(x)$, $V(x) = (v, ipID)$, $ipID \in EPids$.

Proof: Towards contradiction, suppose that $ipID \notin EPids$. Then, it follows by the well-formedness definition $\vdash_v$ that since x is a valid write, then exists $F_h \in F@S$ such that $prioOf(F_h) > prioOf(F)$ and $ipidOf(F_h) = ipID$. However, F is the top frame, so by construction there is no frame with higher priority. Thus, we have a contradiction. Therefore, $ipID \in EPids$. □

Lemma 62 (ID expressions written by effective processes) *If*

- $\Psi, EPids \vdash \Sigma : \mathsf{wf}$,
- $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$,
- $F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c)$,
- $\Psi, PDecl, EPids \vdash N_I \approx N_c$,
- $EPids \vdash V_I \approx V_c$,
- $\mathtt{sMd} = smodeOf(\mathtt{Md})$,
- $\Psi(pID, version, \mathtt{sMd}) = \Gamma$,
- $\Gamma \vdash e : t@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle$,

then $[\![e]\!]_{N_I, V_I, \upsilon} = [\![e]\!]_{N_c, V_c, \upsilon}$.

Proof: By induction over the structure of e.

Base case $e = x$**.** Let $x \mapsto (v, ipID) \in N_I, V_I$ for some v and $ipID$. Since, by assumption, x is typed as fully idempotent, it must be the case that $ipID \in EPids$.

We case on whether $x \in \mathsf{dom}(V_I)$ or $x \in \mathsf{dom}(N_I)$.

Case $x \in \mathsf{dom}(V_I)$**.** If $x \in \mathsf{dom}(V_I)$, then it follows by inversion of Equiv-V on the assumption $EPids \vdash V_I \approx V_c$ that

- $EPids \vdash V_I \Rightarrow X_{ineff}$
- $V_I \setminus X_{ineff} = V_c \setminus X_{ineff}$

Since $ipID \in EPids$, then V-Eff applies. By inversion, we determine that x was not collected into the ineffective writes set X_{ineff}. Hence, it follows by $V_I \setminus X_{ineff} = V_c \setminus X_{ineff}$ that $V_I(x) = V_c(x)$.

Case $x \in \mathsf{dom}(N_I)$**.** If $x \in \mathsf{dom}(N_I)$, then it follows by inversion of Equiv-N on the assumption $\Psi, PDecl, EPids \vdash N_I \approx N_c$ that

- $\Psi, PDecl, EPids \vdash N_I \Rightarrow X_{nid}, X_{cp}$
- $N_I \setminus X_{nid} = N_c \setminus X_{nid}$

Since $ipID \in EPids$ and x's local type is Id, the rule N-Wt-Eff-Id applies. By inversion, we learn that $x \notin X_{nid}$. Therefore, it follows by $N_I \setminus X_{nid} = N_c \setminus X_{nid}$ that $N_I(x) = N_c(x)$.

Inductive case $e = e_1 \odot e_2$**.** There are two cases to consider: $\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}$ or $\mathtt{sMd} = \mathsf{atom}$.

Case $\mathtt{sMd} \in \{\mathsf{jit}, \mathsf{discard}\}$**.** By inversion of T-Binop-Read-DJ on the assumed typing judgment, we have

- $\Gamma \vdash^{\mathtt{sMd}} e_1 : t@q_1, \Lambda_1$
- $\Gamma \vdash^{\mathtt{sMd}} e_2 : t@q_2, \Lambda_2$

By application of the IH to both of these judgments, we obtain that $v_I^1 = v_c^1$ and $v_I^2 = v_c^2$ where

- $N_I, V_I, \upsilon_I \vdash e_1 \Downarrow v_I^1$,
- $N_I, V_I, \upsilon_I \vdash e_2 \Downarrow v_I^2$,
- $N_c, V_c, \upsilon_c \vdash e_1 \Downarrow v_c^1$, and
- $N_c, V_c, \upsilon_c \vdash e_2 \Downarrow v_c^2$.

Therefore, the desired result follows by definition of $\Downarrow$.

Case $\mathtt{sMd} = \mathsf{atom}$**.** The proof in this case is similar to above, instead relying on the rule T-Binop-Read-A.

□

Lemma 63 (High step projection the same) *If*

- $N, \Psi, EPids, \cdot \vdash_n F : \mathsf{wf}$
- $V, \Psi, EPids, \cdot \vdash_v F : \mathsf{wf}$
- $ePidOf(T) = EPids$,
- $T_I|_j \vdash \Sigma \approx \sigma$.
- $\Psi, EPids \vdash \Sigma : \mathsf{wf}$,
- $\Sigma \overset{o}{\Longrightarrow} \Sigma'$,
- $\Sigma = I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q$,
- $\Sigma' = I, IRQ, K', \mathsf{N}', \mathsf{V}', S, F', Q$,
- $F = (\cdots, \mathsf{Nid} \rhd pcS, \cdots, c)$ *and*
- $F' = (\cdots, \mathsf{Nid} \rhd pcS, \cdots, c')$

then $T_I|_{j+1} \vdash \Sigma' \approx \sigma$ *s.t.* $\Psi, EPids \vdash \Sigma' : \mathsf{wf}$.

Proof: We first apply Lemma 46 to establish $\Psi, EPids \vdash \Sigma' : \mathsf{wf}$. By cases on $\Sigma \overset{a}{\Longrightarrow} \Sigma'$. We show the most interesting cases: writing to volatile and nonvolatile memory.

Case V-Assign-I. If $c = x := e$ where $x \in \mathsf{dom}(V)$, then V-Assign-I applies.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, \mathsf{Nid} \rhd pcS, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, \mathsf{Nid} \rhd pcS, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{V}) \\ \mathtt{Md}'_c = \mathtt{Md}_c[Vw \leftarrow Vw \cup \{x\}] \qquad K = (N_k, V_k, S_k) \qquad K' = (N_k, V'_k, S_k) \\ V'_k = \begin{cases} V_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ V_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}[x \mapsto (v, ipID)], S, F', Q} \text{ V-Assign-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$ by its definition. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N, V[x \mapsto (v, ipID)], S, F', Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N \approx N_c$

(c) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V[x \mapsto (v, ipID)] \approx V_c$

(e) $\forall x', V'_k(x') \neq V_c(x'), V[x \mapsto (v, ipID)](x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES* (which computes the effective stack and ids completely independent of β), we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). Since $S'_e = S_e$, the result holds directly by (ii).

Ad (b). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (iii).

Ad (c). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (iv) (no nonvolatile locations are written).

Ad (d). By inversion of Equiv-V on (v), we know that $V \setminus X_{ineff} = V_c \setminus X_{ineff}$. Since by assumption, the top *pc* is Nid, x types as non-idempotent, and it follows by V-NEff that x is collected into X_{ineff} (call it $X'_{ineff} = X_{ineff} \cup \{x\}$). Hence, it follows by Equiv-V that $V \setminus X'_{ineff} = V_c \setminus X'_{ineff}$, as desired.

