# High-Level Small-Step Operational Semantics for Transactions (Technical Companion)

Katherine F. Moore, Dan Grossman

July 15, 2007

**Abstract**

This document is the technical companion to our POPL'08 submission of the same name.

## 1 Navigational Guide

This document gives the full definitions and proofs for the AtomsFamily languages presented in our POPL'08 submission. As explained in more detail in the rest of this introductory section:

- Each section presents a different language. Subsections are used consistently across subsections. Figures at the end of the document summarize the languages' differences.
- For historical reasons, the first language we present is StrongNestedParallel and most other languages are defined in terms of differences from StrongNestedParallel. Most notably, Weak has the thread-pools necessary for internal parallelism but the type system ensures they remain empty (whereas, in our paper, these thread-pools are removed from the syntax and run-time state). StrongBasic was created later for expository purposes and to simplify the equivalence proof for WeakUndo.
- For historical reasons, there are some syntactic differences, particularly the names we use for effects.
- Some theorems are stated more generally than in the paper and have different names. A table below maps each theorem in the paper to the corresponding theorem in this document.

Despite these minor surface-level differences, this document contains complete proofs and definitions, including the semantics for WeakOnCommit.

### 1.1 Organization and Overview

*Organization:* This document is organized by language, with each subsection following a parallel structure. Figure 1 provides a quick section and subsection guide for each language in the AtomsFamily. There is a section for each language, and all sections have similar subsections to preserve parallel structure. For a comprehensive overview of all the languages, refer to the languages-at-a-glance guide in Section 8.

*Overview:* In this document, we present a (relatively) comprehensive comparison of different languages for software transactions. We include an entire family of languages (affectionately dubbed the AtomsFamily, because transactions are syntactically encapsulated in *atomic* blocks). We present 6 different languages, each of which has a slightly different set of transactional properties (e.g. internal parallelism, weak or strong atomicity) and prove equivalence properties among the languages.

Section 2 introduces the first AtomsFamily language we defined, StrongNestedParallel. This language is the starting point for subsequent sections. It models strong atomicity with internally parallel transactions. Section 3 introduces a very similar model to StrongNestedParallel, called Weak. Weak has the same syntax and a similar operational semantics to StrongNestedParallel, and in Section 3.7 we provide a proof of language equivalence given a partition on the heap.

We have also extended the semantics of Weak to allow rollback (we call this language WeakUndo) and have obtained a similar equivalence result. We also have a semantics for software transactions with

| Language Name | Section |
|---|---|
| StrongNestedParallel | 2 |
| Weak | 3 |
| StrongBasic | 4 |
| StrongUndo | 5 |
| WeakUndo | 6 |
| WeakOnCommit | 7 |

| Content | **Sub**section Number |
|---|---|
| Syntax | 1 |
| Evaluation Rules | 2 |
| Type-Checking | 3 |
| Other Typing Rules | 4 |
| Activeness | 5 |
| Type Safety | 6 |
| Equivalence | 7 |

Figure 1: AtomsFamily Guide

lazy update and a commit phase, WeakOnCommit. The final type system and equivalence result for WeakOnCommit are future work.

In hindsight, StrongBasic is more of a "base language" than StrongNestedParallel, but for historical reasons we defined StrongNestedParallel first and proved it equivalent to Weak (rather than defining a weak-atomicity language without the vestiges of internal parallelism).

## 1.2 Syntax Differences between the TR and the POPL submission

1. $\varepsilon$: The effects used in this document are represented as 0, 2, and $\emptyset$. For clarity, we chose to use the more meaningful names ot, wt, and emp, respectively, in our POPL submission.
2. $e$: Although most syntax differences are minor, a few notables ones should be listed here. The expression syntax for spawn expressions used in this document has numerical subscripts 0, 1, and 2 for tl, oc, and ip, respectively. The operational semantics and typing rules have name changes accordingly.
3. Weak has vestiges. For historical reasons, the Weak language has extra *empty* thread-pools in this document. In our POPL'08 submission, the judgment form of Weak is like StrongBasic, $(a; H; e \rightarrow a'; H'; e'; T)$. In this document, it looks like StrongNestedParallel, only we enforce that the final two thread-pools are $\cdot$ using the type system $(a; H; e \rightarrow a'; H'; e'; T; \cdot; \cdot)$.
4. In this document, all our operational rules are defined without evaluation contexts. In the POPL'08 submission, we used evaluation contexts to define many of the inductive rules concisely.

## 1.3 Result and Citation Reference

For the readers' convenience, we have created a quick reference to facilitate easy navigation from theorems and citations found in our POPL'08 paper to their corresponding material in this document.

| POPL'08 | Tech Report |
|---|---|
| Theorem 3.1 (StrongNestedParallel Type Safety) | Section 2.6 |
| Lemma 3.2 (StrongNestedParallel Progress) | Theorem 2.1 |
| Lemma 3.3 (StrongNestedParallel Preservation | Theorem 2.3 |
| Theorem 4.1 (StrongBasic/ Weak Equivalence) | Section 3.7 / Theorem 3.9 |
| Lemma 5.1 (WeakUndo/ StrongBasic Equivalence) | Follows from Theorems 6.5 and 5.13 |
| Lemma 5.2 (WeakUndo/ StrongUndo Equivalence) | Theorem 6.5 |
| Lemma 5.3 (StrongUndo/ StrongBasic Equivalence) | Theorem 5.13 |
| WeakOnCommit | Section 7 |
| Activeness judgment from Figure 7 | Defined in each section; See in particular Section 2.5 |

# 2 The StrongNestedParallel language

The StrongNestedParallel language for software transactions models strong atomicity with internal parallelism inside transactions. We present semantics that require very few changes to model weak atomicity in the Weak language.

## 2.1 Syntax

The following syntax is used as the starting point for every language in the AtomsFamily. We do not distinguish source programs from run-time states in this document. (See our paper for the subset that would correspond to source programs.)

We annotate ref-types with a $t$ and have $\Gamma$ map labels to a type and $t$. The type systems for StrongNestedParallel (and StrongBasic ) ignore these annotations. The type systems for each of our weak languages use these annotations to partition the heap.

$$\begin{array}{rcl}
e & ::= & c \mid l \mid x \mid e_1; e_2 \mid e_1 := e_2 \mid \text{ref } e \mid !e \mid \lambda x.e \mid e_1\ e_2 \mid \mathsf{if}\ e_1\ e_2\ e_3 \\
 & & \mid \mathsf{spawn}_0\ e \mid \mathsf{spawn}_1\ e \mid \mathsf{spawn}_2\ e \mid \mathsf{atomic}\ e \mid \mathsf{inatomic}(a, e, T_1, T_2) \\
v & ::= & c \mid l \mid \lambda x.e \\
H & ::= & \cdot \mid H, l \mapsto v \\
T & ::= & \cdot \mid T \parallel e \\
a & ::= & \circ \mid \bullet \\
\tau & ::= & \text{int} \mid \text{ref}_t \tau \mid \tau \xrightarrow{\varepsilon} \tau' \\
t & ::= & 0 \mid 2 \\
\varepsilon & ::= & t \mid \emptyset \\
\Gamma & ::= & \cdot \mid \Gamma, l : (\tau, t) \mid \Gamma, x : \tau
\end{array}$$

## 2.2 Dynamic Evaluation

### 2.2.1 Whole Program Evaluation

In this section, we present the evaluation rule under which a program in the StrongNestedParallel language steps. Although the rule is slightly different for each member of the AtomsFamily, the form of the judgment never changes.

$$\boxed{a; H; T \rightarrow\ a'; H'; T'}$$

PROGRAM

$$\frac{a; H; e \rightarrow\ a'; H'; e'; T_0; T_1; \cdot}{a; H; T_A \parallel e \parallel T_B \rightarrow\ a'; H'; T_A \parallel e' \parallel T_B \parallel T_0 \parallel T_1}$$

As a result, the initial and terminal configurations for programs in the AtomsFamily are always the same: The **initial configuration** of a source program $e$ has no partially completed transactions and an empty heap. The **terminal configuration** for a program state contains no further computation (i.e. all expressions are values) and requires that no expression can be executing in a transaction.

Initial Configuration: $\circ; \cdot; e$ Terminal Configuration: $\circ; H; \overline{v}$

### 2.2.2 Expression Evaluation

In this section, we present the evaluation rules for the StrongNestedParallel language. Note that the form of the evaluation judgment can create three new thread-pools.

$\boxed{a; H; e \rightarrow\ a'; H'; e'; T_0; T_1; T_2}$

SEQ-1
$$\frac{a; H; e_1 \rightarrow\ a'; H'; e_1'; T_0; T_1; T_2}{a; H; e_1; e_2 \rightarrow\ a'; H'; e_1'; e_2; T_0; T_1; T_2}$$

SEQ-V
$$\frac{}{a; H; v; e_2 \rightarrow\ a; H; e_2; \cdot; \cdot; \cdot}$$

IF-1
$$\frac{a; H; e_1 \rightarrow\ a'; H'; e_1'; T_0; T_1; T_2}{a; H; \mathsf{if}\ e_1\ e_2\ e_3 \rightarrow\ a'; H'; \mathsf{if}\ e_1'\ e_2\ e_3; T_0; T_1; T_2}$$

IF-Z
$$\frac{}{a; H; \mathsf{if}\ 0\ e_2\ e_3 \rightarrow\ a'; H'; e_3; \cdot; \cdot; \cdot}$$

IF-NZ
$$\frac{c \neq 0}{a; H; \mathsf{if}\ c\ e_2\ e_3 \rightarrow\ a'; H'; e_2; \cdot; \cdot; \cdot}$$

SET-1
$$\frac{a; H; e_1 \rightarrow\ a'; H'; e_1'; T_0; T_1; T_2}{a; H; e_1 := e_2 \rightarrow\ a'; H'; e_1' := e_2; T_0; T_1; T_2}$$

SET-2
$$\frac{a; H; e_2 \rightarrow\ a'; H'; e_2'; T_0; T_1; T_2}{a; H; l := e_2 \rightarrow\ a'; H'; l := e_2'; T_0; T_1; T_2}$$

STRONG-SET
$$\frac{}{\circ; H; l := v \rightarrow\ \circ; H, l \mapsto v; v; \cdot; \cdot; \cdot}$$

REF-1
$$\frac{a; H; e \rightarrow\ a'; H'; e'; T_0; T_1; T_2}{a; H; \mathrm{ref}\ e \rightarrow\ a'; H'; \mathrm{ref}\ e'; T_0; T_1; T_2}$$

ALLOC
$$\frac{l \notin \mathrm{Dom}(H)}{a; H; \mathrm{ref}\ v \rightarrow\ a; H, l \mapsto v; l; \cdot; \cdot; \cdot}$$

GET-1
$$\frac{a; H; e \rightarrow\ a'; H'; e'; T_0; T_1; T_2}{a; H; !e \rightarrow\ a'; H'; !e'; T_0; T_1; T_2}$$

STRONG-GET
$$\frac{}{\circ; H; !l \rightarrow\ \circ; H; H(l); \cdot; \cdot; \cdot}$$

APP-1
$$\frac{a; H; e_1 \rightarrow\ a'; H'; e_1'; T_0; T_1; T_2}{a; H; e_1\ e_2 \rightarrow\ a'; H'; e_1'\ e_2; T_0; T_1; T_2}$$

APP-2
$$\frac{a; H; e_2 \rightarrow\ a'; H'; e_2'; T_0; T_1; T_2}{a; H; v\ e_2 \rightarrow\ a'; H'; v\ e_2'; T_0; T_1; T_2}$$

BETA
$$\frac{}{a; H; (\lambda x.e)\ v_2 \rightarrow\ a; H; e[v_2/x]; \cdot; \cdot; \cdot}$$

SPAWN 0
$$\frac{}{a; H; \mathsf{spawn}_0\ e \rightarrow\ a; H; 0; e; \cdot; \cdot}$$

SPAWN 1
$$\frac{}{a; H; \mathsf{spawn}_1\ e \rightarrow\ a; H; 0; \cdot; e; \cdot}$$

SPAWN 2
$$\frac{}{a; H; \mathsf{spawn}_2\ e \rightarrow\ a; H; 0; \cdot; \cdot; e}$$

ENTER ATOMIC
$$\frac{}{\circ; H; \mathsf{atomic}\ e \rightarrow\ \bullet; H; \mathsf{inatomic}(\circ, e, \cdot, \cdot); \cdot; \cdot; \cdot}$$

INATOMIC
$$\frac{a; H; e \rightarrow\ a'; H'; e'; \cdot; T_1'; T_2'}{\bullet; H; \mathsf{inatomic}(a, e, T_1, T_2) \rightarrow\ \bullet; H; \mathsf{inatomic}(a', e', (T_1 \parallel T_1'), (T_2 \parallel T_2')); \cdot; \cdot; \cdot}$$

INATOMIC HELPER
$$\frac{a; H; e \rightarrow\ a'; H'; e'; \cdot; T_1'; T_2'}{\bullet; H; \mathsf{inatomic}(a, e_0, T_1, (T_2 \parallel e \parallel T_2'')) \rightarrow\ \bullet; H; \mathsf{inatomic}(a', e_0, (T_1 \parallel T_1'), (T_2 \parallel e' \parallel T_2'' \parallel T_2')); \cdot; \cdot; \cdot}$$

EXIT ATOMIC
$$\frac{}{\bullet; H; \mathsf{inatomic}(\circ, v, T_1, \overline{v}) \rightarrow\ \circ; H; v; \cdot; T_1; \cdot}$$

Strong atomicity is enforced by requiring that $a = \circ$ in order to access $H$. We facilitate internal parallelism by nesting the values of $a$ inside transactions so that the parallel siblings can govern themselves.

## 2.3 Typecheck $e$

In this section, we give a type-and-effect system to type a single thread (or expression, $e$) under effect $\varepsilon$ in the StrongNestedParallel language. This type-and-effect system prohibits undesirable forms of spawn in the wrong place (see the paper for an informal description).

$\boxed{\Gamma;\varepsilon \vdash e : \tau}$

$$\textsc{T-Const}\ \frac{}{\Gamma;\varepsilon \vdash c : \text{int}} \qquad \textsc{T-Var}\ \frac{}{\Gamma;\varepsilon \vdash x : \Gamma(x)} \qquad \textsc{T-Label}\ \frac{\Gamma(l) = (\tau, t)}{\Gamma;\varepsilon \vdash l : \text{ref}_t\tau} \qquad \textsc{T-Seq}\ \frac{\Gamma;\varepsilon \vdash e_1 : \tau_1 \qquad \Gamma;\varepsilon \vdash e_2 : \tau_2}{\Gamma;\varepsilon \vdash e_1; e_2 : \tau_2}$$

$$\textsc{T-If}\ \frac{\Gamma;\varepsilon \vdash e_1 : \text{int} \qquad \Gamma;\varepsilon \vdash e_2 : \tau \qquad \Gamma;\varepsilon \vdash e_3 : \tau}{\Gamma;\varepsilon \vdash \mathsf{if}\ e_1\ e_2\ e_3 : \tau} \qquad \textsc{T-Set}\ \frac{\Gamma;\varepsilon \vdash e_1 : \text{ref}_t\tau \qquad \Gamma;\varepsilon \vdash e_2 : \tau}{\Gamma;\varepsilon \vdash e_1 := e_2 : \tau}$$

$$\textsc{T-Ref}\ \frac{\Gamma;\varepsilon \vdash e : \tau}{\Gamma;\varepsilon \vdash \text{ref}\ e : \text{ref}_t\tau} \qquad \textsc{T-Get}\ \frac{\Gamma;\varepsilon \vdash e : \text{ref}_t\tau}{\Gamma;\varepsilon \vdash !e : \tau} \qquad \textsc{T-Lambda}\ \frac{\Gamma, x : \tau_1; \varepsilon' \vdash e : \tau_2 \qquad \text{not-active(e)}}{\Gamma;\varepsilon \vdash \lambda x.e : \tau_1 \xrightarrow{\varepsilon'} \tau_2}$$

$$\textsc{T-App}\ \frac{\Gamma;\varepsilon \vdash e_1 : \tau_1 \xrightarrow{\varepsilon'} \tau_2 \qquad \Gamma;\varepsilon \vdash e_2 : \tau_1 \qquad \varepsilon' \leq \varepsilon}{\Gamma;\varepsilon \vdash e_1\ e_2 : \tau_2}$$

$$\textsc{T-Spawn-0}\ \frac{\Gamma; 0 \vdash e : \tau}{\Gamma; 0 \vdash \mathsf{spawn}_0\ e : \text{int}} \qquad \textsc{T-Spawn-1}\ \frac{\Gamma; 0 \vdash e : \tau}{\Gamma;\varepsilon \vdash \mathsf{spawn}_1\ e : \text{int}} \qquad \textsc{T-Spawn-2}\ \frac{\Gamma; 2 \vdash e : \tau}{\Gamma; 2 \vdash \mathsf{spawn}_2\ e : \text{int}}$$

$$\textsc{T-Atomic}\ \frac{\Gamma; 2 \vdash e : \tau}{\Gamma;\varepsilon \vdash \mathsf{atomic}\ e : \tau}$$

$$\textsc{T-InAtomic}\ \frac{\Gamma; 2 \vdash e : \tau \qquad \Gamma; 0 \vdash T_1 \qquad \Gamma; 2 \vdash T_2 \qquad \text{not-active}(T_1) \qquad \text{correct-atomic}(a, e \parallel T_2)}{\Gamma;\varepsilon \vdash \mathsf{inatomic}(a, e, T_1, T_2) : \tau}$$

## 2.4 Other Typing Rules

Here, we present rules to type-check $\varepsilon$, $T$, $H$, and the program state $(a; H; T)$.

$\boxed{\varepsilon \leq \varepsilon'}$

$$\frac{}{\varepsilon \leq \varepsilon} \qquad \frac{}{\emptyset \leq \varepsilon}$$

$\boxed{\Gamma;\varepsilon \vdash T}$

$$\frac{}{\Gamma;\varepsilon \vdash \cdot} \qquad \frac{\Gamma;\varepsilon \vdash T \qquad \Gamma;\varepsilon \vdash e : \tau}{\Gamma;\varepsilon \vdash T \parallel e}$$

$\boxed{\Gamma \vdash H : \Gamma'}$

$$\frac{}{\Gamma \vdash \cdot : \cdot} \qquad \frac{\Gamma \vdash H : \Gamma' \qquad \Gamma;\varepsilon \vdash v : \tau}{\Gamma \vdash H, l \mapsto v : \Gamma', l : (\tau, t)}$$

$\boxed{\vdash a; H; T}$

$$\textsc{Top-Level}\ \frac{\Gamma \vdash H : \Gamma \qquad \Gamma; 0 \vdash T \qquad \text{correct-atomic}(a, T)}{\vdash a; H; T}$$

## 2.5 Activeness

To determine if a program state is valid, we must be able to decide if the value of $a$ corresponds appropriately to the number of threads currently executing in a transaction. The following definitions are used to determine when this is the case.

### 2.5.1 Not-Active($e$)

If not-active($e$) then $e$ is not currently executing a transaction and is has no subexpression containing a partially completed transaction.

$\boxed{\text{not-active}(e)}$

$$\frac{}{\text{not-active}(c)} \qquad \frac{}{\text{not-active}(l)} \qquad \frac{}{\text{not-active}(x)} \qquad \frac{\text{not-active}(e_1) \qquad \text{not-active}(e_2)}{\text{not-active}(e_1; e_2)}$$

$$\frac{\text{not-active}(e_1) \qquad \text{not-active}(e_2) \qquad \text{not-active}(e_3)}{\text{not-active}(\mathsf{if}\ e_1\ e_2\ e_3)} \qquad \frac{\text{not-active}(e_1) \qquad \text{not-active}(e_2)}{\text{not-active}(e_1 := e_2)}$$

$$\frac{\text{not-active}(e)}{\text{not-active}(!e)} \qquad \frac{\text{not-active}(e)}{\text{not-active}(\text{ref } e)} \qquad \frac{\text{not-active}(e_1)}{\text{not-active}(\lambda x.e_1)} \qquad \frac{\text{not-active}(e_1) \qquad \text{not-active}(e_2)}{\text{not-active}(e_1\ e_2)}$$

$$\frac{\text{not-active}(e)}{\text{not-active}(\mathsf{spawn}_0\ e)} \qquad \frac{\text{not-active}(e)}{\text{not-active}(\mathsf{spawn}_1\ e)} \qquad \frac{\text{not-active}(e)}{\text{not-active}(\mathsf{spawn}_2\ e)} \qquad \frac{\text{not-active}(e)}{\text{not-active}(\mathsf{atomic}\ e)}$$

### 2.5.2 Active($e$)

If active($e$) then $e$ is currently executing a transaction and is well-formed in the sense that other than a single topmost partially completed transaction, it obeys the same structural sensibility of the definition of not-active($e$).