Ad (e). For every $x' \neq x$, the desired result follows directly by (vi). Towards contradiction, suppose that $x' = x$ s.t. $V'_k(x) \neq V_c(x)$. Since the top *pc* is Nid, it follows that x is non-idempotent. However, by definition a non-idempotent variable cannot have a location in the checkpointed context. Therefore, $V'_k(x) = V_c(x)$, and the desired result follows directly by (vi).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have estbalished that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$.

Case N-Assign-I. If $c = x := e$ where $c \in \mathsf{dom}(N)$, then N-Assign-I applies.

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_0) \in PDecl \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{N}) \qquad K = (N_k, V_k, S_k) \qquad \Psi(ID_c, version_c, \mathtt{Md}_c) = \Gamma \\ \mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \Gamma(x).1 = \mathsf{Id}(\_) \\ \mathtt{Md}_c & \text{otherwise} \end{cases} \qquad K' = (N'_k, V_k, S_k) \\ N'_k = \begin{cases} N_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ N_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], \mathsf{V}, S, F', Q} \text{ N-Assign-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$ by its definition. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N[x \mapsto (v, ipID)], V, S, F', Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N[x \mapsto (v, ipID)] \approx N_c$

(c) $\forall x', N'_k(x') \neq N_c(x'), N[x \mapsto (v, ipID)](x') = N_c(x')$ and $N[x \mapsto (v, ipID)](x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V \approx V_c$

(e) $\forall x', V_k(x') \neq V_c(x'), V(x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES* (which computes the effective stack and ids completely independent of β), we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). Since $S'_e = S_e$, the result holds directly by (ii).

Ad (b). By inversion of Equiv-N on (iii), we know that $N \setminus X_{nid} = N_c \setminus X_{nid}$. Since by assumption the top *pc* is Nid, x must type as Nid, and hence it follows by N-Wt-Nid and N-Wt-Nid-Cp that x is collected into X_{nid} (call it $X'_{nid} = X_{nid} \cup \{x\}$). Hence, it follows by Equiv-N that $N \setminus X'_{nid} = N_c \setminus X'_{nid}$, as desired.

Ad (c). For every $x' \neq x$, the desired result follows directly by (iv). Now let $x' = x$ and suppose towards contradiction that $N'_k(x) \neq N_c(x)$. However, we know by assumption that since the top *pc* is Nid, the value written to x is non-idempotent, meaning it cannot have a location in $N'_k(x)$ (a contradiction). Therefore, $N'_k(x) = N_c(x)$, and the desired result holds by (iv).

Ad (d). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (v).

Ad (e). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (vi) (no volatile locations are written).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have estbalished that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$.

□

Lemma 64 (High step write only NID) *If*

- $\Sigma = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q)$,
- $\sigma = (I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q)$,
- $\sigma' = (I, IRQ, \mathsf{N}', \mathsf{V}', S, F', Q)$,
- $F = (\cdots, \mathsf{Nid} \rhd pcS, \cdots, \mathsf{end_{if}} :: E, \cdots, \mathsf{skip})$ *and*
- $F' = (\cdots, pcS, \cdots, E, \cdots, \mathsf{skip})$
- $\sigma \longrightarrow \sigma'$,
- $T_I \vdash \Sigma \approx T_C(\sigma)$

then $T_I \vdash \Sigma \approx T_C(\sigma')$.

Proof: By assumption, the step $\sigma \longrightarrow \sigma'$ must be done by IF-END-I:

$$\frac{F = (pID, rID, ID, version, \overrightarrow{pr}, \mathsf{Nid} \rhd pcS, vPol, \upsilon, \mathsf{end_{if}} :: E, \mathtt{Md}, \mathsf{skip}) \quad F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{skip})}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \longrightarrow^* I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q} \text{ IF-END-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I_I, IRQ_I, K_I, N_I, V_I, S_I, F_I, Q_I)$
- $\sigma = (I, IRQ, \mathsf{N}, \mathsf{V}, S, F, Q)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ_I, T_I)$
- $EEvIds = eEvOf(Q_I, T_I)$
- $EInpIds = eInpOf(I_I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F_I \rhd S_I) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F \rhd S$

(iii) $\Psi, PDecl, EPids, \mathsf{Id} \vdash N_I \approx N$

(iv) $\forall x, N_k(x) \neq N(x), N_I(x) = N(x)$ and $N_I(x) = (v, ipID)$ $s.t.$ $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V_I \approx V$

(vi) $\forall x, V_k(x) \neq V(x), V_I(x) = V(x)$ and $V_I(x) = (v, ipID)$ $s.t.$ $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I_I{\downarrow_{EInpIds}} = I{\downarrow_{EIrqIds}}$

(viii) $IRQ_I{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ{\downarrow_{EIrqIds}}$

(ix) $Q_I{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_j \vdash \Sigma_j \approx \sigma'$ by its definition. That is, letting

- $\Sigma_j = T_I{\downarrow_j} = (I_I, IRQ_I, K_I, N_I, V_I, S_I, F_I, Q_I)$
- $\sigma' = (I, IRQ, \mathsf{N}, \mathsf{V}, S, F', Q)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ_I, T_I)$
- $EEvIds = eEvOf(Q_I, T_I)$
- $EInpIds = eInpOf(I_I, T_I)$

we need to establish the following conditions hold, where $EPids, EIrqIds, EEvIds \vdash getES(S_k, F_I \rhd S_I) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F' \rhd S$

(b) $\Psi, PDecl, EPids, topOfPCS(pcS) \vdash N_I \approx N$

(c) $\forall x, N_k(x) \neq N(x), N_I(x) = N(x)$ and $N(x) = (v, ipID)$ $s.t.$ $ipID \in EPids$ and $x \in eNVWtsOf(S'_e)$

(d) $EPids \vdash V_I \approx V$

(e) $\forall x, V_k(x) \neq V(x), V_I(x) = V(x)$ and $V_I(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S'_e)$

(f) $I_I{\downarrow_{EInpIds'}} = I{\downarrow_{EIrqIds}}$

(g) $IRQ_I{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ{\downarrow_{EIrqIds}}$

(h) $Q_I{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs'_d}} = Q{\downarrow_{EEvIds}}$

By definition of *getES*, we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). Follows by (ii) and by definition (which includes $\mathsf{end_{if}}$).

Ad (b). If $topOfPCS(pcS) = \mathsf{Id}$, then the result holds by (iii). If $topOfPCS(pcS) = \mathsf{Nid}$, then the result follows by a combination of (iii) and N-Wt-Eff-Nid.

Ad (c). Let $x \in dom(N' \setminus N)$ be arbitrary, i.e., x is written from N to N', such that $N_k(x) \neq N'(x)$. By typing information, we know that since x was written in an Nid branch, it has type Nid, and hence cannot have a location in N_k. Therefore, we have a contradiction and there exists no such x. Hence, $N_k(x) = N'(x)$. The desired result follows by (iv).

Ad (d). Follows by (v).

Ad (e). Follows by (vi).

Ad (f). Follows by (vii).

Ad (g). Follows by (viii).

Ad (h). Follows by (ix).

Therefore, the desired result holds.

□

Lemma 65 (In-Effective one step) *If* $ePidOf(T) = EPids$, $\Psi, EPids \vdash \Sigma_j : \mathsf{wf}$, $T_I|_j \vdash \Sigma_j \approx \sigma$, $\Sigma_j = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, c)$, $iPidOf(F) \notin EPids$, *and* $\Sigma_j \Longrightarrow \Sigma_{j+1}$, *then* $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$ *and* $\Psi, EPids \vdash \Sigma_{j+1} : \mathsf{wf}$.

Proof: We first apply Lemma 46 to establish $\Psi, EPids \vdash \Sigma_{j+1} : \mathsf{wf}$. The proof is by cases on the step $\Sigma_j \Longrightarrow \Sigma_{j+1}$. Reboots are not considered here, as they are covered in the top-level lemma. The only interesting cases are assignment to nonvolatile memory and assignment to volatile memory.

Case V-Assign-I. Suppose the last step in the trace is V-Assign-I:

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{V}) \\ \mathtt{Md}'_c = \mathtt{Md}_c[Vw \leftarrow Vw \cup \{x\}] \qquad K = (N_k, V_k, S_k) \qquad K' = (N_k, V'_k, S_k) \\ V'_k = \begin{cases} V_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ V_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}[x \mapsto (v, ipID)], S, F', Q} \text{ V-Assign-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$ by its definition. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N, V[x \mapsto (v, ipID)], S, F', Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N \approx N_c$

(c) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V[x \mapsto (v, ipID)] \approx V_c$

(e) $\forall x', V'_k(x') \neq V_c(x'), V[x \mapsto (v, ipID)](x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES* (which computes the effective stack and ids completely independent of β), we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). Since $S'_e = S_e$, the result holds directly by (ii).

Ad (b). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (iii).

Ad (c). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (iv) (no nonvolatile locations are written).

Ad (d). By inversion of Equiv-V on (v), we know that $V \setminus X_{ineff} = V_c \setminus X_{ineff}$. Since by assumption, $ipID \notin EPids'$, it follows by V-NEff that x is collected into X_{ineff} (call it $X'_{ineff} = X_{ineff} \cup \{x\}$). Hence, it follows by Equiv-V that $V \setminus X'_{ineff} = V_c \setminus X'_{ineff}$, as desired.

Ad (e). For every $x' \neq x$, the desired result follows directly by (vi). Now suppose $x' = x$. Since by assumption F is an ineffective process, we know that $V'_k = V_k$, and hence $V'_k(x) = V_k(x)$. The desired result follows again by (vi).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have estbalished that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$.

Case N-Assign-I. Suppose the last step in the trace is N-Assign-I:

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_0) \in PDecl \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{N}) \qquad K = (N_k, V_k, S_k) \qquad \Psi(ID_c, version_c, \mathtt{Md}_c) = \Gamma \\ \mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \Gamma(x).1 = \mathsf{Id}(\_) \\ \mathtt{Md}_c & \text{otherwise} \end{cases} \qquad K' = (N'_k, V_k, S_k) \\ N'_k = \begin{cases} N_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ N_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], \mathsf{V}, S, F', Q} \text{ N-Assign-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$

- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$ by its definition. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N[x \mapsto (v, ipID)], V, S, F', Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N[x \mapsto (v, ipID)] \approx N_c$

(c) $\forall x', N'_k(x') \neq N_c(x'), N[x \mapsto (v, ipID)](x') = N_c(x')$ and $N[x \mapsto (v, ipID)](x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V \approx V_c$

(e) $\forall x', V_k(x') \neq V_c(x'), V(x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES* (which computes the effective stack and ids completely independent of β), we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). Since $S'_e = S_e$, the result holds directly by (ii).

Ad (b). By inversion of Equiv-N on (iii), we know that $N \setminus X_{nid} = N_c \setminus X_{nid}$. Since by assumption, $ipID \notin EPids'$, it follows by N-Wt-Nid and N-Wt-Nid-Cp that x is collected into X_{nid} (call it $X'_{nid} = X_{nid} \cup \{x\}$). Hence, it follows by Equiv-N that $N \setminus X'_{nid} = N_c \setminus X'_{nid}$, as desired.

Ad (c). For every $x' \neq x$, the desired result follows directly by (iv). Now suppose $x' = x$. Since by assumption F is an ineffective process, we know that $N'_k = N_k$, and hence $N'_k(x) = N_k(x)$. The desired result follows again by (iv).

Ad (d). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (v).

Ad (e). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (vi) (no volatile locations are written).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have estbalished that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$.

□

Lemma 66 (Effective (not Nid branch) one step) *If* $ePidOf(T) = EPids$, $\Psi, EPids \vdash \Sigma_j : \mathsf{wf}$, $T_I|_j \vdash \Sigma_j \approx \sigma$, $\Sigma_j = (I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, c)$, $iPidOf(F) \in EPids$, *and* $\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1}$, *and* $topOfPCS(F) \neq \mathsf{Nid}$, *then* $\sigma \xrightarrow{a} \sigma'$ *and* $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$ *and* $\Psi, EPids \vdash \Sigma_{j+1} : \mathsf{wf}$.

Proof: We first apply Lemma 46 to establish $\Psi, EPids \vdash \Sigma_{j+1} : \mathsf{wf}$. The proof is by cases on the step $\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1}$. The main interesting cases are assignment to nonvolatile memory, assignment to volatile memory, branching on idempotent expressions, and starting and ending an atomic region.

Case V-Assign-I. Suppose the last step in the trace is V-Assign-I:

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{V}) \\ \mathtt{Md}'_c = \mathtt{Md}_c[Vw \leftarrow Vw \cup \{x\}] \qquad K = (N_k, V_k, S_k) \qquad K' = (N_k, V'_k, S_k) \\ V'_k = \begin{cases} V_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ V_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}[x \mapsto (v, ipID)], S, F', Q} \text{ V-Assign-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$ by its definition, where $\sigma \xrightarrow{a} \sigma'$ while taking V-Assign-C in the continuous semantics. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N, V[x \mapsto (v, ipID)], S, F', Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c[x \mapsto (v, ipID)], S_c, F'_c, Q_c)$
- $F'_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip})$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$

- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F'_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N \approx N_c$

(c) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V[x \mapsto (v, ipID)] \approx V_c[x \mapsto (v, ipID)]$

(e) $\forall x', V'_k(x') \neq V_c[x \mapsto (v, ipID)](x'), V[x \mapsto (v, ipID)](x') = V_c[x \mapsto (v, ipID)](x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES* (which computes the effective stack and ids completely independent of β), we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). Since $S'_e = S_e$ and F'_c only differs from F_c in its command (i.e., $F_c|_{id} = F'_c|_{id}$), the result holds directly by (ii).