$\boxed{\text{active}(e)}$

$$\frac{}{\text{active}(\mathsf{inatomic}(a, e, T_1, T_2))} \qquad \frac{\text{active}(e_1) \qquad \text{not-active}(e_2)}{\text{active}(e_1; e_2)}$$

$$\frac{\text{active}(e_1) \qquad \text{not-active}(e_2) \qquad \text{not-active}(e_3)}{\text{active}(\mathsf{if}\ e_1\ e_2\ e_3)} \qquad \frac{\text{active}(e_1) \qquad \text{not-active}(e_2)}{\text{active}(e_1 := e_2)} \qquad \frac{\text{active}(e_2)}{\text{active}(l := e_2)}$$

$$\frac{\text{active}(e)}{\text{active}(\text{ref } e)} \qquad \frac{\text{active}(e)}{\text{active}(!e)} \qquad \frac{\text{active}(e_1) \qquad \text{not-active}(e_2)}{\text{active}(e_1\ e_2)} \qquad \frac{\text{active}(e_2)}{\text{active}(v\ e_2)}$$

### 2.5.3 Correct Atomic State of $T$

not-active($T$) and active($T$) lift not-active($e$) and active($e$) to thread-pools in the natural way. correct-atomic($a, T$) ensures that $a$ correctly describes whether or not some thread in $T$ is currently executing a transaction.

$\boxed{\text{not-active}(T)}$

$$\frac{}{\text{not-active}(\cdot)} \qquad \frac{\text{not-active}(T_1) \qquad \text{not-active}(e) \qquad \text{not-active}(T_2)}{\text{not-active}(T_1 \parallel e \parallel T_2)}$$

$\boxed{\text{active}(T)}$

$$\frac{\text{not-active}(T_1) \qquad \text{active}(e) \qquad \text{not-active}(T_2)}{\text{active}(T_1 \parallel e \parallel T_2)}$$

$\boxed{\text{correct-atomic}(a, T)}$

$$\frac{\text{not-active}(T)}{\text{correct-atomic}(\circ, T)} \qquad \frac{\text{active}(T)}{\text{correct-atomic}(\bullet, T)}$$

## 2.6 Type Safety

**Theorem 2.1 (Top Level Progress)** *If $\vdash a; H; T$, then either $\exists \overline{v}$ such that $T = \overline{v}$ or $\exists a'; H', T'$ such that $a; H; T \rightarrow a'; H'; T'$*

**Proof** There is only one typing rule for program states, TOP-LEVEL. Using this rule, correct-atomic$(a, T)$ is true by inversion, and the language definition provides two possible values for $a$:

1. If $a = \circ$ then the definition of correct-atomic provides not-active$(T)$. By inversion, every $e_i \in T$ is not-active. Either every $e_i$ is a value, or there is at least one $e_i$ that is not a value. If every $e_i \in T$ is a value, then $T = \overline{v}$. If there is at least one $e_i \in T$ that is not a value then the lemma for single thread progress states that $e_i$ must be able to step under $a = \circ$ when not-active$(e_i)$. This provides a step for $T$ using the rule PROGRAM.
2. If $a = \bullet$ then the definition of correct-atomic provides active$(T)$. By inversion, there is exactly 1 $e_0 \in T$ such that active$(e_0)$. Single thread progress states that when $a = \bullet$ and active$(e_0)$, $e_0$ can step. This provides a step for $T$ using the rule PROGRAM.

**Lemma 2.2 (Single Thread Progress)** *If $\Gamma \vdash H : \Gamma$, then each of the following must be true:*

1. *If $\Gamma; \varepsilon \vdash e : \tau$, and* active*(e) then $\exists e', a', H', T_0, T_1, T_2$ such that $\bullet; H; e \rightarrow a'; H'; e'; T_0; T_1; T_2$.*
2. *If $\Gamma; \varepsilon \vdash e : \tau$, and* not-active*(e) then $e$ is a value or $\exists e', a', H', T_0, T_1, T_2$ such that $\circ; H; e \rightarrow a'; H'; e'; T_0; T_1; T_2$.*
3. *If $\Gamma; \varepsilon \vdash T$ and correct-atomic(a, T) then $T$ is some $T_A \parallel e \parallel T_B$ such that $a; H; e \rightarrow a'; H'; e'; T_0; T_1; T_2$ or $T$ is all values.*

**Proof** By mutual induction on the typing derivation of $e$ or $T$: This proof is organized by cases on the final rule of the derivation, with part 1 handling the case where active$(e)$ and part 2 handling the case where not-active$(e)$. A proof of part 3 is after proofs of parts 1 and 2.

- T-CONST $e = c$
  1. There is no way to derive active(c). This case is vacuously true.
  2. $e$ is a value, so this is trivial.
- T-VAR $e = x$. In this case, even though $\Gamma \vdash H : \Gamma$, $x \notin dom(\Gamma)$ by the Variables not in $\Gamma$ lemma. As a result, $x$ cannot type check, and thus this case is vacuous.
- T-LABEL $e = l$
  1. There is no way to derive active$(l)$. This case is vacuously true.
  2. $e$ is a value, so this is trivial.
- T-LAMBDA $e = \lambda x.e_1$
  1. There is no way to derive active$(e)$. This case is vacuously true.
  2. $e$ is a value, so this is trivial.
- T-SEQ $e = e_1; e_2$
  1. There is only one way to derive active$(e)$. By inversion, active$(e)$ implies active$(e_1)$ and not-active$(e_2)$. This can only happen when $e_1$ is not a value. By induction, $e_1$ can step under $\bullet$ and evaluation rule SEQ-1 provides a valid step for $e$.
  2. The only way to derive not-active$(e)$ is when not-active$(e_1)$ and not-active$(e_2)$. There are two possibilities:
     (a) $e_1$ is not a value. By induction, $e_1$ can step, so by the rule SEQ-1, $e$ can also step.
     (b) $e_1$ is a value. The rule SEQ-V provides a valid step for $e$.

- T-IF $e = \mathsf{if}\ e_1\ e_2\ e_3$
  1. There is only one way to derive active($e$). By inversion, active($e$) implies active($e_1$), not-active($e_2$), and not-active($e_3$). This can only happen when $e_1$ is not a value. By induction, $e_1$ can step under $\bullet$ and the evaluation rule IF-1 provides a valid step for $e$.
  2. The only way to derive not-active($e$) is when not-active($e_1$), not-active($e_2$) and not-active($e_3$). There are two possibilities:
     (a) $e_1$ is not a value. By induction, $e_1$ can step, so by the rule IF-1, $e$ can also step.
     (b) $e_1$ is a value. Using the canonical forms lemma tells us that $e_1$ must be some $c$, and depending on the value of $c$, the rules IF-Z and IF-NZ provide two possible steps for $e$.

- T-SET $e = e_1 := e_2$.
  1. There are two ways to derive active($e$):
     (a) $e_1$ is not a value and active($e_1$) and not-active($e_2$). By induction, $e_1$ can step. By applying the evaluation rule SET-1, $e$ can also step.
     (b) $e_1$ is some $l$ and $e_2$ is not a value, and active($e_2$). By induction, $e_2$ can step. By applying the evaluation rule SET-2, $e$ can also step.
  2. By inversion, not-active($e$) implies not-active($e_1$) and not-active($e_2$). There are three cases to consider:
     (a) $e_1$ is not a value. By induction, $e_1$ can step. By applying the evaluation rule SET-1, $e$ can also step.
     (b) $e_1$ is some $l$ and $e_2$ is not a value. By induction, $e_2$ can step. By applying the evaluation rule SET-2, $e$ can also step.
     (c) $e_1$ is some $l$ and $e_2$ is some $v$. We need a step for $e$ from $\circ; H; e$ which is provided by the evaluation rule STRONG-SET.

- T-REF $e = \text{ref}\ e_1$
  1. There is only one way to derive active($e$). By inversion, active($e$) implies active($e_1$). This can only happen when $e_1$ is not a value. By induction, $e_1$ can step under $\bullet$ and the evaluation rule REF-1 can be used to derive a step for $e$.
  2. By inversion, not-active($e$) implies not-active($e_1$). There are two sub-cases:
     (a) $e_1$ is not a value. By induction, $e_1$ can step so rule REF-1 provides a step for $e$.
     (b) $e_1$ is some $v$. In this case, ALLOC provides a way for $e$ to step.

- T-GET $e = !e_1$
  1. There is only one way to derive active($e$). By inversion, active($e$) implies active($e_1$). This can only happen when $e_1$ is not a value. By induction, $e_1$ can step under $\bullet$ so rule GET-1 provides a step for $e$.
  2. By inversion, not-active($e$) implies not-active($e_1$). There are two sub-cases:
     (a) $e_1$ is not a value. By induction, $e_1$ can step so rule GET-1 provides a step for $e$.
     (b) $e_1$ is some $l$. We need a step for $e$ from $\circ; H; e$ which is provided by the evaluation rule STRONG-GET.

- T-APP $e = e_1\ e_2$
  1. There are two ways to derive active($e$).
     - $e_1$ is not a value and active($e_1$). By induction, $e_1$ can step, and the rule APP-1 provides a step for $e$.
     - $e_1$ is some $v$ and active($e_2$). By induction, $e_2$ can step, and the rule APP-2 provides a step for $e$.
  2. By inversion, not-active($e$) implies not-active($e_1$) and not-active($e_2$). There are three sub-cases:
     - $e_1$ is not a value. By induction, $e_1$ can step, and the rule APP-1 provides a step for $e$.
     - $e_1$ is some $v$ and $e_2$ is not a value. By induction, $e_2$ can step, and the rule APP-2 provides a step for $e$.

  - $e_1$ is some $v$ and $e_2$ is some $v'$. By inversion, $\Gamma; \varepsilon \vdash e_1 : \tau_1 \xrightarrow{\varepsilon'} \tau_2$, and the canonical forms lemma states that $e_1$ must be some $\lambda x.e_3$. The evaluation rule BETA provides a step for $e$.

- T-SPAWN-0 $e = \mathsf{spawn}_0\ e_1$.
  1. This case is vacuous because there is no way to derive active($e$).
  2. The evaluation rule SPAWN 0 provides a valid step for $e$.
- T-SPAWN-1 $e = \mathsf{spawn}_1\ e_1$.
  1. This case is vacuous because there is no way to derive active($e$).
  2. The evaluation rule SPAWN 1 provides a valid step for $e$.
- T-SPAWN-2 $e = \mathsf{spawn}_2\ e_1$.
  1. This case is vacuous because there is no way to derive active($e$).
  2. The evaluation rule SPAWN 2 provides a valid step for $e$.
- T-ATOMIC $e = \mathsf{atomic}\ e_1$.
  1. This case is vacuous because there is no way to derive active($e$).
  2. The rule ENTER ATOMIC applies, giving $\circ; H; \mathsf{atomic}\ e_1 \rightarrow\ \bullet; H; \mathsf{inatomic}(\circ, e_1, \cdot, \cdot); \cdot; \cdot; \cdot$ as a valid step for $e$.
- T-INATOMIC $e = \mathsf{inatomic}(a'', e_1, T_1', T_2')$
  1. Inversion on the typing rule implies correct-atomic($a''$, $e_1 \parallel T_2'$) and that $e_1$ and $T_2'$ type check under $\varepsilon = 2$. There are three sub-cases to consider:
     (a) If $a'' = \bullet$ and $e_1$ is not a value or $T_2'$ contains at least one expression that is not a value, part 3 of this lemma states that there must be a step for some $e_0 \in e_1 \parallel T_2'$. Depending on the location of $e_0$, there are two sub-cases.
        i. It is $e_1$ that actually takes the step via INATOMIC. In this case, because $\Gamma; 2 \vdash e_1 : \tau$ it must be that $\varepsilon \neq 0$, which by the effects lemma enforces that $T_0 = \cdot$. Thus the entire inatomic expression can step.
        ii. It is some $e_0 \in T_2'$ that actually takes the step via INATOMIC HELPER. In this case, because $\Gamma; 2 \vdash T_2$ it must be that $\varepsilon \neq 0$, which by the effects lemma enforces that $T_0 = \cdot$. Thus the entire inatomic expression can step.
     (b) $a'' = \circ$ and there is some non-value $e_0$ in ($e_1 \parallel T_2'$). This case is similar to the cases when $a'' = \bullet$.
     (c) $a'' = \circ$ and $e'$ is some $v$ and $T_2' = \overline{v}$. It is sufficient to find a step for $e$ when $a = \bullet$. The evaluation rule EXIT ATOMIC provides a valid step.
  2. This case is vacuous because there is no way to derive not-active($e$).
- Part 3 of this lemma is stated again here, with the proof following. If $\Gamma; \varepsilon \vdash T$ and $\Gamma \vdash H : \Gamma$ and correct-atomic(a, $T$) then $T$ is some $T_A \parallel e \parallel T_B$ such that $a; H; e \rightarrow a; H; e'; T_0; T_1; T_2$ or $T$ is all values. There are two cases to consider:
  - If $a = \bullet$, then by inversion on correct-atomic($\bullet, T$) it must be that active($T$). By inversion on active($T$), $T = T_A \parallel e \parallel T_B$ with active($e$) and not-active($T_A$) and not-active($T_B$). By part 1 of this lemma, there is some $a', H', e', T_0, T_1, T_2$ such that $\bullet; H; e \rightarrow\ a'; H'; e'; T_0; T_1; T_2$.
  - If $a = \circ$, then by inversion on correct-atomic($\circ, T$) it must be that not-active($T$). By inversion on not-active($T$), $T = T_A \parallel e \parallel T_B$ where not-active($e$), not-active($T_B$) and not-active($T_B$). By part 2 of this lemma, either $e$ is some $v$ or there is some $a', H', e', T_0, T_1, T_2$ such that $\circ; H; e \rightarrow\ a'; H'; e'; T_0; T_1; T_2$. If every such $e$ is a value and thus cannot step, then $T$ is all values.

**Theorem 2.3 (Top Level Preservation)** *If $\Gamma \vdash H : \Gamma$ and $\Gamma; 0 \vdash T$ and correct-atomic($a, T$) and $a; H; T \rightarrow a'; H'; T'$, then there exists some $\Gamma'$ extending $\Gamma$ such that:*

1. $\Gamma'; 0 \vdash T'$
2. $\Gamma' \vdash H' : \Gamma'$
3. *correct-atomic($a', T'$)*

**Proof** In this case, there is also only one evaluation rule; PROGRAM, which gives $a; H; T_A \parallel e_i \parallel T_B \rightarrow a'; H'; T_A \parallel e'_i \parallel T_B \parallel T_0 \parallel T_1$. $\Gamma'$ is obtained via Lemma 2.4.

1. $T' = T_A \parallel e'_i \parallel T_B \parallel T_0 \parallel T_1$. By inversion it must be the case that $\Gamma; 0 \vdash T_A \parallel T_B$. The weakening lemma gives $\Gamma'; 0 \vdash T_A \parallel T_B$. By the single threaded preservation lemma, it is also the case that $\Gamma'; 0 \vdash e'_i \parallel T_0 \parallel T_1$. Thus, $\Gamma'; 0 \vdash T'$.
2. $\Gamma' \vdash H' : \Gamma'$ by Lemma 2.4.
3. By assumption, correct-atomic$(a, T)$. Also, Lemma 2.4 provides not-active$(T_0 \parallel T_1)$. There are several possibilities for $a$ and $T$ before evaluating $e_i$ one step.
   - $a = \circ$ implies that not-active$(e_i \parallel T_A \parallel T_B)$. If $a' = \bullet$ then by single thread preservation it must be that active$(e'_i)$. If $a' = \circ$ then by single thread preservation it must be that not-active$(e'_i \parallel T_0 \parallel T_1)$. Thus, correct-atomic$(a', T')$.
   - $a = \bullet$ and active$(e_i)$ which means not-active$(T_A \parallel T_B)$. If $a' = \bullet$ then by single thread preservation it must be that active$(e'_i)$. If $a' = \circ$ then by single thread preservation it must be that not-active$(e'_i)$. Thus, correct-atomic$(a', T')$.
   - $a = \bullet$ and not-active$(e_i)$ which means active$(T_A)$ or active$(T_B)$. Since the cases for $T_A$ and $T_B$ are symmetric, assume active$(T_A)$ and not-active$(T_B)$. Since $T_A$ did not change during the step, by Lemma 2.4 $a' = \bullet$ and active$(T_A)$ and not-active$(T_B)$. Single thread preservation gives that not-active$(e'_i)$. Thus, correct-atomic$(a', T')$.

**Lemma 2.4 (Single Thread Preservation)** *If* $a; H; e \rightarrow \; a'; H'; e'; T_0; T_1; T_2$ *and* $\Gamma; \varepsilon \vdash e : \tau$ *and* $\Gamma \vdash H : \Gamma$ *and one of the following:*

1. $a = \circ$ *and* not-active*(e)*
2. $a = \bullet$ *and* not-active*(e)*
3. $a = \bullet$ *and* active*(e)*

*then* $\exists \Gamma'$ *extending* $\Gamma$ *such that:*

1. $\Gamma'; \varepsilon \vdash e' : \tau$
2. $\Gamma' \vdash H' : \Gamma'$
3. not-active$(T_0 \parallel T_1 \parallel T_2)$
4. $\Gamma'; 0 \vdash T_0$ *and*
   $\Gamma'; 0 \vdash T_1$ *and*
   $\Gamma'; 2 \vdash T_2$
5. *All the following (though exactly one is not vacuous):*
   - *If* $a = a'$ *and* active*(e) then* active*(e')*
   - *If* $a = a'$ *and* not-active*(e) then* not-active*(e')*
   - *If* $a = \circ$ *and* $a' = \bullet$ *then* not-active*(e) and* active*(e')*
   - *If* $a = \bullet$ *and* $a' = \circ$ *then* active*(e) and* not-active*(e')*

**Proof** by induction on the typing derivation of $e$ by cases on the final rule used in the derivation.

- T-VAR $e = x$. There are no evaluation rules for $x$, and thus $e$ cannot step. This case is vacuously true.
- T-CONST $e = c$. $e$ is a value, and thus cannot step. This case is vacuously true.
- T-LABEL $e = l$. $e$ is a value, and thus cannot step. This case is vacuously true.
- T-LAMBDA $e = \lambda x.e_1$. $e$ is a value, and thus cannot step. This case is vacuously true.
- T-SEQ $e = e_1; e_2$. There are two possible steps from $e$ to $e'$.
  - Under SEQ-1 $e' = e'_1; e_2$.
    1. By induction, it must be that if $\Gamma; \varepsilon \vdash e_1 : \tau_1$ then $\Gamma'; \varepsilon \vdash e'_1 : \tau_1$. The type for $e'$ is preserved under T-SEQ because $\Gamma'; \varepsilon \vdash e'_1 : \tau_1$ by induction, and the weakening lemma states that $\Gamma; \varepsilon \vdash e_2 : \tau$ implies $\Gamma'; \varepsilon \vdash e_2 : \tau_2$ so $\Gamma'; \varepsilon \vdash e' : \tau_2$.

2. $\Gamma' \vdash H' : \Gamma'$ by induction.
3. not-active$(T_0 \parallel T_1 \parallel T_2)$ by induction.
4. $\Gamma'; 0 \vdash T_0$, and $\Gamma'; 0 \vdash T_1$, and $\Gamma'; 2 \vdash T_2$ by induction.
5. * If $a = a'$ and active$(e)$ then by inversion active$(e_1)$ and not-active$(e_2)$. By induction, active$(e_1)$ implies active$(e_1')$. From active$(e_1')$ and not-active$(e_2)$ it is clear that active$(e)$.
   * If $a = a'$ and not-active$(e)$ then by inversion not-active$(e_1)$ and not-active$(e_2)$. By induction when $a = a'$ and not-active$(e_1)$, then not-active$(e_1')$ From not-active$(e_1')$ and not-active$(e_2)$ it is clear that not-active$(e')$.
   * If $a = \circ$ and $a' = \bullet$ then by induction not-active$(e_1)$ and active$(e_1')$. By assumption, $a = \circ$ implies not-active$(e)$. By inversion, not-active$(e)$ gives not-active$(e_2)$. It is possible to derive active$(e')$ from active$(e_1')$ and not-active$(e_2)$.
   * If $a = \bullet$ and $a' = \circ$ then by induction active$(e_1)$ and not-active$(e_1')$. By assumption, $a = \bullet$ implies either active$(e)$ or not-active$(e)$. However, given active$(e_1)$, there is no way to derive not-active$(e)$ so this case is vacuous. It must be that active$(e)$. Inversion on the derivation of active$(e)$ gives not-active$(e_2)$. It is possible to derive not-active$(e')$ from not-active$(e_1')$ and not-active$(e_2)$.