Ad (b). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (iii).

Ad (c). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (iv) (no nonvolatile locations are written).

Ad (d). By inversion of Equiv-V on (v), we know that $V \setminus X_{ineff} = V_c \setminus X_{ineff}$. Hence, $V[x \mapsto (v, ipID)] \setminus X'_{ineff} = V_c[x \mapsto (v, ipID)] \setminus X'_{ineff}$ where $X'_{ineff} = X_{ineff}$ since x is written to on both exection traces by the same effective process. Therefore, by Equiv-V we conclude $EPids' \vdash V[x \mapsto (v, ipID)] \approx V_c[x \mapsto (v, ipID)]$

Ad (e). For every $x' \neq x$, the desired result follows directly by (vi). Now suppose $x' = x$. If the mode is jit, then the result follows immediately (i.e., x does not satisfy the condition since it is written through). Otherwise, the write to x by the continuous execution trace differs from its value in the checkpointed context. By definition we know that $V[x \mapsto (v, ipID)](x) = V_c[x \mapsto (v, ipID)](x)$, and by assumption we know that $ipID \in EPids'$.

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have established that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$.

Case N-Assign-I. Suppose the last step in the trace is N-Assign-I:

$$\frac{\begin{array}{c} F = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, x := e) \\ \mathsf{task}(ID_c, version_c, pr_c, vPol_c, rPol_c, \lambda x.c_0) \in PDecl \\ F' = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}'_c, \mathsf{skip}) \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = v \qquad x \in \mathsf{dom}(\mathsf{N}) \qquad K = (N_k, V_k, S_k) \qquad \Psi(ID_c, version_c, \mathtt{Md}_c) = \Gamma \\ \mathtt{Md}'_c = \begin{cases} \mathtt{Md}_c[NVw \leftarrow NVw \cup \{x\}] & \text{if } \Gamma(x).1 = \mathsf{Id}(\_) \\ \mathtt{Md}_c & \text{otherwise} \end{cases} \qquad K' = (N'_k, V_k, S_k) \\ N'_k = \begin{cases} N_k[x \mapsto (v, ipID)] & \text{if } \mathtt{Md}_c = \mathsf{jit} \\ N_k & \text{else} \end{cases} \qquad ipID = iPidOf(pID_c, \mathtt{Md}_c) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}[x \mapsto (v, ipID)], \mathsf{V}, S, F', Q} \text{ N-Assign-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$ by its definition, where $\sigma \xrightarrow{a} \sigma'$ while taking N-Assign-C in the continuous semantics. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N[x \mapsto (v, ipID)], V, S, F', Q)$
- $\sigma' = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c)$

- $F'_c = (pID_c, rID_c, ID_c, version_c, \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{skip})$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F'_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N[x \mapsto (v, ipID)] \approx N_c[x \mapsto (v, ipID)]$

(c) $\forall x', N'_k(x') \neq N_c(x'), N[x \mapsto (v, ipID)](x') = N_c[x \mapsto (v, ipID)](x')$ and $N[x \mapsto (v, ipID)](x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V \approx V_c$

(e) $\forall x', V_k(x') \neq V_c(x'), V(x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EInpIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES* (which computes the effective stack and ids completely independent of β), we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). By definition of F'_c we know that $F'_c|_{id} = F_c|_{id}$. Since $S'_e = S_e$, the result holds directly by (ii).

Ad (b). By inversion of Equiv-N on (iii), we know that $N \setminus X_{nid} = N_c \setminus X_{nid}$. Since $ipID \in EPids'$, we proceed in cases on the local type of x. If it is Nid, then N-Wt-Eff-Nid applies, yielding that $N[x \mapsto (v, ipID)] \setminus (X_{nid} \cup \{x\}) = N_c[x \mapsto (v, ipID)] \setminus (X_{nid} \cup \{x\})$. In the case where x's local type is Id, N-Wt-Eff-Id applies and we have that $N[x \mapsto (v, ipID)] \setminus X_{nid} = N_c[x \mapsto (v, ipID)] \setminus X_{nid}$. In both cases, Equiv-N then applies to establish the desired result.

Ad (c). For every $x' \neq x$, the desired result follows directly by (vi). Now suppose $x' = x$. If the mode is jit, then the result follows immediately (i.e., x does not satisfy the condition since it is written through). Otherwise, the write to x by the continuous execution trace differs from its value in the checkpointed context. By definition we know that $N[x \mapsto (v, ipID)](x) = N_c[x \mapsto (v, ipID)](x)$, and by assumption we know that $ipID \in EPids'$.

Ad (d). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (v).

Ad (e). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (vi) (no volatile locations are written).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have established that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$.

Case IF-TRUE-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ pcS = pc_0 \rhd pcS_0 \\ [\![e]\!]_{\mathsf{N},\mathsf{V},\upsilon_c} = \mathsf{true} \qquad \mathsf{task}(ID, version, pr, pcS, vPol, rPol, \lambda x.c) \in PDecl \\ \Psi(ID, version, \mathtt{sMd}) = \Gamma \qquad \mathtt{sMd} = smodeOf(\mathtt{Md}) \qquad pc = (\sqcup_{x \in \Lambda} \Gamma(x)\downarrow_{\overline{q}}) \sqcup pc_0 \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pc \rhd pcS, vPol, \upsilon, \mathsf{end}_{\mathsf{if}} :: E, \mathtt{Md}, c_1) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F', Q}\ \text{IF-TRUE-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I\downarrow_j = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I\downarrow_{EInpIds} = I_c\downarrow_{EIrqIds}$

(viii) $IRQ\downarrow_{EIrqIds} \cup T_I\downarrow_{inIDs_d} = IRQ_c\downarrow_{EIrqIds}$

(ix) $Q\downarrow_{EEvIds} \cup T_I\downarrow_{reIDs_d} = Q_c\downarrow_{EEvIds}$

Since $pc = topOfPCS(F) \neq \mathsf{Nid}$, we know that $pc = \mathsf{Id}$, and hence it follows by definition that $\Gamma \vdash e : t@\langle \mathsf{Id}, \mathsf{Id}, \mathsf{Id} \rangle\ (x)$, where $\Gamma = \Psi(pID, version, \mathtt{sMd})$ and $\mathtt{sMd} = smodeOf(\mathtt{Md})$.