– Under SEQ-V when $e_1$ is some $v$ and $e' = e_2$. The SEQ-V rule produces the following thread-pools and heap: $T_0 = T_1 = T_2 = \cdot$, and $H' = H$.

1. Since $\Gamma; \varepsilon \vdash e : \tau_2$ is given, inverting the typing rule gives $\Gamma; \varepsilon \vdash e_2 : \tau_2$. Because $\Gamma' = \Gamma$, it must also be that $\Gamma'; \varepsilon \vdash e_2 : \tau_2$ and thus $\Gamma'; \varepsilon \vdash e' : \tau_2$.
2. This rule doesn't change $\Gamma$ or the heap, so $\Gamma' = \Gamma$ and $H' = H$. By assumption, $\Gamma \vdash H : \Gamma$ and thus $\Gamma' \vdash H' : \Gamma'$.
3. not-active$(\cdot \parallel \cdot \parallel \cdot)$
4. $\Gamma'; \varepsilon \vdash \cdot$ for any $\varepsilon$.
5. $a = a'$ It is sufficient to show that not-active$(e)$ implies not-active$(e')$ and active$(e)$ implies active$(e')$. Assuming not-active$(e)$, gives not-active$(e_2)$ by inversion. Since $e' = e_2$, it must be that not-active$(e')$. The case for active$(e)$ is vacuous because there is no way to derive active$(v; e_2)$.

• T-IF $e = \mathsf{if}\ e_1\ e_2\ e_3$. There are three possible steps from $e$ to $e'$.

– Under IF-1 $e' = \mathsf{if}\ e_1'\ e_2\ e_3$.

1. By induction, it must be that if $\Gamma; \varepsilon \vdash e_1 : \text{int}$ then $\Gamma'; \varepsilon \vdash e_1' : \tau_1$. The type for $e'$ is preserved under T-IF because $\Gamma'; \varepsilon \vdash e_1' : \text{int}$ by induction, and the weakening lemma states that $\Gamma; \varepsilon \vdash e_2 : \tau$ implies $\Gamma'; \varepsilon \vdash e_2 : \tau$ and $\Gamma; \varepsilon \vdash e_3 : \tau$ implies $\Gamma'; \varepsilon \vdash e_3 : \tau$ so $\Gamma'; \varepsilon \vdash e' : \tau_2$.
2. $\Gamma' \vdash H' : \Gamma'$ by induction.
3. not-active$(T_0 \parallel T_1 \parallel T_2)$ by induction.
4. $\Gamma'; 0 \vdash T_0$, and $\Gamma'; 0 \vdash T_1$, and $\Gamma'; 2 \vdash T_2$ by induction.
5. * If $a = a'$ and active$(e)$ then by inversion active$(e_1)$, not-active$(e_2)$, and not-active$(e_3)$. By induction, active$(e_1)$ implies active$(e_1')$. From active$(e_1')$, not-active$(e_2)$, and not-active$(e_3)$. it is clear that active$(e)$.
   * If $a = a'$ and not-active$(e)$ then by inversion not-active$(e_1)$, not-active$(e_2)$, and not-active$(e_3)$. By induction when $a = a'$ and not-active$(e_1)$, then not-active$(e_1')$ From not-active$(e_1')$, not-active$(e_2)$, and not-active$(e_3)$ it is clear that not-active$(e')$.
   * If $a = \circ$ and $a' = \bullet$ then by induction not-active$(e_1)$ and active$(e_1')$. By assumption, $a = \circ$ implies not-active$(e)$. By inversion, not-active$(e)$ gives not-active$(e_2)$ and not-active$(e_3)$. It is possible to derive active$(e')$ from active$(e_1')$, not-active$(e_2)$, and not-active$(e_3)$.
   * If $a = \bullet$ and $a' = \circ$ then by induction active$(e_1)$ and not-active$(e_1')$. By assumption, $a = \bullet$ implies either active$(e)$ or not-active$(e)$. However, given active$(e_1)$, there is no way to derive not-active$(e)$ so this case is vacuous. It must be that active$(e)$. Inversion on the derivation of active$(e)$ gives not-active$(e_2)$ and not-active$(e_3)$. It is possible to derive not-active$(e')$ from not-active$(e_1')$, not-active$(e_2)$, and not-active$(e_3)$.

- Under IF-Z when $e_1 = 0$ and $e' = e_3$. The IF-Z rule produces the following thread-pools and heap: $T_0 = T_1 = T_2 = \cdot$, and $H' = H$.
  1. Since $\Gamma; \varepsilon \vdash e : \tau$ is given, inverting the typing rule gives $\Gamma; \varepsilon \vdash e_3 : \tau$. Because $\Gamma' = \Gamma$, it must also be that $\Gamma'; \varepsilon \vdash e_3 : \tau$ and thus $\Gamma'; \varepsilon \vdash e' : \tau_3$.
  2. This rule doesn't change $\Gamma$ or the heap, so $\Gamma' = \Gamma$ and $H' = H$. By assumption, $\Gamma \vdash H : \Gamma$ and thus $\Gamma' \vdash H' : \Gamma'$.
  3. not-active$(\cdot \parallel \cdot \parallel \cdot)$
  4. $\Gamma'; \varepsilon \vdash \cdot$ for any $\varepsilon$.
  5. $a = a'$ It is sufficient to show that not-active$(e)$ implies not-active$(e')$ and active$(e)$ implies active$(e')$. Assuming not-active$(e)$, gives not-active$(e_3)$ by inversion. Since $e' = e_3$, it must be that not-active$(e')$. The case for active$(e)$ is vacuous because there is no way to derive active$(0)$.
- Under IF-NZ when $e_1 = c$ and $e' = e_2$. This case is identical to the prior one.

- T-SET $e = e_1 := e_2$. There are three ways $e$ could have become $e'$.
  - Under SET-1 when $e_1$ steps to $e_1'$, then $e' = e_1' := e_2$. The proof of this case resembles that for SEQ-1.
  - Under SET-2 when $e_1$ is some $l$ and $e_2$ steps to $e_2'$, then $e' = l := e_2'$. The proof of this case resembles that for SEQ-1.
  - Under STRONG-SET $e = l := v$ and $e' = v$, and $T_0 = T_1 = T_2 = \cdot$. Pick $\Gamma' = \Gamma$.
    1. Inversion on the typing rule gives $\Gamma; \varepsilon \vdash l : \text{ref}_t\tau$ and $\Gamma; \varepsilon \vdash v : \tau$. Since $\Gamma' = \Gamma$, it must be that $\Gamma'; \varepsilon \vdash l : \text{ref}_t\tau$ and $\Gamma'; \varepsilon \vdash v : \tau$. Since $e' = v$, $\Gamma'; \varepsilon \vdash e' : \tau$.
    2. Since $H' = H[l \mapsto v]$, then $H'(l) = v$. By choosing $\Gamma' = \Gamma$, $\Gamma'; \varepsilon \vdash v : \tau$ and $\Gamma'(l) = \text{ref}_t\tau$. Thus, $\Gamma' \vdash H' : \Gamma'$.
    3. not-active$(\cdot \parallel \cdot \parallel \cdot)$
    4. $\Gamma; \varepsilon \vdash \cdot$ is true for any $\varepsilon$.
    5. Because $e$ stepped under STRONG-SET, $a = a' = \circ$, and not-active$(e)$ is derived from not-active$(l)$ and not-active$(v)$. Since $e' = v$, we know not-active$(e')$. There is no way to derive active$(e)$, so that case is vacuous.

- T-REF If $e = \text{ref } e_0$ There are two evaluation rules that could have taken $e$ to $e'$.
  - Under REF-1, $e_0$ steps to $e_0'$ giving $e' = \text{ref } e_0'$. This case is similar to the SEQ-1 case.
  - Under ALLOC, $e_0$ is some $v$ and $e'$ is some $l$. Pick $\Gamma' = \Gamma, l \mapsto \text{ref}_t\tau$ (the choice of $t$ is irrelevant for this language).
    1. Inversion on the typing rule gives $\Gamma; \varepsilon \vdash e : \text{ref}_t\tau$ and $\Gamma; \varepsilon \vdash v : \tau$. Since $\Gamma' = \Gamma, l : \text{ref}_t\tau$, it must be that $\Gamma'; \varepsilon \vdash l : \text{ref}_t\tau$ and $\Gamma'; \varepsilon \vdash v : \tau$. Since $e' = v$, we get $\Gamma'; \varepsilon \vdash e' : \tau$.
    2. By assumption, $\Gamma \vdash H : \Gamma$. By the weakening lemma, $\Gamma' \vdash H : \Gamma'$. However, $H' = H, l \mapsto v$ since $\Gamma'(l) = \text{ref}_t\tau$ when $H'(l) = v$, and $\Gamma; \varepsilon \vdash v : \tau$ then $\Gamma' \vdash H' : \Gamma'$.
    3. not-active$(\cdot \parallel \cdot \parallel \cdot)$
    4. $\Gamma; \varepsilon \vdash \cdot$ is true for any $\varepsilon$.
    5. Because $e$ stepped under ALLOC, $a = a'$ If $a = a'$, and not-active$(e)$, then it is sufficient to show not-active(e'). In this case, not-active$(l)$ can be derived with no assumptions. There is no way to derive active$(e)$, so that case is vacuous.

- T-APP $e = e_1\ e_2$ There are three evaluation rules that could take $e$ to $e'$.
  - Under APP-1, $e_1$ steps to $e_1'$ giving $e' = e_1'\ e_2$. This case is similar to the SEQ-1 case.
  - Under APP-2, $e_1$ is some $v$ and $e_2$ steps to $e_2'$ giving $e' = v\ e_2'$. This case is similar to the SEQ-1 case.
  - Under BETA, $e_1$ is some $\lambda x.e_3$ and $e_2$ is some $v$. In this case, $e' = e_3[v/x]$, $T_0 = T_1 = T_2 = \cdot$ and $\Gamma' = \Gamma$.
    1. The Substitution Lemma gives $\Gamma'; \varepsilon' \vdash e_3[v/x] : \tau$. Since $\varepsilon' \leq \varepsilon$, the Weakening Lemma gives $\Gamma'; \varepsilon \vdash e_3[v/x] : \tau$.
    2. Here, $H' = H$, and $\Gamma \vdash H : \Gamma$, so $\Gamma' \vdash H' : \Gamma'$.

3. not-active($\cdot \parallel \cdot \parallel \cdot$)
4. $\Gamma; \varepsilon \vdash \cdot$ is true for any $\varepsilon$.
5. In this case, $a = a'$. Because there is no way to derive active($e$), that case is vacuous. Thus, it is sufficient to show that not-active($e$) gives not-active($e'$). By inversion on the rule used to derive not-active($\lambda x.e_3$), it must be that not-active($e_3$). The Substitution Lemma gives that not-active($e_3[v/x]$), thus not-active($e'$).

- T-GET $e = !e_0$ There are two evaluation rules that could take $e$ to $e'$.
  - Under GET-1, $e_0$ steps to $e_0'$ giving $e' = !e_0'$. This case is similar to the SEQ-1 case.
  - Under STRONG-GET $e = !l$ and $e' = v$. This case is similar to the STRONG-SET case.
- T-SPAWN-0 $e = \mathsf{spawn}_0\ e_0$. Thus, $\Gamma; \varepsilon \vdash e : \text{int}$. There is only one evaluation rule for $\mathsf{spawn}_0\ e_0$, SPAWN 0. After $e$ steps to $e'$ using this rule, $e' = 0$, $\Gamma' = \Gamma$, $H' = H$, $T_0 = e_0$, and $T_1 = T_2 = \cdot$.
  1. T-CONST provides $\Gamma'; \varepsilon \vdash 0 : \text{int}$.
  2. $\Gamma' = \Gamma$ and $H' = H$, and by assumption, $\Gamma \vdash H : \Gamma$. Thus $\Gamma' \vdash H' : \Gamma'$.
  3. By assumption, active($e$) or not-active($e$). There is no way to derive active($e$) for spawn expressions, so it must be that not-active($e$) and thus not-active($e_0$). Also, $T_0 = e_0$ and $T_1 = T_2 = \cdot$, so not-active($T_0 \parallel \cdot \parallel \cdot$).
  4. By inversion on the typing rule T-SPAWN-0, it is also the case that $\Gamma; 0 \vdash e_0 : \tau$. Because $T_0 = e_0$, it must be that $\Gamma'; 0 \vdash T_0$. Since $\Gamma'; \varepsilon \vdash \cdot$ is true for any $\varepsilon$ we know that $T_1$ and $T_2$ type-check under the required effect.
  5. The SPAWN 0 rule enforces that $a = a'$. If not-active($e$) then it must be shown that not-active($e'$). Since not-active(0) requires no assumption and $e' = 0$, this is trivial. The case for active($e$) is vacuous because there is no way to derive active($\mathsf{spawn}_0\ e_0$).
- T-SPAWN-1 $e = \mathsf{spawn}_1\ e_1$. This case is similar to that for T-SPAWN-0.
- T-SPAWN-2 $e = \mathsf{spawn}_2\ e_0$. This case is similar to that for T-SPAWN-0.
- T-ATOMIC $e = \mathsf{atomic}\ e_0$. If $e$ steps, it must be via the evaluation rule ENTER ATOMIC, giving $e' = \mathsf{inatomic}(a'', e_0, \cdot, \cdot)$, and $H' = H$, where $\Gamma' = \Gamma$. Also, $T_0 = T_1 = T_2 = \cdot$.
  1. $\Gamma; \varepsilon \vdash e_0 : \tau$ is true by inversion on the typing rule, and $\Gamma'; \varepsilon \vdash e_0 : \tau$ since $\Gamma' = \Gamma$.
  2. $H$ is unchanged, so $\Gamma' \vdash H : \Gamma'$ because $\Gamma' = \Gamma$.
  3. not-active($\cdot \parallel \cdot \parallel \cdot$)
  4. $\Gamma'; \varepsilon \vdash \cdot$ for any $\varepsilon$.
  5. Here, $a = \circ$ and $a' = \bullet$. The correct atomicity is preserved because we assume not-active($e$) when $a = \circ$, and similarly active($e'$) requires no assumptions.
- T-INATOMIC $e = \mathsf{inatomic}(a'', e_0, T_1', T_2')$ with correct-atomic($a'', e_0 \parallel T_2'$). In this case, there are three possible steps from $e$ to $e'$:
  - Under INATOMIC, the distinguished expression $e_0$ steps: $a''; H; e_0 \rightarrow a'''; H'; e_0'; \cdot; T_1''; T_2''$ so $e' = \mathsf{inatomic}(a''', e_0', (T_1' \parallel T_1''), (T_2' \parallel T_2''))$. $T_0 = T_1 = T_2 = \cdot$ and $\Gamma'$ and $H'$ are obtained through induction.
    1. $\Gamma'; 2 \vdash e_0' : \tau$ by induction. Also by induction, $\Gamma'; 2 \vdash T_2''$. By inversion and the weakening lemma, $\Gamma'; 2 \vdash T_2'$ Thus $\Gamma'; 2 \vdash T_2' \parallel T_2''$. not-active($T_1''$) is given by induction, and not-active($T_1'$) by inversion on the typing rule, so not-active($T_1' \parallel T_1$). Similarly $\Gamma'; 0 \vdash T_1''$ by induction and $\Gamma'; 0 \vdash T_1'$ by inversion and the weakening lemma, giving $\Gamma'; 0 \vdash T_1' \parallel T_1''$ All that is still needed that correct-atomic($a''', e_0' \parallel T_2' \parallel T_2''$). Inversion on the typing rule gives correct-atomic($a'', e_0 \parallel T_2'$). Also, not-active($T_2''$) by induction. By the assumptions about $a$ and active or not-active, there are three sub-cases:
       (a) If $a'' = \circ$ then by inversion on the derivation of correct-atomic($\circ, e_0 \parallel T_2'$), it must be that not-active($e_0$) and not-active($T_2'$). After the step, it must be that either $a'''\circ$ and not-active($e_0'$) or $a''' = \bullet$ and active($e_0'$). Either is true by induction. Since $T_2'$ has not changed, it must be that correct-atomic($a''', e_0 \parallel T_2'$), and thus correct-atomic($a''', e_0 \parallel T_2' \parallel T_2''$).

(b) If $a'' = \bullet$ then by inversion on the derivation of correct-atomic($\circ, e_0 \parallel T_2'$), either active($e_0$) or active($T_2'$). In this case, assume active($T_2$) and not-active($e_0$). After $e_0$ steps, it must be that $a''' = \bullet$ and not-active($e_0'$). Thus, correct-atomic($a''', e_0 \parallel T_2'$), and correct-atomic($a''', e_0 \parallel T_2' \parallel T_2'''$).

(c) If $a'' = \bullet$ then by inversion on the derivation of correct-atomic($\circ, e_0 \parallel T_2'$), it must be that either active($e_0$) or active($T_2'$). In this case, assume not-active($T_2$) and active($e_0$). After $e_0$ steps, $a''' = \bullet$ and active($e_0'$) or $a''' = \circ$ and not-active($e_0'$) by induction. Thus, correct-atomic($a''', e_0 \parallel T_2'$), and correct-atomic($a''', e_0 \parallel T_2' \parallel T_2''$).

As a result, it must be that correct-atomic($a''', e_0 \parallel T_2' \parallel T_2''$). From this it can be derived that $\Gamma'; \varepsilon \vdash e' : \tau$

2. $\Gamma' \vdash H' : \Gamma'$ by induction.
3. not-active($\cdot \parallel \cdot \parallel \cdot$)
4. $\Gamma'; \varepsilon \vdash \cdot$ for any $\varepsilon$.
5. Here, $a = a'$. The case where not-active($e$) is vacuous because there is no way to derive not-active($e$). Showing that active($e$) implies active($e'$) is trivial because active($e'$) requires no assumptions.

– Under INATOMIC HELPER, $T_2' = T_2'' \parallel e_1 \parallel T_2'''$ and $a''; H; e_1 \rightarrow a'''; H'; e_1'; \cdot; T_1; T_2$ so $e' =$ inatomic$(a''', e_0, T_1', (T_2 \parallel T_2'' \parallel e_1 \parallel T_2'''))$. This case is similar to that when $e$ steps under INATOMIC.

– Under EXIT ATOMIC, $a'' = \circ$, $e_0 = v$, and $T_2 = \overline{v}$. As a result, $a; H; e \rightarrow \ \circ; H; v; \cdot; T_1; \cdot$. Thus, $e' = v$, $\Gamma' = \Gamma$ and $H' = H$. Also, $T_0 = T_2 = \cdot$.

1. By inversion, $e_0$ is some $v$ with type $\tau$, and $\Gamma; 2 \vdash v : \tau$. The values effectless lemma (and $\Gamma' = \Gamma$) provides that $\Gamma'; \varepsilon \vdash v : \tau$. Because $e' = v$ then $\Gamma'; \varepsilon \vdash e' : \tau$.
2. By assumption, $\Gamma \vdash H : \Gamma$. Also, $\Gamma' = \Gamma$ and $H' = H$, so $\Gamma' \vdash H' : \Gamma'$.
3. By inversion on the typing derivation, not-active($T_1$) is given. Thus, not-active($\cdot \parallel T_1 \parallel \cdot$).
4. By inversion, $\Gamma; 0 \vdash T_1$ so by the weakening lemma $\Gamma'; 0 \vdash T_1$. For $T_0$ and $T_2$, $\Gamma'; \varepsilon \vdash \cdot$ for any $\varepsilon$.
5. Here, $a = \bullet$ and $a' = \circ$. The correct atomicity is preserved because active($e$) can be derived with no assumptions, and not-active($e'$) can also be derived with no assumptions.

**Definition 2.5 (Context Extension)** *$\Gamma'$ extends $\Gamma$ if for every $x$ or $l \in Dom(\Gamma)$, $\Gamma'(x) = \Gamma(x)$ and $\Gamma'(l) = \Gamma(l)$.*

**Lemma 2.6 (Weakening Lemma)** *1. If $\Gamma; \varepsilon \vdash e : \tau$ and $\Gamma'$ extends $\Gamma$ and $\varepsilon \leq \varepsilon'$, then $\Gamma'; \varepsilon' \vdash e : \tau$*

*2. If $\Gamma; \varepsilon \vdash T$ and $\Gamma'$ extends $\Gamma$ and $\varepsilon \leq \varepsilon'$, then $\Gamma'; \varepsilon' \vdash T$.*

**Proof** 1. By mutual induction on the typing derivation of $e$ or $T$, by cases on the final typing rule.

- T-VAR $\Gamma; \varepsilon \vdash x : \tau$. In this case, $\Gamma; \varepsilon \vdash x : \Gamma(x)$, where $\Gamma(x) = \tau$. In this case, $x \in Dom(\Gamma)$. By the definition of $\Gamma'$ extending $\Gamma$, it must be that $x \in Dom(\Gamma')$ and $\Gamma'(x) = \Gamma(x)$. Thus, $\Gamma'; \varepsilon' \vdash x : \Gamma'(x)$ and $\Gamma'(x) = \tau$. Note the effect is irrelevant in T-VAR.
- T-CONST $\Gamma; \varepsilon \vdash c :$ int. In this case, neither $\Gamma$ nor $\varepsilon$ is used in the typing derivation of $e$. This case is immediate.
- T-LABEL $\Gamma; \varepsilon \vdash l : \text{ref}_t\tau$. By inversion, $\Gamma(l) = (\tau, t)$. In order to extend $\Gamma$, $\Gamma'(l) = \Gamma(l)$. Thus, $\Gamma'; \varepsilon' \vdash l : \text{ref}_t\tau$ can be derived. Note the effect is irrelevant.
- T-LAMBDA $\Gamma; \varepsilon \vdash \lambda x.e_1 : \tau_1 \xrightarrow{\varepsilon''} \tau_2$. By inversion, $\Gamma, x \mapsto \tau_1; \varepsilon'' \vdash e_1 : \tau_2$. By induction, $\Gamma', x \mapsto \tau_1; \varepsilon'' \vdash e_1 : \tau_2$ so $\Gamma'; \varepsilon' \vdash \lambda x.e_1 : \tau_1 \xrightarrow{\varepsilon''} \tau_2$. Note the effect is irrelevant.
- T-SEQ $\Gamma; \varepsilon \vdash e_1; e_2 : \tau_2$. By inversion, $\Gamma; \varepsilon \vdash e_1 : \tau_1$ and $\Gamma; \varepsilon \vdash e_2 : \tau_2$. So by induction, $\Gamma'; \varepsilon' \vdash e_1 : \tau_1$ and $\Gamma'; \varepsilon' \vdash e_2 : \tau_2$. Thus, $\Gamma'; \varepsilon' \vdash e_1; e_2 : \tau_2$.
- T-IF This case is very similar to the T-SEQ case.
- T-SET This case is very similar to the T-SEQ case.
- T-REF This case is very similar to the T-SEQ case.
- T-GET This case is very similar to the T-SEQ case.

- T-APP $\Gamma; \varepsilon \vdash e_1\ e_2 : \tau$. By inversion, $\Gamma; \varepsilon \vdash e_1 : \tau' \xrightarrow{\varepsilon''} \tau$, $\Gamma; \varepsilon \vdash e_2 : \tau'$, and $\varepsilon'' \leq \varepsilon$ for some $\tau'$ and $\varepsilon''$. So by induction, $\Gamma'; \varepsilon' \vdash e_1 : \tau' \xrightarrow{\varepsilon''} \tau$ and $\Gamma'; \varepsilon' \vdash e_2 : \tau'$. So to us T-APP to derive $\Gamma'; \varepsilon' \vdash e_1\ e_2 : \tau$ we just need $\varepsilon'' \leq \varepsilon'$. This follows from $\varepsilon'' \leq \varepsilon$ and $\varepsilon \leq \varepsilon'$ provided subeffecting is transitive, which we can show by exhaustive case analysis on the small number of effects.
- T-SPAWN-0 $\Gamma; 0 \vdash \mathsf{spawn}_0\ e_1 : \text{int}$. $\Gamma$ is not used so it follows directly that $\Gamma'; 0 \vdash \mathsf{spawn}_0\ e_1 : \text{int}$. Since $0 \leq \varepsilon'$ implies $\varepsilon' = 0$, this suffices.
- T-SPAWN-1 $\Gamma; \varepsilon \vdash \mathsf{spawn}_1\ e_1 : \text{int}$. $\Gamma$ is not used so it follows directly that $\Gamma'; \varepsilon \vdash \mathsf{spawn}_1\ e_1 : \text{int}$. Note the effect is irrelevant.
- T-SPAWN-2 $\Gamma; 2 \vdash \mathsf{spawn}_2\ e_1 : \text{int}$. $\Gamma$ is not used so it follows directly that $\Gamma'; 2 \vdash \mathsf{spawn}_2\ e_1 : \text{int}$. Since $2 \leq \varepsilon'$ implies $\varepsilon' = 2$, this suffices.
- T-ATOMIC This case is very similar to the T-SEQ case. Note the effect is irrelevant.
- T-INATOMIC $\Gamma; \varepsilon \vdash \mathsf{inatomic}(a, e', T_1, T_2) : \tau$. By inversion on the typing rule, it must be that:
  - $\Gamma; 2 \vdash e : \tau$
  - $\Gamma; 0 \vdash T_1$
  - $\Gamma; 2 \vdash T_2$
  - not-active$(T_1)$
  - correct-atomic$(a, e \parallel T_2)$

  The following statements are all true by induction from parts 1 or 2 of the weakening lemma.
  - $\Gamma'; 2 \vdash e : \tau$
  - $\Gamma'; 0 \vdash T_1$
  - $\Gamma'; 2 \vdash T_2$

  Similarly, the following remain true:
  - not-active$(T_1)$
  - correct-atomic$(a, e \parallel T_2)$

  From these facts, the following can be derived: $\Gamma'; \varepsilon' \vdash \mathsf{inatomic}(a, e', T_1, T_2) : \tau$. Note the effect is irrelevant.

2. The proof of part 2 has two cases:
   - If $T = \cdot$ then the rule used to type $T$ was $\Gamma; \varepsilon \vdash \cdot$. This rule requires no assumption so $\Gamma'; \varepsilon' \vdash \cdot$ follows trivially.
   - If $T = T_1 \parallel e$ then the rule used to type $T$ was $\Gamma; \varepsilon \vdash T_1 \parallel e$. By inversion, it must be that $\Gamma; \varepsilon \vdash T_1$ and $\Gamma; \varepsilon \vdash e : \tau$. By induction, it must be that $\Gamma'; \varepsilon' \vdash T_1$ and $\Gamma'; \varepsilon' \vdash e : \tau$. From this, it can be derived that $\Gamma'; \varepsilon' \vdash T_1 \parallel e$.

**Lemma 2.7 (Substitution)** *If* $\Gamma, x : \tau'; \varepsilon \vdash e : \tau$, *and* $\Gamma; \varepsilon' \vdash v : \tau'$, *and* not-active*(e) then:*

1. $\Gamma; \varepsilon \vdash e[v/x] : \tau$, *and*
2. not-active*(*$e[v/x]$*)*

**Proof** By induction on the typing derivation of $e$ and by induction on the derivation of not-active$(e)$, organized by cases on the final rule used in the typing derivation:

- T-VAR.
  1. There are two sub-cases. Either $e = x$ or $e \neq x$.
     - When $e = x$, it must be that $\Gamma, x : \tau'; \varepsilon \vdash x : \tau'$. After substituting, $x[v/x] = v$. Assumption and the values effectless lemma give $\Gamma; \varepsilon \vdash v : \tau'$
     - When $e \neq x$ let $e = y$. By inversion on the typing judgment, it must be that $\Gamma; \varepsilon \vdash y : \tau$. Since $y[v/x] = y$, this is still true.
  2. When $e = x$ then not-active$(v)$ is given by the values inactive lemma. When $e = y \neq x$ then not-active$(y)$ requires no assumptions, so it can always be derived.
- T-CONST $e = c$
  1. In this case, $c[v/x] = c$. Before and after substitution, $e = c$. We can derive $\Gamma; \varepsilon \vdash c : \text{int}$ directly.

2. not-active($c$) requires no assumption, so it can always be derived.

- T-LABEL $e = l$ This case is similar to the T-CONST case.
- T-LAMBDA Where $e = \lambda y.e_1$ and $\tau = \tau_1 \xrightarrow{\varepsilon''} \tau_2$.
  1. By convention, it is acceptable to assume that $x \neq y$. Permutation on the given sub-derivation provides $\Gamma, y : \tau_1, x : \tau'; \varepsilon'' \vdash e_1 : \tau_2$. The weakening lemma and the values effectless lemma provide that $\Gamma, y : \tau_1; \varepsilon \vdash v : \tau'$ By induction, $\Gamma, y : \tau_1; \varepsilon'' \vdash e_1[v/x] : \tau_2$. By inversion, not-active($e_1$), so by induction not-active($e_1[v/x]$). These facts can be used to derive the following via T-LAMBDA: $\Gamma; \varepsilon \vdash \lambda y.e_1[v/x] : \tau$.
  2. By inversion on the typing derivation of $e$, it must be that not-active($e_1$). Thus, not-active($e_1[v/x]$) by induction. This provides a derivation for not-active($\lambda x.e_1[v/x]$).
- T-SEQ Where $e = e_1; e_2$
  1. By inversion on the typing rule, $\Gamma; \varepsilon \vdash e_1 : \tau_1$ and $\Gamma; \varepsilon \vdash e_2 : \tau_2$. By induction, $\Gamma; \varepsilon \vdash e_1[v/x] : \tau_1$ and $\Gamma; \varepsilon \vdash e_2[v/x : \tau_2$. This provides a derivation for $\Gamma; \varepsilon \vdash (e_1; e_2)[v/x] : \tau_2$.
  2. By inversion on the rule for not-active($e_1; e_2$), it must be that not-active($e_1$) and not-active($e_2$). By induction, not-active($e_1[v/x]$) and not-active($e_2[v/x]$). Thus, we can derive not-active($e_1; e_2$).
- T-IF Where $e = \mathsf{if}\ e_1\ e_2\ e_3$
  1. By inversion on the typing rule, $\Gamma; \varepsilon \vdash e_1 : \text{int}$, $\Gamma; \varepsilon \vdash e_2 : \tau$, and $\Gamma; \varepsilon \vdash e_3 : \tau$. By induction, $\Gamma; \varepsilon \vdash e_1[v/x] : \text{int}$ $\Gamma; \varepsilon \vdash e_2[v/x : \tau$, and $\Gamma; \varepsilon \vdash e_3[v/x : \tau$. This provides a derivation for $\Gamma; \varepsilon \vdash (\mathsf{if}\ e_1\ e_2\ e_3)[v/x] : \tau$.
  2. By inversion on the rule for not-active($\mathsf{if}\ e_1\ e_2\ e_3$), it must be that not-active($e_1$), not-active($e_2$), and not-active($e_3$). By induction, not-active($e_1[v/x]$), not-active($e_2[v/x]$), and not-active($e_3[v/x]$). Thus, we can derive not-active($\mathsf{if}\ e_1\ e_2\ e_3$).
- T-SET The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-REF The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-GET The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-APP The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-SPAWN-0 The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-SPAWN-1 The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-SPAWN-2 The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-ATOMIC The proof for this case is by induction, using an argument similar to that for T-SEQ.
- T-INATOMIC It is never the case that not-active($\mathsf{inatomic}(a, e', T_1, T_2)$). Thus, this case is vacuous.

**Lemma 2.8 (Values Effectless)** *If* $\Gamma; \varepsilon \vdash v : \tau$ *then* $\Gamma; \varepsilon' \vdash v : \tau$.

**Proof** To prove the Values Effectless lemma, examine each of the value expressions in StrongNestedParallel, of which there are three: $c, l$, and $\lambda x.e$. The proof is organized by cases on the typing rule for each value expression.

- If the rule used to type $v$ was T-CONST, then $v$ is some $c$. This rule places no restriction on $\varepsilon$, so $\varepsilon$ could be replaced with any $\varepsilon'$.
- If the rule used to type $v$ was T-LABEL, then $v$ is some $l$. This rule places no restriction on $\varepsilon$, so $\varepsilon$ could be replaced with any $\varepsilon'$.
- If the rule used to type $v$ was T-LAMBDA, then $v$ is some $\lambda x.e'$. This rule places no restriction on $\varepsilon$, so $\varepsilon$ could be replaced with any $\varepsilon'$.

**Lemma 2.9 (Canonical Forms)** *If* $\Gamma; \varepsilon \vdash v : \tau$ *and*

- $\tau = \tau_1 \xrightarrow{\varepsilon} \tau_2$ *then* $v$ *is some* $\lambda x.e$
- $\tau = \text{ref}_t \tau'$ *then* $v$ *is some* $l$
- $\tau = \text{int}$ *then* $v$ *is some* $c$

**Proof** Follows from inspection of the typing rules for $v$

**Lemma 2.10 (Variables not in $\Gamma'$)** *If* $\Gamma \vdash H : \Gamma'$ *then* $x \notin Dom(\Gamma')$.

**Proof** By induction on the typing derivation of $H$: If $H = \cdot$ then $\Gamma' = \cdot$ and $x \notin Dom(\cdot)$. If $H$ is non-empty then $\Gamma \vdash H, l \mapsto v : \Gamma'', l{:}(\tau, t)$. By inversion, $\Gamma \vdash H : \Gamma''$. By induction, $x \notin Dom(\Gamma'')$ and $x$ is not added by the conclusion of the typing rule.

**Lemma 2.11 (Values Inactive)** *If* $\Gamma; \varepsilon \vdash v : \tau$ *then* not-active*(v).*

**Proof** This language has three value expressions, $c, l$, and $\lambda x.e$. Proof is by cases on the typing rule for each value expression.

- If the rule used to type $v$ was T-CONST, then $v$ is some $c$. Deriving not-active($c$) requires no assumption.
- If the rule used to type $v$ was T-LABEL, then $v$ is some $l$. Deriving not-active($l$) requires to assumption.
- If the rule used to type $v$ was T-LAMBDA, then $v$ is some $\lambda x.e'$. By inversion on the typing rule, it must be that not-active($e'$). Thus, it is possible to derive not-active($\lambda x.e'$).

**Lemma 2.12 (Effects Lemma)** *Suppose* $\Gamma; \varepsilon \vdash e : \tau$ *and* $a; H; e \to a'; H'; e'; T_0; T_1; T_2$ *then the following must hold:*

1. *If* $\varepsilon \neq 0$*, then* $T_0 = \cdot$.
2. *If* $\varepsilon \neq 2$*, then* $\implies T_2 = \cdot$.

**Proof** by induction on the typing derivation of $e$ by cases on the final rule used in the derivation. If the final rule was:

- T-VAR $e = x$. There are no evaluation rules for $x$, and thus $e$ cannot step. This case is vacuously true.
- T-CONST $e = c$. $e$ is a value, and thus cannot step. This case is vacuously true.
- T-LABEL $e = l$. $e$ is a value, and thus cannot step. This case is vacuously true.
- T-LAMBDA $e = \lambda x.e_1$. $e$ is a value, and thus cannot step. This case is vacuously true.
- T-SEQ $e = e_1; e_2$. There are two possible situations:
  1. $e_1$ is not a value. Thus, the evaluation rule SEQ-1 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
  2. $e_1$ is a value. The evaluation rule SEQ-V applies, so $T_0 = T_1 = T_2 = \cdot$.
- T-IF $e = \mathsf{if}\ e_1\ e_2\ e_3$. There are two possible situations:
  1. $e_1$ is not a value. Thus, the evaluation rule IF-1 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
  2. $e_1$ is a value. Depending on the value, the evaluation rules IF-Z or IF-NZ apply, so $T_0 = T_1 = T_2 = \cdot$.
- T-SET $e = e_1 := e_2$. There are three possible situations:
  1. $e_1$ is not a value. Thus, the evaluation rule SET-1 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
  2. $e_1$ is some $l$ and $e_2$ is not a value. The evaluation rule SET-2 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
  3. $e_1$ is some $l$ and $e_2$ is some $v$. The evaluation rule STRONG-SET applies, which always creates $T_0 = T_1 = T_2 = \cdot$.
- T-REF If $e = \text{ref}\ e'$ There are two possibilities:
  1. $e'$ is not a value. The evaluation rule REF-1 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.

   2. $e'$ is some $v$. The evaluation rule ALLOC applies, and creates $T_0 = T_1 = T_2 = \cdot$.

- T-GET $e = !e_1$ has two sub-cases:
   1. $e_1$ is not a value. The evaluation rule GET-1 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
   2. $e_1$ is some $v$. The evaluation rule STRONG-GET applies, and creates $T_0 = T_1 = T_2 = \cdot$.
- T-APP $e = e_1\ e_2$. There are three possible sub-cases:
   1. $e_1$ is not a value. Thus, the evaluation rule APP-1 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
   2. $e_1$ is some $v$ and $e_2$ is not a value. Thus, the evaluation rule APP-2 applies. By induction, if $\varepsilon \neq 0$ then $T_0 = \cdot$ and if $\varepsilon \neq 2$ then $T_2 = \cdot$.
   3. $e_1$ is some $\lambda x.e_3$ and $e_2$ is some $v$. In this case, BETA applies, which creates $T_0 = T_1 = T_2 = \cdot$.
- T-SPAWN-0 $e = \mathsf{spawn}_0\ e_1$. It must be that $\varepsilon = 0$. It is sufficient to show that $T_2 = \cdot$. The only evaluation rule that applies is SPAWN 0, which always creates $T_2 = \cdot$.
- T-SPAWN-1 $e = \mathsf{spawn}_1\ e_1$. The only evaluation rule that applies is SPAWN 1, which always creates $T_0 = T_2 = \cdot$.
- T-SPAWN-2 $e = \mathsf{spawn}_2\ e_1$. It must be that $\varepsilon = 2$. It is sufficient to show that $T_0 = \cdot$. The only evaluation rule that applies is SPAWN 2, which always creates $T_0 = \cdot$.
- T-ATOMIC $e = \mathsf{atomic}\ e_1$. The only evaluation rule that applies in this case is ENTER ATOMIC, which creates $T_0 = T_1 = T_2 = \cdot$.
- T-INATOMIC $e = \mathsf{inatomic}(a, e', T_1, T_2)$. There are three possible sub-cases:
   1. Assume $e'$ is not a value, and the chosen evaluation rule was INATOMIC. INATOMIC always creates $T_0 = T_1 = T_2 = \cdot$.
   2. Assume $T_2$ contains at least one expression that is not a value, and the chosen evaluation rule was INATOMIC HELPER. INATOMIC HELPER always creates $T_0 = T_1 = T_2 = \cdot$.
   3. $e'$ is some $v$ and $T_2 = \overline{v}$. The evaluation rule that applies here is EXIT ATOMIC which creates $T_0 = T_2 = \cdot$.

## 2.7 Equivalence

The StrongNestedParallel language is trivially equivalent to itself.

# 3 The Weak language

This section describes a new language that is equivalent to the StrongNestedParallel language when given a partition on the heap. The equivalence result in our POPL'08 submission (between Weak and StrongNestedParallel) is given in Section 3.7.

## 3.1 Syntax

No Changes.

## 3.2 Dynamic Evaluation

### 3.2.1 Whole Program Evaluation

$\boxed{a; H; T \rightarrow\ a'; H'; T'}$

No Changes.