Therefore, by Lemma 62 applied to (x), (iii), (v), and assumption $\Psi, EPids \vdash \Sigma_j : \mathsf{wf}$, we learn that $v = v_c$ where $N, V, \upsilon \vdash e \Downarrow v$ and $N_c, V_c, \upsilon \vdash e \Downarrow v_c$. Since $v = \mathsf{true}$, it follows that $N_c, V_c, \upsilon \vdash e \Downarrow \mathsf{true}$, and hence IF-TRUE-C applies:

$$\frac{\begin{array}{c} F_c = (pID_c, rID_c, ID_c, version_c, pc \rhd \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, E_c, \mathtt{Md}_c, \mathsf{if}\ e^{\Lambda}\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2) \\ [\![e]\!]_{\mathsf{N}_c, \mathsf{V}_c, \upsilon_c} = \mathsf{true} \qquad \mathsf{task}(ID, v, pr, vPol, rPol, \lambda x.c) \in PDecl \\ F'_c = (pID_c, rID_c, ID_c, version_c, pc \rhd \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \mathsf{end}_{\mathsf{if}} :: E_c, \mathtt{Md}_c, c_1) \end{array}}{PDecl \vdash I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c \longrightarrow I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c}\ \text{IF-TRUE-C}$$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$, where σ' is the stepped configuration. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K, N, V, S, F', Q)$
- $\sigma' = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c)$
- $F'_c = (pID_c, rID_c, ID_c, version_c, pc \rhd \overrightarrow{pr}_c, pcS_c, vPol_c, \upsilon_c, \mathsf{end}_{\mathsf{if}} :: E_c, \mathtt{Md}_c, c_1)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F'_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N \approx N_c$

(c) $\forall x', N'_k(x') \neq N_c(x'), N(x') = N_c(x')$ and $N(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V \approx V_c$

(e) $\forall x', V_k(x') \neq V_c(x'), V(x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES*, we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). By definition of F'_c we know that $F'_c|_{id} = F_c|_{id}$. Since $S'_e = S_e$, the result holds directly by (ii).

Ad (b). By inversion of EQUIV-N on (iii), we know that $N \setminus X_{nid} = N_c \setminus X_{nid}$. Since $ipID \in EPids'$, we proceed in cases on the local type of x. If it is Nid, then N-WT-EFF-NID applies, yielding that $N[x \mapsto (v, ipID)] \setminus (X_{nid} \cup \{x\}) = N_c[x \mapsto (v, ipID)] \setminus (X_{nid} \cup \{x\})$.

In the case where x's local type is Id, N-Wt-Eff-Id applies and we have that $N[x \mapsto (v, ipID)] \setminus X_{nid} = N_c[x \mapsto (v, ipID)] \setminus X_{nid}$. In both cases, Equiv-N then applies to establish the desired result.

Ad (c). For every $x' \neq x$, the desired result follows directly by (vi). Now suppose $x' = x$. If the mode is jit, then the result follows immediately (i.e., x does not satisfy the condition since it is written through). Otherwise, the write to x by the continuous execution trace differs from its value in the checkpointed context. By definition we know that $N[x \mapsto (v, ipID)](x) = N_c[x \mapsto (v, ipID)](x)$, and by assumption we know that $ipID \in EPids'$.

Ad (d). Note that $EPids = EPids'$ by cases on the mode of F, and since no new process is started when writing to a volatile location. Therefore, the desired result holds directly by (v).

Ad (e). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (vi) (no volatile locations are written).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have established that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$.

Case If-false-I. This case is similar to If-true-I, but uses the rule for else branching.

Case StartAtom-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \\ \mathsf{start}_{\mathsf{atom}}(\mathsf{aID}, \omega, \mu, \beta, \rho, Inits); c; \mathsf{end}_{\mathsf{atom}}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad K' = (N_k \leftsquigarrow N{\downarrow_\omega}, V_k, F_0 \rhd S_k) \qquad ipidOf(F) = ipID \\ vPol = (ShAcc, nIds, Inits) \qquad F_0 = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, 0, \omega, \mu, \rho, Inits, \cdot, \cdot), initN; c; \mathsf{end}_{\mathsf{atom}}) \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \Longrightarrow I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F_0, Q} \text{ StartAtom-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

Hence StartAtom-C applies:

$$\frac{F_c = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, \mathsf{start}_{\mathsf{atom}}(\cdots); c; \mathsf{end}_{\mathsf{atom}}) \qquad F'_c = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, c)}{PDecl \vdash I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c \longrightarrow I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c}\ \textsc{StartAtom-C}$$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$, where σ' is the stepped configuration. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K, N, V, S, F', Q)$
- $\sigma' = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c)$
- $F'_c = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(F_0 \rhd S_k, F_0 \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$ and $\mathsf{N}'_k = N_k \leftrightsquigarrow N{\downarrow_\omega}$:

(a) $S'_e|_{id} = F'_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N \approx N_c$

(c) $\forall x', N'_k(x') \neq N_c(x'), N(x') = N_c(x')$ and $N(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V \approx V_c$

(e) $\forall x', V_k(x') \neq V_c(x'), V(x') = V_c(x')$ and $V_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES*, we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). By definition of *getES* and (ii), we know that $S_e|_{id} = F_k \rhd S^0_e|_{id} = F_k \rhd S^0_e|_{id}$ s.t. $S^0_e|_{id} = S_c$. By definition of *getES*, we know that $S'_e|_{id} = F_0 \rhd S^0_e|_{id} = F_0 \rhd S^0_e|_{id}$. Since $S^0_e|_{id} = S_c$, it follows that $F_0 \rhd S^0_e|_{id} = F'_c \rhd S_c$. Since F_0 is effective and its top *pc* is Id by construction, it follows that $F_0 \rhd S^0_e|_{id} = F'_c \rhd S_c$, as desired.

Ad (b). Note that $EPids = EPids'$ by cases on the mode of F. Therefore, the result follows directly by (iii).

Ad (c). Let x be arbitrary, and first suppose that $x \in \omega$. It follows by definition of N'_k and by the JIT invariant from preservation that $N'_k(x) = N_c(x)$. Otherwise, if $x \notin \omega$, then $N_k(x) = N'_k(x)$ and the desired result follows from (iv). Since x was arbitrary, the result holds for all such x.

Ad (d). Since $EPids' = EPids$ and no new process is started when writing to a volatile location, the desired result holds directly by (v).

Ad (e). Since $EPids' = EPids$ and $S'_e = S_e$, the desired result follows directly from (vi) (no volatile locations are written).

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have established that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$.