### 3.2.2 Expression Evaluation

The StrongNestedParallel language enforces *strong atomicity* because the two evaluation rules that access the heap (STRONG-SET and STRONG-GET) require that $a = \circ$. Because the expression evaluation form is unchanged, we simply *weaken* the semantics of this language by relaxing the restrictions on $a$. As a result, we replace the STRONG-SET and the STRONG-GET rules from StrongNestedParallel with the WEAK-SET and WEAK-GET rules below:

$\boxed{a; H; e \rightarrow\ a'; H'; e'; T_1; T_2; T_3}$

$$\text{WEAK-SET}\quad \frac{}{a; H; l := v \rightarrow\ a; H, l \mapsto v; v; \cdot; \cdot; \cdot} \qquad \text{WEAK-GET}\quad \frac{}{a; H; !l \rightarrow\ a; H; H(l); \cdot; \cdot; \cdot}$$

## 3.3 Typecheck $e$

**Definition 3.1 (Heap Partition)** *A heap $H$ is partitioned when each label (after being initialized) is read and written* always *in a transaction or* never *in a transaction.*

We present a type and effect system to partition the heap into two parts; data that is *always* read or written in a transaction and data that is *never* read or written in a transaction. To obtain this partition, we use the extra information stored in $\Gamma$ to keep track of and ensure appropriate use of each label. Until now, it was not used in StrongNestedParallel, although our equivalence result requires that programs in Weak and StrongNestedParallel both typecheck using the partition rules.

The following changes to the type system for the StrongNestedParallel language (Section 2.3) achieve this goal:

1. Remove typing rules T-SPAWN-1 and T-SPAWN-2 and require that the last two components of any inatomic expression are $\cdot$. Assuming well-typedness, this is equivalent to removing the corresponding spawn flavors from the language since programs with them would never type-check. This approach simplifies our definition of a partition and the type system that enforces it.
2. Replace T-SET, T-GET, T-REF, and T-LABEL with rules that enforce a partition: T-SET-PARTITION, T-GET-PARTITION, T-REF-PARTITION, and T-LABEL-PARTITION, listed below. Note the new restrictions on $t$ when compared to the partner rules in Section 2.3.

$$\text{T-SET-PARTITION}\quad \frac{\Gamma; t \vdash e_1 : \text{ref}_t\tau \qquad \Gamma; t \vdash e_2 : \tau}{\Gamma; t \vdash e_1 := e_2 : \tau} \qquad \text{T-GET-PARTITION}\quad \frac{\Gamma; t \vdash e : \text{ref}_t\tau}{\Gamma; t \vdash !e : \tau} \qquad \text{T-REF-PARTITION}\quad \frac{\Gamma; \varepsilon \vdash e : \tau}{\Gamma; \varepsilon \vdash \text{ref } e : \text{ref}_t\tau} \qquad \text{T-LABEL-PARTITION}\quad \frac{\Gamma(l) = (\tau, t)}{\Gamma; \varepsilon \vdash l : \text{ref}_t\tau}$$

Notice that T-REF-PARTITION allows programs to create a new label in one part of the partition and insist that it always be accessed in a different part. This does not introduce data races because when a label is created it must be thread local.

## 3.4 Other Typing Rules

No Changes.

## 3.5 Activeness

No Changes.

## 3.6 Type Safety

| Theorem/Lemma | Restated | Changes |
|---|---|---|
| Top-Level Progress | No | None. |
| Single Thread Progress | No | T-SPAWN-1: Remove this case<br>T-SPAWN-2: Remove this case |
| Top-Level Preservation | No | None. |
| Single Thread Preservation | | The T-LABEL case is still vacuous.<br>The T-SET and T-GET cases change to say $a$ does not change by the primitive step (rather than assuming $a$ remains $\circ$; it may now remain $\bullet$). |
| Weakening Lemma | No | T-SPAWN-0 and T-SPAWN-2: The effect becomes irrelevant.<br>T-GET and T-SET: $\epsilon$ is either 0 or 2 so $\epsilon' = \epsilon$. |
| Substitution Lemma | No | None. |
| Values Effectless Lemma | No | Nothing significant; notice that a label may occur anywhere. Only reading or writing its contents is restricted. |
| Variables not in $\Gamma'$ | No | None. |
| Values Inactive | No | None. |
| Effects Lemma | Remove Part 2 | The second part becomes trivial without T-SPAWN-2. |

### 3.6.1 A Note About Progress

Strictly speaking, the type-safety result from Section 2.6 does not apply to the weak-atomicity semantics and type system and we should repeat the proof. Type-safety for each language is important because the equivalence proof in Section 3.7 will assume that if a well-typed initial configuration takes $n$ steps then the resulting configuration is well-typed.

However, the proofs of Progress require no significant changes since it is a strictly easier result to show. The type system for Weak is strictly stronger than the one for StrongNestedParallel, and Weak's operational semantics is strictly more lenient than the semantics for StrongNestedParallel. The type system accepts a subset of the configurations accepted by StrongNestedParallel, and the Weak semantics allows a superset of the transitions.

## 3.7 Equivalence

In this section, we show that the StrongNestedParallel and Weak languages are equivalent under a heap partition. Because the syntax of both languages is the same, we distinguish the weak semantic rules and the strong semantics rules, using $\rightarrow_w$ to mean evaluation under the weak operational rules and $\rightarrow_s$ to mean evaluation under the strong operational rules. The difference between these semantics is given in Section 3.2. Any typing judgments, such as $\Gamma \vdash H : \Gamma$ and $\Gamma; \varepsilon \vdash e : \tau$ refer to the partitioned type system given in Section 3.3. Even though StrongNestedParallel was defined under a slightly different type system, we limit its syntax and modify the typing rules in the intuitive manner.

**Lemma 3.2 (Top-level Reordering)** *Suppose all the following hold:*

1. $\Gamma \vdash H_0 : \Gamma$
2. $\Gamma; 0 \vdash e_A : \tau_A$
3. $\Gamma; 0 \vdash e_B : \tau_B$
4. $\text{active}(e_A)$
5. $\text{not-active}(e_B)$
6. $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_s \bullet; H_1; e'_A \parallel e_B \parallel T \rightarrow_w \bullet; H_2; e'_A \parallel e'_B \parallel T'$

*Then there exists some* $H'_1$ *such that* $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_w \bullet; H'_1; e_A \parallel e'_B \parallel T' \rightarrow_s \bullet; H_2; e'_A \parallel e'_B \parallel T'$.

**Top-level Reordering** Proof by induction on the form of $e_A$, organized by cases on $e_A$.

- $e_A = v$. By the values inactive lemma, $e_A$ cannot be active. This case is vacuous.
- $e_A = x$. $x$ has no viable steps, so this case is vacuous.
- $e_A = \mathsf{spawn}_i\ e, 0 \leq i < 2$. There is no way to derive that $e_A$ is active. This case is vacuous.
- $e_A = \mathsf{atomic}\ e$. There is no way to derive that $e_A$ is active. This case is vacuous.
- $e_A = e_1; e_2$. There is only one way to derive active($e_A$). This is when active($e_1$). In this case, it must be that $e_1$ is not a value, and it must be that $e_1$ stepped. $H'_1$ is obtained through induction.
- $e_A = \mathsf{if}\ e_1\ e_2\ e_3$. This case is similar to that when $e_A = e_1; e_2$.
- $e_A = e_1 := e_2$. There are two ways to derive active($e_A$), either $e_1$ is not a value and active($e_1$) or $e_1$ is some $l$ and active($e_2$).
  1. active($e_1$). Here $e_1$ took a step, and $H'_1$ is obtained through induction.
  2. active($e_2$) and $e_1 = l$. Here $e_2$ took a step, and $H'_1$ is obtained through induction.
- $e_A = \text{ref}\ e_1$. This case is similar to that when $e_A = e_1; e_2$.
- $e_A = !e_1$. This case is similar to that when $e_A = e_1; e_2$.
- $e_A = e_1\ e_2$. This case is similar to that when $e_A = e_1 := e_2$.
- $e_A = \mathsf{inatomic}(a'', e_1, \cdot, \cdot)$. Either $e_1$ is a value or it is not.
  1. If $e_1$ is not a value, then $e_1$ it must be the expression that steps. To apply Lemma 3.3, we need to show that each precondition applies:
     (a) $\Gamma \vdash H_0 : \Gamma$ by assumption.
     (b) By inversion on the typing rule for $e_A$, we know $\Gamma; 2 \vdash e_1 : \tau_A$.
     (c) $\Gamma; 0 \vdash e_B : \tau_B$ by assumption
     (d) not-active($e_B$) by assumption
     (e) Since $e_A$ must step under the INATOMIC rule, we have $a''; H_0; e_1 \rightarrow_s a'''; H_1; e'_1; \cdot; \cdot; \cdot$ to evaluate $e_A$, (so $\bullet; H_0; e_A \rightarrow_s \bullet; H_1; e'_A; \cdot; \cdot; \cdot$) and $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_0; \cdot; \cdot$ is true by assumption.

     Apply Lemma 3.3 (with $e_1$ and $e_B$). to obtain an appropriate $H'_1$.
  2. $e_1 = v$. The only evaluation rule that applies to $e_A$ is EXIT ATOMIC, which changes $a$ from $\bullet$ to $\circ$. Because we require that $a = a' = \bullet$, this case is vacuous.

**Lemma 3.3 (Nested Reordering)** *Suppose all the following hold:*

1. $\Gamma \vdash H_0 : \Gamma$
2. $\Gamma; 2 \vdash e_A : \tau_A$
3. $\Gamma; 0 \vdash e_B : \tau_B$
4. $\text{not-active}(e_B)$
5. $a; H_0; e_A \rightarrow_s a'; H_1; e'_A; \cdot; \cdot; \cdot$ *and* $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_0; \cdot; \cdot$

*Then there exists some* $H'_1$ *such that* $\bullet; H_0; e_B \rightarrow_w \bullet; H'_1; e'_B; T_0; \cdot; \cdot$ *and* $a; H'_1; e_A \rightarrow_s a'; H_2; e'_A; \cdot; \cdot; \cdot$

**Nested Reordering** Proof by induction on the evaluation of $e_A$; organized by cases on the form of $e_A$. (Note the interesting cases use the next two lemmas.)

- $e_A = v$. $v$ has no viable steps, so this case is vacuous.
- $e_A = x$. $x$ has no viable steps, so this case is vacuous.
- $e_A = \text{ref } e''_A$ There are two cases:
  - $e''_A$ is not a value, so $e''_A$ steps. $H'_1$ is obtained through induction.
  - $e''_A = v_A$. In this case, $e_A$ steps in the following manner: $a; H_0; \text{ref } e''_A \rightarrow_s a'; H_0, l_{new} \mapsto v_A; l_{new}; \cdot; \cdot; \cdot$ As a result, $H_1 = H_0, l_{new} \mapsto v_A$, and $l_{new} \notin \mathit{Dom}(H_0)$. Since $\Gamma \vdash H_0 : \Gamma$ we know that $l_{new} \notin \mathit{Dom}(\Gamma)$. By the assumption that $e_B$ is well-typed (i.e. $\Gamma; 0 \vdash e_B : \tau_B$), it must be $e_B$ cannot refer to $l_{new}$ and type-check. In this case, choose $H'_1 = H_2 \setminus \{l_{new} \mapsto v_A\}$.
- $e_A = \mathsf{spawn}_0\ e$. This is impossible because $\Gamma; 2 \vdash e_A : \tau_A$ prohibits it.
- $e_A = \mathsf{spawn}_1\ e$. This expression won't type-check under the available rules, so this case is vacuous.
- $e_A = \mathsf{spawn}_2\ e$. This expression won't type-check under the available rules, so this case is vacuous.
- $e_A = \mathsf{atomic}\ e$. In this case, $e_A$ steps under the ENTER ATOMIC rule. Using this rule, we know that $H_1 = H_0$, and therefore $e_B$ effectively steps under $H_0$ even before reordering. If $e_B$ creates some different heap ($H_2$), then $e_B$ was already doing this from $H_0$, so pick $H'_1 = H_2$. The other evaluation results for $e_B$ (i.e. $T_0$, $e'_B$ )are also preserved. Similarly, we know that $e_A$ doesn't read or write any $H$ when stepping using the INATOMIC rule, so evaluating $e_A$ is also unaffected by reordering.
- $e_A = e_1; e_2$. There are two cases:
  - $e_1$ is not a value. $H'_1$ is obtained through induction.
  - $e_1 = v$ then $e_A$ becomes $e_2$ in one step, with no heap reads or writes. Choosing $H'_1 = H_2$ is acceptable by an argument similar to that for the $\mathsf{atomic}\ e$ case.
- $e_A = \mathsf{if}\ e_1\ e_2\ e_3$. This case is similar to that when $e_A = e_1; e_2$.
- $e_A = e_1 := e_2$. There are 3 cases:
  - $e_1$ is not a value. $H'_1$ is obtained through induction.
  - $e_1 = l_A$ and $e_2$ is not a value. $H'_1$ is obtained through induction.
  - $e_1 = l_A$ and $e_2 = v_A$ ($e_A$ steps using STRONG-SET) The following are true:
    1. By assumption, $\Gamma \vdash H_0 : \Gamma$
    2. By the STRONG-SET rule, it must be that $H_1 = H_0, l_A \mapsto v_A$.
    3. $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_0; \cdot; \cdot$ by assumption.
    4. By assumption, $\Gamma; 2 \vdash l_A := v_A : \tau_A$.
    5. By assumption, $\Gamma; 0 \vdash e_B : \tau_B$
    6. By assumption, not-active($e_B$)

    Apply Lemma 3.4 to obtain $H'_1$.
- $e_A = !e_1$. There are two cases:
  - $e_1$ is not a value. $H'_1$ is obtained through induction.
  - $e_1 = l_A$ ($e_A$ steps using STRONG-GET). All of the following are true:
    1. By assumption, $\Gamma \vdash H_0 : \Gamma$
    2. By the STRONG-GET rule for reading the heap, it must be that $H_0(l_A) = v_A$ for some $v_A$. Also, $H_1 = H_0$, so $H_1(l_A) = H_0(l_A) = v_A$.
    3. $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_0; \cdot; \cdot$ by assumption.
    4. By assumption, $\Gamma; 2 \vdash !l_A : \tau_A$.
    5. By assumption, $\Gamma; 0 \vdash e_B : \tau_B$
    6. By assumption, not-active($e_B$)

    Apply Lemma 3.5 to obtain $H'_1$.
- $e_A = e_1\ e_2$. There are three cases:
  - If $e_1$ is not a value, then $H'_1$ is obtained through induction.

  - If $e_1 = \lambda x.e_3$ and $e_2$ is not a value, then $H_1'$ is obtained through induction.
  - If $e_1 = \lambda x.e_3$ and $e_2 = v$ then $e_A$ steps under the BETA rule and $e_A' = e_3[v/x]$, with $H_1 = H_0$. The argument here is similar to that for the $\mathsf{atomic}\ e$ case.

- $e_A = \mathsf{inatomic}(a, e_A'', \cdot, \cdot)$. There are two cases:
  - $e_A''$ is not a value. In this case, $H_1'$ is obtained through induction.
  - $e_A'' = v_A$. Here, $e_A$ exits the transaction via EXIT ATOMIC and gives $H_1 = H_0$. The argument here is similar to that for the $\mathsf{atomic}\ e$ case.

**Lemma 3.4 (Independent Write Under Partition)** *Let $e_A = l_A := v_A$, so under the strong semantics $a; H_0; l_A := v_A \rightarrow_s a'; H_1; v_A; \cdot; \cdot; \cdot$.*

1. $\Gamma \vdash H_0 : \Gamma$
2. $H_1 = H_0, l_A \mapsto v_A$
3. $\bullet; H_0, l_A \mapsto v_A; e_B \rightarrow_w \bullet; H_2; e_B'; T_0; \cdot; \cdot$
4. $\Gamma; 2 \vdash e_A : \tau_A$ *(and by inverting on* T-SET-PARTITION *and* T-LABEL-PARTITION *we know that* $\Gamma(l_A) = (\tau_A, 2)$*)*
5. $\Gamma; 0 \vdash e_B : \tau_B$
6. not-active*(*$e_B$*)*

*then*

1. $H_2(l_A) = v_A$ *(i.e. result prior to reorder is preserved for* $e_A$*)*
2. $\bullet; H_0; e_B \rightarrow_w \bullet; H_1'; e_B'; T_0; \cdot; \cdot$ *(i.e.* $e_B$ *becomes the same* $e_B'$ *as before with the same* $T_0$*, and* $H_1'$ *is like* $H_2$ *except that* $H_1'(l_A) = H_0(l_A)$*)*

**Proof** Independent Write By induction on the evaluation of $e_B$, organized by cases on $e_B$.

- $e_B = v_B$. $e_B$ does not step, so this case is vacuous.
- $e_B = x$. $e_B$ does not step, so this case is vacuous.
- $e_B = \mathsf{spawn}_0\ e_B''$. The only step available to $e_B$ is $\bullet; H_0; \mathsf{spawn}_0\ e_B'' \rightarrow_w \bullet; H_0; 0; e_B''; \cdot; \cdot$. From this, it is clear that before reordering the heaps for evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$ (and won't change $H_0$'s mapping either) so $H_2(l_A) = v_A$ as before. Similarly, $e_B$ is unaffected by the values in $H_0$ and so $e_B' = 0$ and $T_0 = e_B''$ are as before.
- $e_B = \mathsf{spawn}_1\ e_B''$. This expression cannot typecheck in the current language, so this case is vacuous.
- $e_B = \mathsf{spawn}_2\ e_B''$. This expression cannot typecheck in the current language, so this case is vacuous.
- $e_B = \mathsf{atomic}\ e$. $e_B$ cannot step under $\bullet$. This case is vacuous.
- $e_B = e_1; e_2$. There are two cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = v_B$ and $e_2$ is not a value. Here, $e_B$ steps using the SEQ-V rule, so $\bullet; H_0; v; e_2 \rightarrow_w \bullet; H_0; e_2; \cdot; \cdot; \cdot$. Notice that $H_1' = H_0$ and so $e_A$ executes under $H_1'$ to get $H_2$ as before. Thus, $H_2(l_A) = v_A$), and $e_B' = e_2$ with $T_0 = \cdot$ as before.
- $e_B = \mathsf{if}\ e_1\ e_2\ e_3$. This case is similar to that when $e_B = e_1; e_2$.
- $e_B = e_1 := e_2$. There are three cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = l_B$ and $e_2$ is not a value. By induction.
  - $e_1 = l_B$ and $e_2 = v_B$. By assumption, we have $\Gamma; 0 \vdash e_B : \tau_B$, and since $e_1$ is some $l_B$, we can invert on the typing rules to get: $\Gamma(l_B) = (\tau_B, 0)$. We know that $\Gamma(l_A) = (\tau_A, 2)$, therefore $l_A$ and $l_B$ must be distinct. We know from before reordering that $H_2 = H_0, l_A \mapsto v_A, l_B \mapsto v_B$, so we pick $H_1' = H_0, l_B \mapsto v_B$ and after evaluating $e_A$ we have $H_2 = H_0, l_B \mapsto v_B, l_A \mapsto v_A$. As a result, $H_2(l_A) = v_A$. Additionally, $e_B' = v_B$ and $T_0 = \cdot$ as before.
- $e_B =!e_1$. There are two cases:

  - $e_1$ is not a value. By induction.
  - $e_1 = l_B$ By assumption, we have $\Gamma; 0 \vdash e_B : \tau_B$, and since $e_1$ is some $l_B$, we can invert on the typing rules to get: $\Gamma(l_B) = (\tau_B, 0)$. It is given that $\Gamma(l_A) = (\tau_A, 2)$, therefore $l_A$ and $l_B$ must be distinct. By assumption, $H_1 = H_0, l_A \mapsto v_A$. Because the labels are distinct, we also know that $H_0(l_B) = v_B$ (and therefore $e'_B = v_B = H_0(l_B)$ are as before). By picking $H'_1 = H_0$, $e_A$ evaluates under $H_0$ as it did before, giving $H_2 = H_0, l_A \mapsto v_A$ with $H_2(l_A) = v_A$.

- $e_B = e_1\ e_2$. There are three cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = \lambda x.e''_B$ and $e_2$ is not a value. By induction.
  - $e_1 = \lambda x.e''_B$ and $e_2 = v_B$. The only step available to $e_B$ is $\bullet; H_0; \lambda x.e''_B\ v_B \rightarrow\ \bullet; H_0; e''_B[v_B/x]; \cdot; \cdot; \cdot$. From this, it is clear that before reordering the heaps for evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$, so $H_2(l_A) = v_A$ as before. Similarly, $e_B$ does not use any values in $H$ and so $e'_B$ and $T_0$ are as before.
- $e_B = \mathsf{inatomic}(a, e''_B, \cdot, \cdot)$. By the assumption that not-active$(e_B)$, this case is vacuous.
- $e_B = \text{ref } e''_B$. There are two cases:
  - $e''_B$ is not a value. By induction.
  - $e''_B$ is some $v$ In this case, a label is created and $H_2 = H_0, l_A \mapsto v_A, l_{new} \mapsto v$ and $e'_B = l_{new}$. We know that $l_{new} \notin Dom(H_0)$, and since $\Gamma \vdash H_0 : \Gamma$ it must be that $l_{new} \notin Dom(\Gamma)$. Since we also assume that $\Gamma(l_A) = (\tau_A, 2)$, it must be that that $l_A \neq l_{new}$ (otherwise, $e_A$ could not type-check under $\Gamma$). As a result, $e_B$ can step under $H_0$ in the following way: $\bullet; H_0; e_B \rightarrow_w\ \bullet; H_0, l_{new} \mapsto v; l_{new}; \cdot; \cdot; \cdot$, giving $e'_B = l_{new}$ and $T_0 = \cdot$ as before. By picking $H'_1 = H_0, l_{new} \mapsto v$, we have $H'_1(l_A) = H_0(l_A)$, and after the write to $l_A$ we have $H_2 = H_0, l_{new} \mapsto v, l_A \mapsto v_A$ with $H_2(l_A) = v_A$.

**Lemma 3.5 (Independent Read Under Partition)** *Let $e_A = !l_A$, so under the strong semantics $a; H_0; !l_A \rightarrow_s\ a'; H_0; H_0(l_A); \cdot; \cdot; \cdot$*

1. $\Gamma \vdash H_0 : \Gamma$
2. $H_1 = H_0$, *and* $H_0(l_A) = v_A$
3. $\bullet; H_1; e_B \rightarrow_w\ \bullet; H_2; e'_B; T_0; \cdot; \cdot$
4. $\Gamma; 2 \vdash e_A : \tau_A$ *(and by inverting on* T-GET-PARTITION *and* T-LABEL-PARTITION *we know that* $\Gamma(l_A) = (\tau_A, 2)$*).*
5. $\Gamma; 0 \vdash e_B : \tau_B$
6. not-active*($e_B$)*

*then*

1. $H_2(l_A) = v_A$ *(i.e. $e_A$ reads the same value from $H_2$ as it did from $H_0$)*
2. $\bullet; H_0; e_B \rightarrow_w\ \bullet; H_2; e'_B; T_0; \cdot; \cdot$ *(i.e. $e_B$ evaluates to the same $T_0$ and $e'_B$ under $H_0$ as it did under $H_1$. This is relatively obvious since we have picked $H_1 = H_0$).*

**Proof** Independent Read by induction on the evaluation of $e_B$, organized by cases on $e_B$.

- $e_B = v_B$. $e_B$ does not step, so this case is vacuous.
- $e_B = x$. $e_B$ does not step, so this case is vacuous.
- $e_B = \mathsf{spawn}_0\ e''_B$. The only step available to $e_B$ is $\bullet; H_0; \mathsf{spawn}_0\ e''_B \rightarrow_w\ \bullet; H_0; 0; e''_B; \cdot; \cdot$. Because $H_0 = H_1 = H_2$, it is clear that before reordering the evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$, so $H_2(l_A) = v_A$. Similarly, $e_B$ is unaffected by the heap and so $e'_B$ and $T_0$ are as before.
- $e_B = \mathsf{spawn}_1\ e''_B$. This case is vacuous because it violates the assumption that $\Gamma; 0 \vdash e_B : \tau_B$.
- $e_B = \mathsf{spawn}_2\ e''_B$. This case is vacuous because it violates the assumption that $\Gamma; 0 \vdash e_B : \tau_B$.
- $e_B = \mathsf{atomic}\ e$. This case is vacuous because $e_B$ cannot step under $\bullet$.
- $e_B = e_1; e_2$. There are two cases:

  - $e_1$ is not a value. By induction.
  - $e_1 = v_B$ and $e_2$ is not a value. Here, $\bullet; H_0; v; e_2 \rightarrow_w \bullet; H_0; e_2; \cdot; \cdot; \cdot$. Because we have picked $H_1' = H_0 = H_1$, $e_B$ can evaluate under $H_0$ as it did under $H_1$. As a result, $e_B' = e_2$ and $T_0 = \cdot$ as before. Also, $e_B$ steps using SEQ-V and thus $H_2 = H_1$ was true before reordering. The choice of $H_1' = H_2 = H_1 = H_0$ gives that $H_2(l_A) = v_A$.

- $e_B = \mathsf{if}\ e_1\ e_2\ e_3$. This case is similar to that when $e_B = e_1; e_2$.
- $e_B = e_1 := e_2$. There are three cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = l_B$ and $e_2$ is not a value. By induction.
  - $e_1 = l_B$ and $e_2 = v_B$. By assumption, we know that $\Gamma; 0 \vdash e_B : \tau_B$, and since $e_B = l_B$, inversion on the typing derivation for $e_B$ gives that $\Gamma(l_B) = (\tau_B, 0)$. It is given that $\Gamma(l_A) = (\tau_A, 2)$, therefore $l_A$ and $l_B$ are distinct. Thus, $H_2 = H_0, l_B \mapsto v_B$. By picking $H_1' = H_2$ and noting that $H_1 = H_0$, $H_2(l_A) = v_A$, with $e_B'$ and $T_0 = \cdot$ as before.
- $e_B = !e_1$. There are two cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = l_B$ By assumption, we know that $\Gamma; 0 \vdash e_B : \tau_B$, and since $e_B = !l_B$, inversion on the typing derivation for $e_B$ gives that $\Gamma(l_B) = (\tau_B, 0)$. It is given that $\Gamma(l_A) = (\tau_A, 2)$, therefore $l_A$ and $l_B$ are distinct. By picking $H_1' = H_0$ where $H_0(l_B) = v_B$, and $H_2 = H_0 = H_1$, we can show that $H_2(l_A) = v_A$, and $e_B' = v_B = H_0(l_B)$ and $T_0 = \cdot$ are as before.
- $e_B = e_1\ e_2$. There are three cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = \lambda x.e_B''$ and $e_2$ is not a value. By induction.
  - $e_1 = \lambda x.e_B''$ and $e_2 = v_B$. The only step available to $e_B$ is $\bullet; H_0; \lambda x.eB''\ v_B \rightarrow_w \bullet; H_0; e_B''[v_B/x]; \cdot; \cdot; \cdot$. From this, it is clear that $H_0 = H_2$, so $H_2(l_A) = H_0(l_A) = v_A$ Similarly, $e_B'$ is unaffected by the values on the heap and so $e_B'$ and $T_0$ are as before.
- $e_B = \mathsf{inatomic}(a, e_B'', \cdot, \cdot)$. By the assumption that not-active($e_B$), this case is vacuous.
- $e_B = \text{ref}\ e_B''$. There are two cases:
  - $e_B''$ is not a value. By induction.
  - $e_B''$ is some $v$ In this case, a label is created and $H_1' = H_0, l_{new} \mapsto v$ and $e_B' = l_{new}$. We know that $l_{new} \notin Dom(H_0)$, and since $\Gamma \vdash H_0 : \Gamma$ it must be that $l_{new} \notin Dom(\Gamma)$. Since we also assume that $\Gamma(l_A) = (\tau_A, 2)$, it must be that that $l_A \neq l_{new}$ (otherwise, $e_A$ could not type-check under $\Gamma$). As a result, $e_B$ can step under $H_0$ in the following way: $\bullet; H_0; e_B \rightarrow_w \bullet; H_0, l_{new} \mapsto v; l_{new}; \cdot; \cdot; \cdot$, giving $e_B' = l_{new}$ and $T_0 = \cdot$ as before. Similarly, $H_1'(l_A) = H_0(l_A)$, so $H_2(l_A) = v_A$.

**Lemma 3.6 (Topmost Enter-Atomic)** *Suppose*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; 0 \vdash e_B : \tau'$
3. $\Gamma; 0 \vdash e_A : \tau$
4. not-active*($e_A$)*
5. not-active*($e_B$)*
6. $\circ; H; e_A \parallel e_B \parallel T \rightarrow_s \bullet; H; e_A' \parallel e_B \parallel T$ *and*
   $\bullet; H; e_A' \parallel e_B \parallel T \rightarrow_w \bullet; H'; e_A' \parallel e_B' \parallel T'$.

*Then the following must be true:*

1. active*($e_A'$)*
2. $\circ; H; e_A \parallel e_B \parallel T \rightarrow_w \circ; H'; e_A \parallel e_B' \parallel T$ *and*
   $\circ; H'; e_A \parallel e_B' \parallel T' \rightarrow_s \bullet; H'; e_A' \parallel e_B' \parallel T'$

**Proof of Lemma 3.6** Proof by cases on $e_A$.

- $e_A = v$. $v$ cannot step. This case is vacuous.
- $e_A = x$. $x$ cannot step. This case is vacuous.
- $e_A = \text{ref } e''_A$ There are two cases:
  - If $e''_A$ is not a value, this is true by induction.
  - If $e''_A$ is some $v$, then $e_A$ steps using ALLOC under which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = \mathsf{spawn}_0\ e$. The only step for $e_A$ is SPAWN-0, under which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = \mathsf{spawn}_1\ e$. This expression is not in the current language, so this case is vacuous.
- $e_A = \mathsf{spawn}_2\ e$. This expression is not in the current language, so this case is vacuous.
- $e_A = \mathsf{atomic}\ e_0$. Here, only one evaluation step applies, ENTER-ATOMIC. Thus, $\circ; H; e_A \rightarrow_s \bullet; H; \mathsf{inatomic}(\circ, e_0, \cdot, \cdot)$ Deriving active$(e'_A)$ is obvious. Showing that $e_B$ can be moved to before $e_A$ entered a transaction requires cases on $e_B$. For each case, we must show: that $e_B$ is unaffected by $e_A$'s evaluation in terms of $H$ and $T$ and $a$. Notice that by assumption, $\bullet; H; T \parallel e_B \rightarrow_w \bullet; H'; T' \parallel e'_B$. Or rather, that $e_B$ stepped using the weak rules under $\bullet$. Also by assumption, $e_A$ does not change $H$ or $T$.
  - $e_B = v$. $v$ cannot step. This case is vacuous.
  - $e_B = x$. $x$ cannot step. This case is vacuous.
  - $e_B = \text{ref } e''_B$. There are two cases:
    * If $e''_B$ is not a value, this is true by induction.
    * If $e''_B$ is some $v$, then the step for $e_B$ is to create a new, thread-local label, $l_{new}$, for $v$ using the ALLOC rule. By assumption, $l_{new} \notin Dom(H)$ and since $\Gamma \vdash H : \Gamma$, we know that $l_{new} \notin Dom(\Gamma)$. As a result, $e_A$ could not type-chek if it referred to $l_{new}$. As a result, it is not possible that $e_A$ relies upon or is affected by $l_{new}$.
  - $e_B = \mathsf{spawn}_0\ e$. $e_B$ steps in the following manner: $\bullet; H; e_B \rightarrow_w \bullet; H; (0 \parallel e \parallel T)$. Since $e_B$ creates $H' = H$ and $T' = e \parallel T$, while $e_A$ steps under ENTER-ATOMIC (and, as stated earlier, preserves $H$ and $T$), it must be the case that $e_B$ neither affects $e_A$ nor is affected by it. Similarly, $e_A$ is not affected by $e_B$.
  - $e_B = \mathsf{spawn}_1\ e$. This expression is not in the current language, so this case is vacuous.
  - $e_B = \mathsf{spawn}_2\ e$. This expression is not in the current language, so this case is vacuous.
  - $e_B = \mathsf{atomic}\ e$. Here, $e_B$ cannot step under $\bullet$. This case is vacuous.
  - $e_B = e_1; e_2$. There are two cases:
    * If $e_1$ is not a value, this is true by induction.
    * If $e_1$ is some $v$, then the step for $e_B$ is to become $e_2$. Again, $e_B$ has no need to access the heap or create new threads, so $H$ and $T$ are as before.
  - $e_B = \mathsf{if}\ e_1\ e_2\ e_3$. This case is similar to that when $e_B = e_1; e_2$.
  - $e_B = e_1 := e_2$. There are three cases:
    * If $e_1$ is not a value, this is true by induction.
    * If $e_2$ is not a value, this is true by induction.
    * If $e_1$ is some $l$ and $e_2$ is some $v$. Here $e_B$ writes the heap using WEAK-SET,so $H' = H, l \mapsto v$ and since $e_A$ doesn't change $H$ this can happen either before or after $e_A$ steps.
  - $e_B = !e_1$. There are two cases:
    * If $e_1$ is not a value, this is true by induction.
    * If $e_1$ is some $l$, then the step for $e_B$ is to read the value at $l$ using WEAK-GET. Thus, $e'_B = H(l)$. By assumption, $e_A$ does not write to $H$ or create new threads, so $H$ and $T$ after $e_A$ evaluates are the same as before.
  - $e_B = e_1\, e_2$. There are three cases:
    * If $e_1$ is not a value, this is true by induction.
    * If $e_2$ is not a value, this is true by induction.

    * If $e_1$ is some $\lambda x.e_3$, and $e_2$ is some $v$, then the step for $e_B$ is to become $e_3[v/x]$, under BETA which creates $H' = H$ and $T' = T$.
  - $e_B = \mathsf{inatomic}(a, e''_B, \cdot, \cdot)$. This case violates the assumption that not-active($e_B$), so this case is vacuous.

- $e_A = e_1; e_2$. If $e_1 \neq v$, then we use induction. If $e_1 = v$, then $e_A$ steps under the SEQ-V rule, in which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = \mathsf{if}\ e_1\ e_2\ e_3$. If $e_1 \neq v$, then we use induction. If $e_1 = v$, then $e_A$ steps under the IF-Z or IF-NZ rules, under which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = e_1 := e_2$. If $e_1 \neq v$, or $e_2 \neq v$, then we use induction. If $e_1 = l_A$ and $e_2 = v_A$ then $e_A$ steps under the STRONG-SET rule, in which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = !e_1$. If $e_1 \neq v$, then we use induction. If $e_1 = l_A$, then $e_A$ steps under the STRONG-GET rule, in which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = e_1\, e_2$. If $e_1 \neq v$, or $e_2 \neq v$. then we use induction. If $e_1 = \lambda x.e$ and $e_2 = v_A$ then $e_A$ steps under the BETA rule, in which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.
- $e_A = \mathsf{inatomic}(a, e''_A, \cdot, \cdot)$. This case violates the assumption that not-active($e_A$), so this case is vacuous.

**Lemma 3.7 (Weak and Strong under different values of $a$)** *Each of the following is true: ,*

1. $\circ; H; T \rightarrow_w\ a; H'; T'$
   *If and only if*
   $\circ; H; T \rightarrow_s\ a; H'; T'$
2. $\bullet; H; T \rightarrow_w\ \circ; H; T'$
   *If and only if*
   $\bullet; H; T \rightarrow_s\ \circ; H; T'$
3. $\bullet; H; T_A \parallel e \parallel T_B \rightarrow_w\ \bullet; H'; T_A \parallel e' \parallel T_B$ *and* active*(e) and* active*(e')*
   *If and only if*
   $\bullet; H; T_A \parallel e \parallel T_B \rightarrow_s\ \bullet; H'; T_A \parallel e' \parallel T_B$ *and* active*(e) and* active*(e')*

**Lemma 3.7** The proof (by induction on the derivation of the assumed evaluation step) follows directly from examination of the weak and strong evaluation rules.

**Lemma 3.8 (Serializability of Transactions)** *Suppose:*

1. $\vdash \circ; \cdot; e$
2. $\circ; \cdot; e \rightarrow^n_w\ a; H; T$

*then there exists some sequence such that*

1. $\circ; \cdot; e \rightarrow^n_s\ a; H; T$
2. *If $a = \bullet$ then the sequence ends with* $\circ; H; T_A \parallel e_i \parallel T_B \rightarrow_s\ \bullet; H; (T_A \parallel e'_i \parallel T_B)\ \bullet; H; T \parallel e'_i \rightarrow^k_s \bullet; H; (T_A \parallel e''_i \parallel T_B)$ *where* $T_A \parallel e''_i \parallel T_B = T$

**Proof of Lemma 3.8** Proof by induction on $n$.

- When $n = 0$, neither semantics takes a step and $a = \circ$, so this case holds trivially.
- Assume the lemma is true for $n-1$, and show that this holds for $n$. By assumption:
  $\circ; \cdot; e \rightarrow^{n-1}_w\ a; H; T \rightarrow_w\ a'; H'; T'$. The following statements are true by induction on the first $n-1$ steps of evaluation:
  (A) $\circ; \cdot; e \rightarrow^{n-1}_s\ a; H; T$
  (B) If $a = \bullet$ then the sequence ends with
  $\circ; H; T_A \parallel e_i \parallel T_B \rightarrow_s\ \bullet; H; (T_A \parallel e'_i \parallel T_B) \rightarrow^k_s \bullet; H; (T_A \parallel e''_i \parallel T_B)$ where $T_A \parallel e''_i \parallel T_B = T$

There are two possibilities for $a$ after these $(n-1)$ steps of evaluation:

1. If $a = \circ$ then we need to show that the $n^{th}$ step taken under the weak semantics is also valid under the strong semantics. Lemma 3.7, part 1, gives us that any single step of evaluation under the weak semantics that begins with $a = \circ$ is also valid under the strong semantics. Thus, $\circ; H; T \rightarrow_s \ a'; H'; T'$ must be true. By combining this with fact (A) from above, then we know that $\circ; \cdot; e \rightarrow_s^n \ a'; H'; T'$ where $T_A \parallel e_i'' \parallel T_B = T'$. If $a' = \bullet$ then $e_i$ entered a transaction, and the proof conditions are satisfied when we let $k = 0$.
2. If $a = \bullet$ then by correct-atomic$(\bullet, T_A \parallel e_i \parallel T_B)$, we know that there is some active thread $e_i \in T$. If $T$ contains more than one thread, then all other threads, $e_j$ are not active. By assumption, either the active $e_i$ takes a step or (if it exists) a not-active $e_j$ takes a step.
   (a) $e_i$ took the last weak step, and active$(e_i)$. Either $e_i$ exited the atomic block on the next weak step, or $e_i$ did some evaluation within the transaction.
      – If $e_i$'s next step was to exit atomic, then by assumption the final weak evaluation rule has the following structure: $\bullet; H; T \rightarrow_w \ \circ; H'; T'$. Lemma 3.7, part 2, gives us that this evaluation is equivalent under strong evaluation, so it must be that $\bullet; H; T \rightarrow_s \ \circ; H'; T'$. By combining this statement with fact (A) above, we know that $\circ; \cdot; e \rightarrow_s^n \ \circ; H'; T'$.
      – $e_i$ did some evaluation inside a transaction and did not exit the transaction. Thus the final weak evaluation rule has the following structure: $\bullet; H; T \rightarrow_w \ \bullet; H'; T'$. We also know that active$(e_i)$ so Lemma 3.7, part 3, gives us that this step is equivalent under strong evaluation, so it must be that $\bullet; H; T \rightarrow_s \ \bullet; H'; T'$. By combining this statement with fact (B) above, we know that there is a sequence of $n$ steps of evaluation that ends with $\circ; H; T_A \parallel e_i \parallel T_B \rightarrow_s \ \bullet; H; (T_A \parallel e_i' \parallel T_B) \rightarrow_s^{k+1} \bullet; H; (T_A \parallel e_i'' \parallel T_B)$ where $T_A \parallel e_i'' \parallel T_B = T$. Because active transactions do not spawn threads at the top level, we know that $T = T'$.