Case EndAtom-I.

$$\frac{\begin{array}{c} F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw), \mathsf{end}_{\mathsf{atom}}) \\ F' = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathsf{jit}, \mathsf{skip}) \\ K = (N_k, V_k, F_k \rhd S_k) \qquad F_k = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \\ \mathsf{atom}(\mathsf{aID}, rb, \omega, \mu, \rho, Inits, NVw, Vw), c) \qquad V'_k = V_k \leftsquigarrow V{\downarrow_{Vw}} \\ N'_k = N_k \leftsquigarrow N{\downarrow_{NVw}} \qquad K' = (N'_k{\downarrow_{ckVarOf(S_k)}}, V'_k, F_J(pID) \rhd S_k) \qquad \mathsf{N}, \mathsf{V}, \upsilon_c \vdash res \Downarrow v \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{endAtom}(pID, \mathsf{aID}, rb, v)}{\Longrightarrow} I, IRQ, K', \mathsf{N}, \mathsf{V}, S, F', Q} \text{ EndAtom-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$

- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(F_k \rhd S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V_k(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EInpIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

Hence StartAtom-C applies:

$$\frac{F_c = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \texttt{Md}, \mathsf{end_{atom}}) \quad F'_c = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \texttt{Md}, \mathsf{skip})}{PDecl \vdash I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c \longrightarrow I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c}\ \textsc{EndAtom-C}$$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$, where σ' is the stepped configuration. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K, N, V, S, F', Q)$
- $\sigma' = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F'_c, Q_c)$
- $F'_c = (pID, rID, ID, version, \overrightarrow{pr}, vPol, \upsilon, E, \texttt{Md}, c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold, where $EPids', EIrqIds', EEvIds' \vdash getES(F_J(pID) \rhd S_k, F' \rhd S) = (S'_e, inIDs'_d, reIDs'_d)$:

(a) $S'_e|_{id} = F'_c \rhd S_c$

(b) $\Psi, PDecl, EPids' \vdash N \approx N_c$

(c) $\forall x', N'_k{\downarrow_{ckVarOf(S_k)}}(x') \neq N_c(x'), N(x') = N_c(x')$ and $N(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eNVWtsOf(S'_e)$

(d) $EPids' \vdash V \approx V_c$

(e) $\forall x', V'_k(x') \neq V_c(x'), V(x') = V_c(x')$ and $V'_k(x') = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x' \in eVWtsOf(S'_e)$

(f) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(g) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(h) $Q{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

By definition of F and F', and *getES*, we know that $S'_e = S_e$, $inIDs'_d = inIDs_d$, and $reIDs'_d = reIDs_d$.

Ad (a). By definition of *getES* and (ii), we know that $S_e|_{id} = F_k \rhd S^0_e|_{id} = F_k \rhd S^0_e|_{id}$ s.t. $S^0_e|_{id} = S_c$. By definition of *getES*, we know that $S'_e|_{id} = F_J(pID) \rhd S^0_e|_{id} = F_J(pID) \rhd S^0_e|_{id}$. Since $S^0_e|_{id} = S_c$, it follows that $F_J(pID) \rhd S^0_e|_{id} = F'_c \rhd S_c$. Since F_0 is effective and its top *pc* is Id by construction, it follows that $F_J(pID) \rhd S^0_e|_{id} = F'_c \rhd S_c$, as desired.

Ad (b). Note that $EPids = EPids'$ by cases on the mode of F. Therefore, the result follows directly by (iii).

Ad (c). Let x be arbitrary, and first suppose that $x \in NVw$. It follows by definition of N'_k that $N'_k(x) = N(x) = N_c(x)$. Additionally since $x \in NVw$, it was last written by the last atomic region execution, which is by definition an effective process. Otherwise, if $x \notin NVw$, then x was written by a previous process and the desired result follows from (iv). Since x was arbitrary, the result holds for all such x.

Ad (d). Since $EPids' = EPids$ and no new process is started when writing to a volatile location, the desired result holds directly by (v).

Ad (e). The proof is similar to (c), proceeding by cases on whether an arbitrary $x \in Vw$ or $x \notin Vw$.

Ad (f). By definition, $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$, and hence the desired result follows from (vii).

Ad (g). Since $EIrqIds' = EIrqIds$ and $inIDs'_d = inIDs_d$, we learn the desired result directly from (viii).

Ad (h). By definition, $EEvIds' = EEvIds$, and since $reIDs'_d = reIDs_d$, the desired result follows immediately from (ix).

In summary, we have established that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$.

□

E.10.1 Correctness

Theorem 20 (Intermittent correctness) $\vdash PDecl : \mathsf{ok}$, $\Sigma = (I, IRQ, K_{init}, \mathsf{N}_{init}, \mathsf{V}_{init}, \emptyset, F_{main}, \emptyset)$ $\sigma = (I, IRQ, \mathsf{N}_{init}, \mathsf{V}_{init}, \emptyset, F_{main}, \emptyset)$ *then* $\forall T_I \in \mathcal{R}_I(\Sigma)$, $\forall m \leq |T_I|$, $\exists T_C \in \mathcal{R}_C(\sigma)$, *s.t.* $T_I|_m \approx T_C$

Proof: By induction over m.

Base case $m = 0$**.** If $m = 0$, then the rule TEquiv-Base applies.

Inductive case $m = j + 1\ (\exists j \geq 0)$**.** By the IH, we know that $T_I|_j \approx T_C$. We want to find a trace T'_C such that $T_I|_{j+1} \approx T'_C$.

Let $\Sigma_j = T_I{\downarrow_j}$ and $\Sigma_{j+1} = T_I{\downarrow_{j+1}}$ where $\Sigma_j = (I, \mathit{IRQ}, K, \mathsf{N}, \mathsf{V}, S, F, c)$. We examine the step taken $\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1}$.

Case *iPidOf*$(F) \notin$ *EPids*. If F is an ineffective process, we apply Lemma 65 to relate the stepped intermittent configuration to the same continuous configuration (i.e., the ineffective step is not taken by the continuous execution trace). We obtain that

$$T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$$

By applying TEquiv-InEff,

$$\frac{T_I|_{j+1} = T_I|_j \overset{a}{\Longrightarrow} \Sigma_{j+1} \qquad \begin{array}{c} T_I|_j \approx T_C \qquad \Sigma_{j+1} = T_I{\downarrow_{j+1}} \\ \mathit{inEffective}(\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1}) \end{array} \qquad T_I|_{j+1} \vdash T_I{\downarrow_{j+1}} \approx \sigma_n}{T_I|_{j+1} \approx T_C}\ \textsc{TEquiv-InEff}$$

we obtain the desired result:

$$T_I|_{j+1} \approx T_C$$

Case *iPidOf*$(F) \in$ *EPids* and *topOfPCS*$(F) \neq \mathsf{Nid}$. If the step is effective and not occurring in an Nid branch, we apply Lemma 66 to relate the stepped intermittent configuration with the stepped continuous configuration (i.e., the effective step should be taken by the continuous execution trace). We obtain that

$$T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'$$

where $T_C(\sigma) \xrightarrow{a} \sigma'$.