   (b) If the next step was some $e_j$ where not-active$(e_j)$, then we need to show that this step can be moved before $e_i$ enters the transaction (by symmetry, we assume $T_B = e_j \parallel T_B'$). By applying Lemma 3.2, $k$ times and using fact (B) above, we can get the following order of evaluation:
      $\circ; H; (T_A \parallel e_i \parallel T_B) \ \rightarrow_s \ \ \bullet; H; (T_A \parallel e_i' \parallel e_j \parallel T_B') \ \rightarrow_w \ \bullet; H; (T_A \ \parallel \ e_i' \ \parallel \ e_j' \ \parallel \ T_B') \ \rightarrow_s^k$ $\bullet; H; (T_A \parallel e_i'' \parallel e_j' \parallel T_B')$, where $T_A \parallel e_i'' \parallel e_j' \parallel T_B' = T$.
      Next, it is possible to move the step for $e_j$ before $e_i$ enters the transaction using Lemma 3.6, which gives:
      $\circ; H; (T_A \parallel e_i \parallel e_j \parallel T_B') \rightarrow_w \ \ \circ; H; (T_A \parallel e_i \parallel e_j' \parallel T_B'') \rightarrow_s \bullet; H; (T_A \ \parallel \ e_i' \ \parallel \ e_j' \ \parallel \ T_B'') \rightarrow_s^k$ $\bullet; H; (T_A \parallel e_i'' \parallel e_j' \parallel T_B'')$, where $T_A \parallel e_i'' \parallel e_j' \parallel T_B'' = T$.
      Then, Lemma 3.7, part 1 gives that the step for $e_j$ when $a = \circ$ is equivalent to the same step under strong semantics, giving the desired result:
      $\circ; H; (T_A \parallel e_i \parallel e_j \parallel T_B') \rightarrow_s \ \ \circ; H; (T_A \parallel e_i' \parallel e_j' \parallel T_B'') \rightarrow_s \bullet; H; (T_A \ \parallel \ e_i' \ \parallel \ e_j' \ \parallel \ T_B'') \rightarrow_s^k$ $\bullet; H; (T_A \parallel e_i'' \parallel e_j' \parallel T_B'')$, where $T_A \parallel e_i'' \parallel e_j' \parallel T_B'' = T$.

**Theorem 3.9 (Weak and Strong Equivalence)** *If* $\vdash \circ; \cdot; e$ *(using the type-system from Section 3.3), then* $\circ; \cdot; e \rightarrow_s^n \ a; H; T$ *iff* $\circ; \cdot; e \rightarrow_w^n \ a; H; T$.

**Proof** There are two directions to prove for the iff statement:

- "strong implies weak": Every trace using the strong evaluation rules *is* an equivalent trace under the weak evaluation rules. This is because the weak rules allow a strict super set of traces.
- "weak implies strong": Follows as a corollary to Lemma 3.8 (the first conclusion of that lemma is what we need).

# 4 The StrongBasic language

To explore similar equivalence results to *even weaker* syntax for software transactions, we simplify the StrongNestedParallel language by removing internal parallelism and retain the top-level parallelism. We call this simplified language StrongBasic.

## 4.1 Syntax

To summarize the syntax for the StrongBasic language, we redefine $e$ by removing the syntax for $\mathsf{spawn}_0$, $\mathsf{spawn}_1$, and $\mathsf{spawn}_2$ from StrongNestedParallel and add a new generic form of top-level thread creation: spawn. We have also replaced inatomic with appropriately simplified syntax.

It is valuable to note that the new syntax for inatomic does *not* contain an internal version of $a$, because there are no internally parallel threads. This results in some interesting features in the updated typing rule in Section 4.3.

$$\begin{array}{rcl} e & ::= & c \mid l \mid x \mid e_1; e_2 \mid e_1 := e_2 \mid \text{ref } e \mid !e \mid \lambda x.e \mid e_1\ e_2 \\ & & \mid \mathsf{spawn}\ e \mid \mathsf{atomic}\ e \mid \mathsf{inatomic}(e) \end{array}$$

## 4.2 Dynamic Evaluation

### 4.2.1 Whole Program Evaluation

Since we have removed the two internally parallel types of spawn from StrongNestedParallel, we can only create a single, top-level, inductive $T$ whenever $e$ takes a step. This changes the form PROGRAM rule only slightly (to accommodate the new evaluation form for $e$).

$\boxed{a; H; T \rightarrow\ a'; H'; T'}$

PROGRAM
$$\frac{a; H; e \rightarrow a'; H'; e'; T}{a; H; T_A \parallel e \parallel T_B \rightarrow\ a'; H'; T_A \parallel e' \parallel T_B \parallel T}$$

### 4.2.2 Expression Evaluation

Removing the internally parallel versions of spawn changes the judgment form of $e$ in the intuitive manner. We have included the SEQ-1 and STRONG-GET rules as examples of how to modify the other evaluation rules to handle empty and inductive values of $T$. We have also modified the ENTER ATOMIC, INATOMIC, and EXIT ATOMIC rules to reflect the syntax changes for atomic blocks and introduced a new SPAWN rule to accommodate the syntax changes for thread creation. Although we removed the INATOMIC HELPER, SPAWN 0, SPAWN 1, and SPAWN 2 rules, note the similarities between SPAWN in this language and SPAWN 0 in the StrongNestedParallel language.

$\boxed{a; H; e \rightarrow a'; H'; e'; T}$

SEQ-1
$$\frac{a; H; e_1 \rightarrow a'; H'; e_1'; T}{a; H; e_1; e_2 \rightarrow a'; H'; e_1'; e_2; T}$$

STRONG-GET
$$\frac{}{\circ; H; !l \rightarrow \circ; H; H(l); \cdot}$$

SPAWN
$$\frac{}{a; H; \mathsf{spawn}\ e \rightarrow a; H; 0; e}$$

ENTER ATOMIC
$$\frac{}{\circ; H; \mathsf{atomic}\ e \rightarrow \bullet; H; \mathsf{inatomic}(e); \cdot}$$

INATOMIC
$$\frac{a; H; e \rightarrow a'; H'; e'; \cdot}{\bullet; H; \mathsf{inatomic}(e) \rightarrow \bullet; H; \mathsf{inatomic}(e'); \cdot}$$

EXIT ATOMIC
$$\frac{}{\bullet; H; \mathsf{inatomic}(v) \rightarrow \circ; H; v; \cdot}$$

## 4.3 Typecheck $e$

As with the syntax and semantics changes, we make minor adjustments to type-checking $e$ under $\varepsilon$. We remove the typing rules for no-longer-existent syntax ($\mathsf{spawn}_0$, $\mathsf{spawn}_1$, $\mathsf{spawn}_2$) add a rule for the new $\mathsf{spawn}$ syntax, and modify the $\mathsf{inatomic}$ rule to accommodate the syntax changes. Note the difference in the T-INATOMIC rule's use of correct-atomic. We still require that there be a meaningful correspondence between $e$ and $a$, although without $T_2$ the syntax does not actually need to keep track of an internal value of $a$ for coordination purposes.

$\boxed{\Gamma; \varepsilon \vdash e : \tau}$

$$\text{T-SPAWN}\quad \frac{\Gamma; 0 \vdash e : \tau}{\Gamma; 0 \vdash \mathsf{spawn}\, e : \mathrm{int}} \qquad\qquad \text{T-INATOMIC}\quad \frac{\Gamma; 2 \vdash e : \tau \qquad \text{correct-atomic}(a, e)}{\Gamma; \varepsilon \vdash \mathsf{inatomic}(e) : \tau}$$

## 4.4 Other Typing Rules

No Changes.

## 4.5 Activeness

### 4.5.1 Not-Active($e$)

We remove the not-active($e$) rules for deleted syntax ($\mathsf{spawn}_0$, $\mathsf{spawn}_1$, and $\mathsf{spawn}_2$). We add a rule for the added syntax ($\mathsf{spawn}$):

$\boxed{\text{not-active}(e)}$

$$\frac{\text{not-active}(e)}{\text{not-active}(\mathsf{spawn}\, e)}$$

### 4.5.2 Active($e$)

Similarly, we update the active($e$) rule for $\mathsf{inatomic}$:

$\boxed{\text{active}(e)}$

$$\frac{}{\text{active}(\mathsf{inatomic}(e))}$$

### 4.5.3 Correct Atomic State of $T$

No Changes.

## 4.6 Type Safety

It is not necessary to show that this language is typesafe.

## 4.7 Equivalence

## 5 The StrongUndo language

This language is like StrongBasic in that it has strong atomicity and no internal parallelism, but like WeakUndo in that it has logging-and-rollback. This combination is bizarre by itself, but our purpose in defining this language is to stage the equivalence of WeakUndo to StrongBasic by proving WeakUndo equivalent to StrongUndo, which largely involves serializing transactions much like the proof of equivalence between StrongNestedParallel and Weak, and proving the equivalence of StrongUndo to WeakUndo, which largely involves proving that rolled back transactions have no effects other than to allocate unused heap locations.

To make matters even easier, StrongUndo is "very" strong, meaning that if one thread is active, then no other thread can take *any* step (by contrast StrongBasic allows steps like beta-reductions to be interleaved with transactions).

We also, for now at least, prevent nested transactions by restricting the type system. This dramatically simplifies the equivalence proof, though we believe the more general result holds. Many of the lemmas stated for equivalence is false and/or too narrow in the presence of nested transactions.

This language is probably easier to understand after WeakUndo, but putting it in-between StrongBasic and WeakUndo makes sense from the perspective of the equivalence proof. Note that in both languages, logs, which are heaps, are treated syntactically. In particular, they may have repeated elements. More specifically, when $H$ is used as a heap (e.g., as part of a program state), $H, l \mapsto v$ or $HH'$ is partial-map update, with the new bindings on the right shadowing those on the left. However, when $H$ is used as a log, $H, l \mapsto v$ or $HH'$ is treated as syntactic extension, so if/when we rollback using the log, the *leftmost* binding for any $l$ will "win". This abuse of notation will hopefully be clear from context; perhaps the conference-paper should not overload juxtaposition in this way.

### 5.1 Syntax

The syntax is the same as for WeakUndo:

$$\begin{array}{rcl} e & ::= & c \mid l \mid x \mid e_1; e_2 \mid e_1 := e_2 \mid \text{ref } e \mid !e \mid \lambda x.e \mid e_1\ e_2 \\ & & \mid \mathsf{spawn}\, e \mid \mathsf{atomic}\ e \mid \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) \mid \mathsf{inrollback}(H_{\mathsf{log}}, e_0) \end{array}$$

### 5.2 Dynamic Evaluation

The form of the judgments and the treatment of logs is the same as for WeakUndo, but the treatment of $a$ is like StrongBasic, but "even stronger".

#### 5.2.1 Whole Program Evaluation

$\boxed{a; H; T \to a'; H'; T'}$

$$\textsc{program}\quad \frac{a; H; e \to a'; H'; e'; T; H_{\mathsf{log}}}{a; H; T_A \parallel e \parallel T_B \to a'; H'; T_A \parallel e' \parallel T_B \parallel T}$$

#### 5.2.2 Expression Evaluation

Not shown are rules SET-1, SET-2, REF-1, GET-1, APP-1 and APP-2, which are analogous to SEQ-1.

$\boxed{a; H; e \to a'; H'; e'; T; H_{\mathsf{log}}}$

SEQ-1

$$\frac{a; H; e_1 \rightarrow a'; H'; e_1'; T; H_{\mathsf{log}}}{a; H; (e_1; e_2) \rightarrow a'; H'; (e_1'; e_2); T; H_{\mathsf{log}}}$$

SEQ-V

$$\frac{}{\circ; H; (v; e_2) \rightarrow \circ; H; e_2; \cdot; \cdot}$$

ALLOC

$$\frac{l \notin \mathrm{Dom}(H)}{\circ; H; \mathrm{ref}\ v \rightarrow \circ; H, l \mapsto v; l; \cdot; \cdot}$$

BETA

$$\frac{}{\circ; H; (\lambda x.e)\ v_2 \rightarrow \circ; H; e[v_2/x]; \cdot; \cdot}$$

SPAWN

$$\frac{}{\circ; H; \mathsf{spawn}\ e \rightarrow \circ; H; 0; e; \cdot}$$

SET-EAGER

$$\frac{}{\circ; H; l := v \rightarrow \circ; H, l \mapsto v; v; \cdot; l \mapsto H(l)}$$

GET-EAGER

$$\frac{}{\circ; H; !l \rightarrow \circ; H; H(l); \cdot; \cdot}$$

ENTER ATOMIC

$$\frac{}{\circ; H; \mathsf{atomic}\ e \rightarrow \bullet; H; \mathsf{inatomic}(\circ, e, \cdot, e); \cdot; \cdot}$$

INATOMIC

$$\frac{a; H; e \rightarrow a'; H'; e'; \cdot; H_{\mathsf{log}}'}{\bullet; H; \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) \rightarrow \bullet; H'; \mathsf{inatomic}(a', e', H_{\mathsf{log}} H_{\mathsf{log}}', e_0); \cdot; \cdot}$$

ENTER ROLLBACK

$$\frac{}{\bullet; H; \mathsf{inatomic}(\circ, e, H_{\mathsf{log}}, e_0) \rightarrow \bullet; H; \mathsf{inrollback}(H_{\mathsf{log}}, e_0); \cdot; \cdot}$$

DO ROLLBACK

$$\frac{}{\bullet; H; \mathsf{inrollback}(H_{\mathsf{log}}, l \mapsto v_{\mathsf{old}}, e_0) \rightarrow \bullet; H, l \mapsto v_{\mathsf{old}}; \mathsf{inrollback}(H_{\mathsf{log}}, e_0); \cdot; \cdot}$$

COMPLETE ROLLBACK

$$\frac{}{\bullet; H; \mathsf{inrollback}(\cdot, e_0) \rightarrow \circ; H; \mathsf{atomic}\ e_0; \cdot; \cdot}$$

COMMIT

$$\frac{}{\bullet; H; \mathsf{inatomic}(\circ, v, H_{\mathsf{log}}, e_0) \rightarrow \circ; H; v; \cdot; H_{\mathsf{log}}}$$

## 5.3 Typecheck $e$

The type system is the same as for WeakUndo except we change T-ATOMIC and T-INATOMIC to prevent nested transactions, simplifying the equivalence proof for now. The essential change is putting 0 in the rules' conclusions instead of $\varepsilon$; also notice the $a$ inside inatomic is always $\circ$.

$\boxed{\Gamma; \varepsilon \vdash e : \tau}$

T-ATOMIC

$$\frac{\Gamma; 2 \vdash e : \tau}{\Gamma; 0 \vdash \mathsf{atomic}\ e : \tau}$$

T-INROLLBACK

$$\frac{\Gamma; 2 \vdash e_0 : \tau \qquad \Gamma \vdash H_{\mathsf{log}} : \Gamma' \qquad \Gamma \text{ extends } \Gamma' \qquad \mathsf{all2}(\Gamma') \qquad \text{not-active}(e_0)}{\Gamma; 0 \vdash \mathsf{inrollback}(H_{\mathsf{log}}, e_0) : \tau}$$

T-INATOMIC

$$\frac{\text{correct-atomic}(\circ, e) \qquad \Gamma; 2 \vdash e_0 : \tau \qquad \begin{array}{c}\Gamma; 2 \vdash e : \tau \\ \text{not-active}(e_0)\end{array} \qquad \Gamma \vdash H_{\mathsf{log}} : \Gamma' \qquad \Gamma \text{ extends } \Gamma' \qquad \mathsf{all2}(\Gamma')}{\Gamma; 0 \vdash \mathsf{inatomic}(\circ, e, H_{\mathsf{log}}, e_0) : \tau}$$

## 5.4 Other Typing Rules

No changes

## 5.5 Activeness

The type system is the same as for WeakUndo. However, it's convenient when stating lemmas to have a form of the active judgment that also computes what, if we were to enter rollback, we would use for the log and the rolled-back expression, as well as whether or not we are already in rollback. We use ir for "in rollback" and ia for "in atomic" — any active expression is one or the other.

$$r ::= \mathsf{ir} \mid \mathsf{ia}$$

$\boxed{\text{activeplus}(e; H_{\mathsf{log}}; e_0; r)}$

$$\frac{}{\text{activeplus}(\mathsf{inatomic}(\circ, e, H_{\mathsf{log}}, e_0); H_{\mathsf{log}}; \mathsf{atomic}\ e_0; \mathsf{ia})} \qquad \frac{}{\text{activeplus}(\mathsf{inrollback}(H_{\mathsf{log}}, e_0); H_{\mathsf{log}}; \mathsf{atomic}\ e_0; \mathsf{ir})}$$

$$\frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}((e_1; e_2); H_{\mathsf{log}}; (e_0; e_2); r)} \qquad \frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}(e_1 := e_2; H_{\mathsf{log}}; e_0 := e_2; r)}$$

$$\frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}(v := e_1; H_{\mathsf{log}}; v := e_0; r)} \qquad \frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}(\text{ref } e_1; H_{\mathsf{log}}; \text{ref } e_0; r)} \qquad \frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}(!e_1; H_{\mathsf{log}}; !e_0; r)}$$

$$\frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}(e_1\ e_2; H_{\mathsf{log}}; e_0\ e_2; r)} \qquad \frac{\text{activeplus}(e_1; H_{\mathsf{log}}; e_0; r)}{\text{activeplus}(v\ e_1; H_{\mathsf{log}}; v\ e_0; r)}$$

## 5.6 Type Safety

The type-safety proof is almost the same as for WeakUndo, but we have to account for changes to the dynamic semantics (being "very" strong instead of weak) and the type system (preventing nested transactions). We summarize the relevant issues here (this probably deserves fleshing out a bit):

- Progress: The stricter type system cannot be a problem. For the stricter dynamic semantics, note that when $a = \bullet$ the proof for WeakUndo always chooses the active thread to take a step, which we can still do, i.e., we never needed the "weak" in weak-atomicity to take a step.
- Preservation: The stricter dynamic semantics cannot be a problem. For the stricter type system, we need to strengthen Single Thread Preservation to argue as follows: The third bullet in conclusion 6 now reads, "If $a = \circ$ and $a' = \bullet$ then not-active($e$) and active($e'$) and $\epsilon = 0$." In other words, the expression can enter a transaction only if it type-checked under effect 0 before the step. This is relevant in various cases of the proof:
  - If $e = \mathsf{atomic}\ e_1$, then under our new T-ATOMIC $\epsilon = 0$, so the conclusion holds.
  - If $e$ steps via INATOMIC, then under our new T-INATOMIC, the inner expression steps with $\circ$ and type-checks under 2, so inductively, the inner step must produce $\circ$ and preserve the not-activeness of the inner expression.
  - For the various inductive cases, we need that the subexpression type-checks under the same effect as the whole expression, so we can still conclude the third bullet.

The key new corollary of Preservation, which arises directly from inspection of T-INATOMIC is that the body of an inatomic expression is not-active.

## 5.7 Equivalence

We show only that traces in this language are "mostly" possible in StrongBasic. The other direction is nontrivial (since we are "very" strong instead of strong), but showing traces in StrongBasic are possible in WeakUndo is trivial, i.e., we do not use this intermediate language for that direction. By "mostly" we must account for:

- Allocations inside transactions that rollback lead to garbage in the heap for StrongUndo that will not be there in StrongBasic, so we allow StrongUndo to produce an extension of the heap produced under StrongBasic.
- When StrongUndo is doing a rollback there is not corresponding trace in StrongBasic, so we have to say the state after the entire rollback is reachable under StrongBasic. Relatedly, rollback means StrongBasic may need to take fewer steps to reach a state reached by StrongUndo.
- Though the source languages are the same, inatomic expressions in StrongBasic do not have logs.

The heart of the proof is showing that rollback is correct, which requires a strengthened induction hypothesis to demonstrate that at any point in a transaction, the logging can be used to return to an appropriate heap. Stating this correctly is complicated significantly by nested transactions (hence we banned them) and moderately by the "allocations inside transactions" issue. This issues motivate some of the definitions that precede the key lemmas.

We first define a simple "translate" metafunction for removing the log. The idea is that if translate($e_1; e_2$), then $e_1$ in StrongUndo corresponds to $e_2$ in StrongBasic. The metafunction is partial (e.g., expressions doing rollback have no counterpart) and encodes the very simple idea of simplifying the at-most-one inatomic expression in a program. We also extend it to thread-pools in the natural way.

Note we write $a; H; e \rightarrow a'; H'; e'; T; H_{\mathsf{log}}$ for a step in StrongUndo and $a; H; e \rightarrow_{\mathsf{no}} a'; H'; e'; T$ for a step in StrongBasic, and similarly for top-level evaluation.