By applying TEquiv-Eff,

$$\frac{\begin{array}{c} T_I{\downarrow_j} \approx T_C \qquad \Sigma_{j+1} = T_I{\downarrow_{j+1}} \qquad T_I|_{j+1} = T_I|_j \overset{a}{\Longrightarrow} \Sigma_{j+1} \\ \mathit{Effective}(\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1}) \qquad T_C(\sigma) \xrightarrow{a} \sigma' \end{array} \qquad T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma'}{T_I|_{j+1} \approx T_C \xrightarrow{a} \sigma'}\ \textsc{TEquiv-Eff}$$

we obtain the desired result:

$$T_I|_{j+1} \approx T_C \xrightarrow{a} \sigma'$$

Case power failure and reboot. Suppose that the last step in T_I is a reboot. To reason about the reboot state with the state before failure, we examine a combination of the rules PowerFail-I and Reboot-I.

$$\frac{\begin{array}{c} K = (N_k, V_k, S_k) \qquad K' = (N_k, V_k, updJitS(S_k, F \rhd S)) \\ (F' \rhd S') = incRb(updJitS(S_k, F \rhd S)) \qquad K'' = (N_k, V_k, F' \rhd S') \\ F = (pID, rID, ID, version, \overrightarrow{pr}, pcS, vPol, \upsilon, E, \mathtt{Md}, c) \\ polOf(F') \neq \mathsf{altOf}(\_) \qquad Q = Q_d, Q_k \qquad \forall \mathsf{tk} \in Q_d, polOf(\mathsf{tk}) = \mathsf{discard} \\ \forall \mathsf{tk} \in Q_k, polOf(\mathsf{tk}) = \mathsf{continue} \qquad ipidOf(F') = ipID' \end{array}}{\Psi, PDecl \vdash I, IRQ, K, \mathsf{N}, \mathsf{V}, S, F, Q \overset{\mathsf{rb}}{\Longrightarrow} I, IRQ, K', N_k^{ipID'} \rightsquigarrow \mathsf{N}, V_k^{ipID'}, S', F', Q_k} \text{ PowerFail-I/Reboot-I}$$

By assumption, we know that $T_I|_j \vdash \Sigma_j \approx \sigma$. Thus, by definition, where

- $\Sigma_j = T_I{\downarrow_j} = (I, IRQ, K, N, V, S, F, Q)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids = ePidOf(T_I|_j)$
- $EIrqIds = eIrqOf(IRQ, T_I)$
- $EEvIds = eEvOf(Q, T_I)$
- $EInpIds = eInpOf(I, T_I)$

we know that the following hold:

(i) $EPids, EIrqIds, EEvIds \vdash getES(S_k, F \rhd S) = (S_e, inIDs_d, reIDs_d)$

(ii) $S_e|_{id} = F_c \rhd S_c|_{id}$

(iii) $\Psi, PDecl, EPids \vdash N \approx N_c$

(iv) $\forall x, N_k(x) \neq N_c(x), N(x) = N_c(x)$ and $N(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eNVWtsOf(S_e)$

(v) $EPids \vdash V \approx V_c$

(vi) $\forall x, V(x) \neq V_c(x), V(x) = V_c(x)$ and $V(x) = (v, ipID)$ *s.t.* $ipID \in EPids$ and $x \in eVWtsOf(S_e)$

(vii) $I{\downarrow_{EInpIds}} = I_c{\downarrow_{EIrqIds}}$

(viii) $IRQ{\downarrow_{EIrqIds}} \cup T_I{\downarrow_{inIDs_d}} = IRQ_c{\downarrow_{EIrqIds}}$

(ix) $Q{\downarrow_{EEvIds}} \cup T_I{\downarrow_{reIDs_d}} = Q_c{\downarrow_{EEvIds}}$

We proceed to proving that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx \sigma$ by its definition. That is, letting

- $\Sigma_{j+1} = T_I{\downarrow_{j+1}} = (I, IRQ, K', N_k^{ipID'} \rightsquigarrow \mathsf{N}, V_k^{ipID'}, S', F', Q_k)$
- $\sigma = (I_c, IRQ_c, \mathsf{N}_c, \mathsf{V}_c, S_c, F_c, Q_c)$
- $EPids' = ePidOf(T_I|_{j+1})$
- $EIrqIds' = eIrqOf(IRQ, T_I)$
- $EEvIds' = eEvOf(Q_k, T_I)$
- $EInpIds' = eInpOf(I, T_I)$

we need to establish the following conditions hold:

(a) $EPids', EIrqIds', EEvIds' \vdash getES(F' \rhd S', F' \rhd S') = (S'_e, inIDs'_d, reIDs'_d)$

(b) $S'_e|_{id} = F_c \rhd S_c|_{id}$

(c) $\Psi, PDecl, EPids' \vdash N_k^{ipID'} \rightsquigarrow \mathsf{N} \approx N_c$

(d) $\forall x, N_k(x) \neq N_c(x), (N_k^{ipID'} \rightsquigarrow \mathsf{N})(x) = N_c(x)$ and $(N_k^{ipID'} \rightsquigarrow \mathsf{N})(x) = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x \in eNVWtsOf(S'_e)$

(e) $EPids' \vdash V_k^{ipID'} \approx V_c$

(f) $\forall x, V_k^{ipID'}(x) \neq V_c(x), V_k^{ipID'}(x) = V_c(x)$ and $V_k^{ipID'}(x) = (v, ipID)$ *s.t.* $ipID \in EPids'$ and $x \in eVWtsOf(S'_e)$

(g) $I{\downarrow_{EInpIds'}} = I_c{\downarrow_{EIrqIds'}}$

(h) $IRQ{\downarrow_{EIrqIds'}} \cup T_I{\downarrow_{inIDs'_d}} = IRQ_c{\downarrow_{EIrqIds'}}$

(i) $Q_k{\downarrow_{EEvIds'}} \cup T_I{\downarrow_{reIDs'_d}} = Q_c{\downarrow_{EEvIds'}}$

Ad (a). By definition of *updJitS*, we know that $S_e = updJitS(S_k, F \rhd S)$, and by definition of *getES* that *incRb* preserves the effectiveness of frames on S_e, and hence it follows that $S'_e = incRb(S_e)$ and $inIDs'_d = inIDs_d$ and $reIDs'_d = reIDs_d$. Therefore, the desired result follows by (i).

Ad (b). Since we know that $S'_e = incRb(S_e)$, the desired result follows by definition and from (ii): $S'_e|_{id} = incRb(S_e|_{id}) = incRb(F_c \rhd S_c|_{id}) = F_c \rhd S_c|_{id}$.

Ad (c). Let $x \in \mathsf{dom}(N)$ be arbitrary. There are two cases to consider: $N(x) = N_k \rightsquigarrow N(x)$ and otherwise. In the former, the rebooted process does not write to x, and we invert on the deriving rule and immediately apply the same rule to obtain the desired result. Otherwise, x is overwritten by the rebooted process $pidOf(F')$. Since $pidOf(F')$ just began its execution, it must not be in $EPids'$ of the subtrace $T_I|_{j+1}$. By definition of N_k, we know that $x \in cpOf(pidOf(F'))$, and hence it follows by N-Wt-Nid-Cp that $x \in X_{nid}$. By Equiv-N, the relation between memories continues to hold.

Ad (d). Here, we need to show that the condition holds for variables in $\mathsf{dom}(N_k)$ whose values are updated from the checkpointed context upon reboot. Suppose towards contradiction that there exists such an x s.t. $N_k(x) \neq N_c(x)$. Then it must be the case that $ipidOf(F') \in EPids'$, a contradiction. Therefore, there are no such x, and the condition for variables in $\mathsf{dom}(N) \setminus \mathsf{dom}(N_k \rightsquigarrow N)$ follows directly from (iv).

Ad (e). The reasoning is similar to (c). Since $pidOf(F') \notin EPids'$, it follows by V-NEff that $x \in X_{ineff}$, and the desired result holds vacuously by Equiv-V (i.e., of course $\cdot = \cdot$).

Ad (f). The proof is similar to (d). By contradiction, there is no such $x \in \mathsf{dom}(V_k^{ipID'})$ because $pidOf(F') \notin EPids'$. Therefore, the condition follows vacuously.

Ad (g). The desired result follows by (vii) and the observation that $EInpIds' = EInpIds$ and $EIrqIds' = EIrqIds$.

Ad (h). The desired result follows by (viii), $EIrqIds = EIrqIds'$, and $inIDs_d =$

$inIDs'_d$.

Ad (i). Since $reIDs = reIDs'$, it follows by the equivalence definition (DELAY-EV in particular) that $EEvIds = EEvIds'$. By definition of $eEvOf$, we know that $Q_k{\downarrow_{EEvIds'}} = Q{\downarrow_{EEvIds}}$. Hence, the desired result follows by (ix) and $reIDs' = reIDs$.

Case power failure and reboot with alternative. This case is similar to the above, but uses a combination of POWERFAIL-I and REBOOT-IA.

Case $iPidOf(F) \in EPids$ and $topOfPCS(F) = \mathsf{Nid}$. There are two additional subcases to consider:

Case $\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1}$ via IF-END-I. At the end of an Nid branch, both the intermittent and continuous execution traces take a step. We first step the intermittent execution trace via TEQUIV-NID-IF1, and then the continuous execution trace via TEQUIV-NID-IF2.

Stepping intermittent execution. Since $topOfPCS(F_\Sigma) = \mathsf{Nid}$, it follows that $nidIf(\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1})$ (i). In order to apply TEQUIV-NID-IF1, we need to show that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx T_C(\sigma)$.

By the IH, we know that $T_I|_j \approx T_C$ (ii). By inversion of TEQUIV-NID-IF1 on (ii), we know that $T_I|_j \vdash \Sigma_j \approx T_C(\sigma)$. By Lemma 63, we obtain $T_I|_{j+1} \vdash \Sigma_{j+1} \approx T_C(\sigma)$ (iii).

Lastly, we apply TEQUIV-NID-IF1 to (i), (ii), and (iii) to establish the desired result:

$$T_I|_{j+1} \approx T_C \ (iv)$$

Stepping continuous execution. Since $topOfPCS(F_\sigma) = \mathsf{Nid}$, it follows that $nidIf(\sigma \xrightarrow{a} \sigma')$ (v). In order to apply TEQUIV-NID-IF2, we need to show that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx T_C(\sigma')$, which we proceed to prove by its definition. By Lemma 64, the desired result holds:

$$T_I|_{j+1} \vdash \Sigma_{j+1} \approx T_C(\sigma') \ (vi)$$

Therefore, it follows by TEQUIV-NID-IF2 applied to (iv), (v), and (vi) that

$$T_I|_{j+1} \approx T_C \xrightarrow{a} \sigma'$$

Case otherwise. By the IH, we know that $T_I|_j \approx T_C$ (i). By inversion of TEQUIV-NID-IF1 on (i), we learn that $T_I|_j \vdash \Sigma_j \approx T_C(\sigma)$ (ii). By application of Lemma 63 to (ii), we learn that $T_I|_{j+1} \vdash \Sigma_{j+1} \approx T_C(\sigma)$ (iii). Since $topOfPCS(F_\Sigma) = \mathsf{Nid}$, it follows that $nidIf(\Sigma_j \overset{a}{\Longrightarrow} \Sigma_{j+1})$ (iv).

Therefore, the desired result follows by TEQUIV-NID-IF1 applied to (i), (iii), and (iv):

$$T_I|_{j+1} \approx T_C$$

□

We write $T{\downarrow_{\mathsf{res}}}$ as the sequence of results of T.

Corollary 2 (Intermittent results correctness)
$\vdash PDecl : \mathsf{ok}$, $\Sigma = (I, IRQ, K_{init}, \mathsf{N}_{init}, \mathsf{V}_{init}, \emptyset, F_{main}, \emptyset)$, $\sigma = (I, IRQ, \mathsf{N}_{init}, \mathsf{V}_{init}, \emptyset, F_{main}, \emptyset)$ *imply* $\forall T_I \in \mathcal{R}_I(\Sigma)$, $\exists T_C \in \mathcal{R}_C(\sigma)$, *s.t.* $T_I{\downarrow_{\mathsf{res}}} = T_C{\downarrow_{\mathsf{res}}}$.

Proof: Let $T_I \in \mathcal{R}_I(\Sigma)$ be arbitrary and $m = |T_I|$. By Theorem 20, we know that $\exists T_C \in \mathcal{R}_C(\sigma)$ s.t. $T_I \approx T_C$. By definition of trace equivalence, in all cases, we have that the last state of traces T_I and T_C are related: $T_I \vdash T_I(\Sigma_f) \approx T_C(\sigma_f)$. Letting $\Sigma_f = (I_I, IRQ_I, K_I, N_I, V_I, S_I, F_I, Q_I)$ and $\sigma_f = (I_c, IRQ_c, K_c, N_c, V_c, S_c, F_c, Q_c)$, it follows by definition that $\Psi, PDecl, EPids, \mathsf{Id} \vdash N_I \approx N_c$, meaning that the final memories are equivalent in their idempotent values. Likewise, it follows that $EPids \vdash V_I \approx V_c$, meaning that the volatile memories between these two configurations are equivalent in their idempotent values. Since $\vdash PDecl : \mathsf{ok}$, all tasks are well-typed, and hence all effective processes running intermittently generate fully idempotent *res*. Since the intermittent and continuous memories are the same in their idempotent values, it follows that $T_I{\downarrow_{\mathsf{res}}} = T_C{\downarrow_{\mathsf{res}}}$. □