$\boxed{\text{translate}(e_1; e_2)}$

$$\frac{\text{not-active}(e)}{\text{translate}(e; e)} \qquad \frac{\text{not-active}(e)}{\text{translate}(\mathsf{inatomic}(\circ, e, H_{\mathsf{log}}, e_0); \mathsf{inatomic}(e))} \qquad \frac{\text{translate}(e_1; e_1')}{\text{translate}((e_1; e_2); (e_1'; e_2))}$$

$$\frac{\text{translate}(e_1; e_1')}{\text{translate}(e_1 := e_2; e_1' := e_2)} \qquad \frac{\text{translate}(e_1; e_1')}{\text{translate}(v := e_1; v := e_1')} \qquad \frac{\text{translate}(e_1; e_1')}{\text{translate}(\text{ref } e_1; \text{ref } e_1')}$$

$$\frac{\text{translate}(e_1; e_1')}{\text{translate}(!e_1; !e_1')} \qquad \frac{\text{translate}(e_1; e_1')}{\text{translate}(e_1\ e_2; e_1'\ e_2)} \qquad \frac{\text{translate}(e_1; e_1')}{\text{translate}(v\ e_1; v\ e_1')}$$

**Definition 5.1 (Heap Extension)** *$H'$ extends $H$ if for all $l \in \text{Dom}(H)$, $H'(l) = H(l)$.*

**Definition 5.2 (Rollsbackto)** *The definition of* rollsbackto($H_{\mathsf{pre}}; H_{\mathsf{log}}; H_{\mathsf{post}}$) *defines when doing rollback with log $H_{\mathsf{log}}$ should transform $H_{\mathsf{pre}}$ to $H_{\mathsf{post}}$. We have two inference rules:*

$$\frac{}{\text{rollsbackto}(H; \cdot; H)} \qquad \frac{l \in \text{Dom}(H) \qquad \text{rollsbackto}(H, l \mapsto v; H'; H'')}{\text{rollsbackto}(H; H', l \mapsto v; H'')}$$

The $H, l \mapsto v$ is in the partial-map sense and the $H', l \mapsto v$ is in the syntactic-log sense.

**Lemma 5.3 (Rollback Extension)** *If* rollsbackto($H; H_{\mathsf{log}}; H'$) *and $l \notin \text{Dom}(H)$, then* rollsbackto($H, l \mapsto v; H_{\mathsf{log}}; H', l \mapsto v$)

**Proof** By induction on the derivation of rollsbackto($H; H_{\mathsf{log}}; H'$)

**Lemma 5.4 (Active-Plus Determinism)** active($e$) *if and only if there exist unique $H_{\mathsf{log}}$, $e_0$, and $r$ such that* activeplus($e; H_{\mathsf{log}}; e_0; r$).

**Proof** By induction on the derivation of not-active($e$). The base cases are immediate and the inductive cases are straightforward.

**Lemma 5.5 (Starting Inactive)** *If:*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; \epsilon \vdash e : \tau$
3. not-active($e$)
4. $\circ; H; e \rightarrow a; H'; e'; T; H'_{\mathsf{log}}$

*Then* $\circ; H; e \rightarrow_{\mathsf{no}} a; H'; e''; T$ *where* translate($e'; e''$).

**Proof** By induction on the derivation of assumption 4, with cases on the bottommost rule. Note in the immediate cases except ENTER-ATOMIC, not-active($e$) and Preservation ensure not-active($e'$) so $e'' = e'$.

- SEQ-1 Let $e$ be $e_1; e_2$ and $e' = e_1'; e_2$. Inverting assumptions, 2, 3, and 4 ensures induction applies and $\circ; H; e_1 \rightarrow_{\mathsf{no}} a; H'; e_1''; \cdot$ where translate($e_1'; e_1''$). So we can derive $\circ; H; (e_1; e_2) \rightarrow_{\mathsf{no}} a; H'; (e_1''; e_2); \cdot$ and translate($e_1'; e_2; e_1''; e_2$).

- SET-1 Analogous to SEQ-1 with $e$ being $e_1 := e_2$
- SET-2 Analogous to SEQ-1 with $e$ being $v := e_1$
- REF-1 Analogous to SEQ-1 with $e$ being ref $e_1$
- GET-1 Analogous to SEQ-1 with $e$ being $!e_1$
- APP-1 Analogous to SEQ-1 with $e$ being $e_1\ e_2$
- APP-2 Analogous to SEQ-1 with $e$ being $v\ e_1$
- SEQ-V Immediate; assumption 4 ensures we can take the same step in $\mathsf{StrongBasic}$.
- ALLOC Immediate; assumption 4 ensures we can take the same step in $\mathsf{StrongBasic}$.
- BETA Immediate; assumption 4 ensures we can take the same step in $\mathsf{StrongBasic}$.
- SPAWN Immediate; assumption 4 ensures we can take the same step in $\mathsf{StrongBasic}$.
- SET-EAGER Immediate; assumption 4 ensures we can take the same step in $\mathsf{StrongBasic}$.
- GET-EAGER Immediate; assumption 4 ensures we can take the same step in $\mathsf{StrongBasic}$.
- ENTER-ATOMIC Immediate; assumption 4 ensures we can take a similar step in $\mathsf{StrongBasic}$; the difference is mediated by translate$(e'; e'')$.
- INATOMIC Vacuous given assumption 4 ($\circ$ vs. $\bullet$)
- ENTER ROLLBACK Vacuous given assumption 4
- DO ROLLBACK Vacuous given assumption 4
- COMPLETE ROLLBACK Vacuous given assumption 4
- COMMIT Vacuous given assumption 4

**Lemma 5.6 (Becoming Active)** *If:*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; \epsilon \vdash e : \tau$
3. not-active$(e)$
4. $\circ; H; e \to \bullet; H'; e'; T; H'_{\mathsf{log}}$

*Then* $H' = H$, $T = \cdot$, $H'_{\mathsf{log}} = \cdot$, *and* activeplus$(e'; \cdot; e; \mathsf{ia})$.

**Proof** By induction on the derivation of assumption 4, with cases on the bottommost rule used:

- ENTER ATOMIC: by inspection of the rule $H' = H$, $T = \cdot$, and $H'_{\mathsf{log}} = \cdot$. Furthermore, $e$ has the form $\mathsf{atomic}\ e_1$ and $e'$ is is $\mathsf{inatomic}(\circ, e_1, \cdot, e_1)$. We can derive activeplus$(\mathsf{inatomic}(\circ, e_1, \cdot, e_1); \cdot; \mathsf{atomic}\ e_1; \mathsf{ia})$, which is what we need since $e$ is $\mathsf{atomic}\ e_1$.
- SEQ-1 Let $e$ be $e_1; e_2$ and $e' = e'_1; e_2$. Inverting assumptions, 2, 3, and 4 ensures induction applies so $H' = H$, $T = \cdot$, $H'_{\mathsf{log}} = \cdot$, and activeplus$(e'_1; \cdot; e_1; \mathsf{ia})$. So we can derive activeplus$(e'_1; e_2; \cdot; e_1; e_2; \mathsf{ia})$, which is what we need since $e$ is $e_1; e_2$.
- SET-1, SET-2, REF-1, GET-1, APP-1, and APP-2 are analogous to SEQ-1.
- SEQ-V, ALLOC, BETA, SPAWN, SET-EAGER, GET-EAGER, INATOMIC, ENTER ROLLBACK, DO ROLLBACK, COMPLETE ROLLBACK, and COMMIT are all vacuous since we need to step from a state with $\circ$ to a state with $\bullet$.

**Lemma 5.7 (Becoming Inactive)** *If:*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; \epsilon \vdash e : \tau$
3. activeplus$(e; H_{\mathsf{log}}; e_0; r)$
4. $\bullet; H; e \to \circ; H'; e'; T; H'_{\mathsf{log}}$

*Then:*

- *If* $r = \mathsf{ir}$*, then* $H_{\mathsf{log}} = H'_{\mathsf{log}} = \cdot,$*,* $T = \cdot$*,* $H' = H$*, and* $e' = e_0$*.*
- *If* $r = \mathsf{ia}$*, then* $H' = H$ *and* $\bullet; H; e'' \rightarrow_{\mathsf{no}} \circ; H; e'; T$ *where* translate$(e; e'')$*.*

**Proof** By induction on the derivation of assumption 4, with cases on the bottommost rule used:

- COMPLETE ROLLBACK: By inspecting the form of the rule and inverting assumption (3), $e$ is $\mathsf{inrollback}(\cdot, e_0)$ and $e'$ is $\mathsf{atomic}\ e_0$. The second result holds vacuously; we can derive only activeplus$(e; \cdot; \mathsf{atomic}\ e_0; \mathsf{ir})$. The first result then follows directly form the form of the COMPLETE ROLLBACK rule.
- COMMIT: By inspecting the form of the rule and inverting assumption (3), there must exist a $v$ such that $e$ is $\mathsf{inatomic}(\circ, v, H_{\mathsf{log}}, e_0)$. The first result holds vacuously; we can derive only activeplus$(e; H_{\mathsf{log}}; e_0; \mathsf{ia})$ The second result holds because translate$(e; e'')$ ensures $e''$ is $\mathsf{inatomic}(v)$ and we can derive $\bullet; H; \mathsf{inatomic}(v) \rightarrow_{\mathsf{no}} \circ; H; v; \cdot$ (note from the form of the rule $H' = H$ and $T = \cdot$).
- SEQ-1: Let $e$ be $e_1; e_2$ and $e' = e'_1; e_2$. Inverting assumptions, 2, 3, and 4 ensures induction applies so:
  - If $r = \mathsf{ir}$, then $H_{\mathsf{log}} = H'_{\mathsf{log}} = \cdot$, $T = \cdot$, and where $e'_1 = e'_0$ where $e_0 = e'_0; e_2$. Hence we can use $\bullet; H; e_1 \rightarrow \circ; H; e'_0; \cdot; \cdot$ to derive $\bullet; H; e_1; e_2 \rightarrow \circ; H; e'_0; e_2; \cdot; \cdot$.
  - If $r = \mathsf{ia}$, then $H' = H$ and $\bullet; H; e''_1 \rightarrow_{\mathsf{no}} \circ; H; e'_1; T$ where translate$(e_1; e''_1)$. Hence we can derive $\bullet; H; e''_1; e_2 \rightarrow_{\mathsf{no}} \circ; H; e'_1; e_2; T$ and translate$(e; e'')$ ensures $e''$ is $e''_1; e_2$.
- SET-1, SET-2, REF-1, GET-1, APP-1, and APP-2 are analogous to SEQ-1.
- SEQ-V, ALLOC, BETA, SPAWN, SET-EAGER, GET-EAGER, INATOMIC, ENTER-ROLLBACK, DO-ROLLBACK, and ENTER-ATOMIC are all vacuous since we need to step from a state with $\bullet$ to a state with $\circ$.

**Lemma 5.8 (Correct Logging Inside Transactions)** *If:*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; 2 \vdash e : \tau$
3. not-active$(e)$
4. $\circ; H; e \rightarrow \circ; H'; e'; \cdot; H'_{\mathsf{log}}$
5. rollsbackto$(H; H_{\mathsf{log}}; H_0)$

*then* $\exists H'_0$ *such that* $H'_0$ *extends* $H_0$ *and* rollsbackto$(H'; H_{\mathsf{log}}H'_{\mathsf{log}}; H'_0)$*. (Note by* $H_{\mathsf{log}}H'_{\mathsf{log}}$ *we mean syntactic extension.)*

**Proof** By induction on the derivation of $\circ; H; e \rightarrow \circ; H'; e'; \cdot; \cdot H'_{\mathsf{log}}$, with cases for the bottommost rule:

- SEQ-1 Let $e$ be $e_1; e_2$ By inverting the derivations of assumptions 2, 3, and 4 we have $\Gamma; 2 \vdash e_1 : \tau$, not-active$(e_1)$, and $\circ; H; e_1 \rightarrow \circ; H'; e'_1; \cdot; \cdot H'_{\mathsf{log}}$ for some $e'_1$. So the result follows from induction.
- SET-1 Analogous to SEQ-1 with $e$ being $e_1 := e_2$
- SET-2 Analogous to SEQ-1 with $e$ being $v := e_1$
- REF-1 Analogous to SEQ-1 with $e$ being ref $e_1$
- GET-1 Analogous to SEQ-1 with $e$ being $!e_1$
- APP-1 Analogous to SEQ-1 with $e$ being $e_1\ e_2$
- APP-2 Analogous to SEQ-1 with $e$ being $v\ e_1$
- SEQ-V The result follows from assumption 5 letting $H'_0 = H_0$ since $H' = H$ and $H'_{\mathsf{log}} = \cdot$.
- ALLOC The result follows from assumption 5 and the Rollback Extension Lemma letting $H'_0 = H_0, l \mapsto v$ since this $H'_0$ extends $H_0$ and $H'_{\mathsf{log}} = \cdot$.
- BETA Analogous to SEQ-V
- SPAWN Vacuous given assumption 4

- SET-EAGER Let $e$ be $l := v$, $H' = H, l \mapsto v$ (in the partial-map sense) and $H'_{\mathsf{log}} = l \mapsto H(l)$. Letting $H'_0 = H_0$, we need rollsbackto$(H, l \mapsto v; H_{\mathsf{log}}, l \mapsto H(l); H_0)$. From the definition of rollsbackto, it suffices to show rollsbackto$(H, l \mapsto v, l \mapsto H(l); H_{\mathsf{log}}; H_0)$, which is assumption 4 recalling that the left heap is treated as a partial map.
- GET-EAGER Analogous to SEQ-V
- ENTER ATOMIC Vacuous given assumption 4
- INATOMIC Vacuous given assumption 4
- ENTER ROLLBACK Vacuous given assumption 4
- DO ROLLBACK Vacuous given assumption 4
- COMPLETE ROLLBACK Vacuous given assumption 4
- COMMIT Vacuous given assumption 4

**Lemma 5.9 (Staying Active)** *If:*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; \epsilon \vdash e : \tau$
3. activeplus$(e; H_{\mathsf{log}}; e_0; r)$
4. $\bullet; H; e \rightarrow \bullet; H'; e'; T; H''_{\mathsf{log}}$
5. rollsbackto$(H; H_{\mathsf{log}}; H_0)$

*Then there exist $H'_0$, $H'_{\mathsf{log}}$, and $r'$ such that:*

1. *$T = \cdot$ and $H''_{\mathsf{log}} = \cdot$*
2. rollsbackto$(H'; H'_{\mathsf{log}}; H'_0)$ *and $H'_0$ extends $H_0$*
3. activeplus$(e'; H'_{\mathsf{log}}; e_0; r')$
4. *If $r = \mathsf{ir}$, then $r' = \mathsf{ir}$.*
5. *If $r = r' = \mathsf{ia}$, then $\bullet; H; e_{no} \rightarrow_{\mathsf{no}} \bullet; H'; e'_{no}; T$ where* translate$(e; e_{no})$ *and* translate$(e'; e'_{no})$.

**Proof** By induction on the derivation of assumption (4), proceeding by cases on the bottommost rule used:

- INATOMIC Inverting assumptions (2), (3), and (4), $e$ has the form $\mathsf{inatomic}(\circ, e_1, H_{\mathsf{log}}, e_0)$ and $e'$ has the form $\mathsf{inatomic}(\circ, e'_1, H_{\mathsf{log}} H_{\mathsf{log}_1}, e_0)$ where not-active$(e_1)$, $\Gamma; 2 \vdash e_1 : \tau$, and $\circ; H; e_1 \rightarrow \circ; H'; e'_1; \cdot; H_{\mathsf{log}_1}$. So the Correct Logging Inside Transactions Lemma ensures there exists an $H'_0$ that extends $H_0$ and rollsbackto$(H; H_{\mathsf{log}} H_{\mathsf{log}_1}; H'_0)$, satisfying result (2). Results (1) and (3) are immediate from the form of the rule. Result (4) is vacuous because $r = \mathsf{ia}$. Result (5) can be derived directly from $\circ; H; e_1 \rightarrow \circ; H'; e'_1; \cdot; H_{\mathsf{log}_1}$.
- ENTER ROLLBACK Assumption (4) ensures $e$ has the form $\mathsf{inatomic}(\circ, e_1, H_{\mathsf{log}}, e_0)$ and $e'$ has the form $\mathsf{inrollback}(H_{\mathsf{log}}, e_0)$. Also $H' = H$ and result (1) is immediate. Letting $H'_{\mathsf{log}} = H_{\mathsf{log}}$ and $H'_0 = H_0$, result (2) is assumption (5). Further letting $r' = \mathsf{ir}$, result (3) can be derived directly. Result (4) is vacuous because $r = \mathsf{ia}$. Result (5) is vacuous because $r' = \mathsf{ir}$.
- DO ROLLBACK Assumption (4) ensures $e$ has the form $\mathsf{inrollback}(H_{\mathsf{log}_1}, l \mapsto v, e_0)$ and $e'$ has the form $\mathsf{inrollback}(H_{\mathsf{log}_1}, e_0)$. Also $H' = H, l \mapsto v$ and result (1) is immediate. Letting $H'_{\mathsf{log}} = H_{\mathsf{log}_1}$ and $H'_0 = H_0$, inverting assumption (5) ensures result (2). Letting $r' = \mathsf{ir}$, result (3) can be derived directly and result (4) is immediate. Result (5) is vacuous because $r = \mathsf{ir}$.
- SEQ-1 Let $e$ be $e_1; e_2$ and $e' = e'_1; e_2$. Inverting assumptions, 2, 3, and 4 ensures induction applies. Inductive results (1), (2), and (4) are the results (1), (2), and (4) we need. For result (3), activeplus$(e'_1; H'_{\mathsf{log}}; e'_0; r')$ (where $e_0$ is $e'_0; e_2$) lets us derive activeplus$((e'_1; e_2); H'_{\mathsf{log}}; (e'_0; e_2); r')$. Result (5) follows from induction much like in the Starting Inactive Lemma.
- SET-1, SET-2, REF-1, GET-1, APP-1, and APP-2 are analogous to SEQ-1.
- SEQ-V, ALLOC, BETA, SPAWN, SET-EAGER, GET-EAGER, ENTER ATOMIC, COMPLETE ROLLBACK, and COMMIT are all vacuous since we need to step from a state with $\bullet$ to a state with $\bullet$.

**Lemma 5.10 (Only Active Steps)**

1. *If* not-active$(e)$*, then there is no* $a'$*,* $H'$*,* $e'$*,* $H'_{\mathsf{log}}$*,* $T'$ *such that* $\bullet; H; e \to a'; H'; e'; T'; H'_{\mathsf{log}}$*.*
2. *If* not-active$(T)$*, then there is no* $a'$*,* $H'$*,* $T'$ *such that* $\bullet; H; T \to_{\mathsf{rb}} a'; H'; T'$*.*

**Proof** 1. By induction on the derivation of not-active$(e)$

2. By induction on the derivation of not-active$(T)$

**Lemma 5.11 (Heap Strengthening)**

1. *If* $a'; H'; e' \to a; H; e; T; H_{\mathsf{log}}$*,* $\Gamma'_1 \vdash H'_1 : \Gamma'_1$*,* $\Gamma'_1; \epsilon \vdash e' : \tau$*, and* $H'$ *extends* $H'_1$*, then there exists a* $H_1$ *such that* $a'; H'_1; e' \to a; H_1; e; T; H_{\mathsf{log}}$ *and* $H$ *extends* $H_1$*.*
2. *If* $a'; H'; T' \to_{\mathsf{rb}} a; H; T$*,* $\Gamma'_1 \vdash H'_1 : \Gamma'_1$*,* $\Gamma'_1; 0 \vdash T$*, and* $H'$ *extends* $H'_1$*, then there exists an* $H_1$ *such that* $a'; H'_1; T' \to_{\mathsf{rb}} a; H_1; T$ *and* $H$ *extends* $H_1$*.*

**Proof** 1. By induction on the derivation of $a'; H'; e' \to a; H; e; T; H_{\mathsf{log}}$, proceeding by cases on the last rule used:

- SEQ-1, SET-1, SET-2, REF-1, GET-1, APP-1, APP-2 are all by straightforward induction.
- SEQ-V, BETA, SPAWN, ENTER-ATOMIC, ENTER-ROLLBACK, COMPLETE ROLLBACK, and COMMIT are all immediate because $H' = H$ and its from does not affect applicability of the rule.
- SET-EAGER and GET-EAGER follow because $\Gamma'_1; 0 \vdash e' : \tau$ ensures the label accessed is in $\mathrm{Dom}(H'_1)$. $H_1$ is the restriction of $H$ to the domain of $H'_1$.
- ALLOC follows because if $l \notin \mathrm{Dom}(H')$, then $l \notin \mathrm{Dom}(H'_1)$. Let $H_1$ be the restriction of $H$ to the domain of $H'_1$.
- INATOMIC follows from induction. Note $T = \cdot$ and the log from the induction is not the resulting $H_{\mathsf{log}}$, which is $\cdot$.
- DO ROLLBACK follows because $\Gamma'_1; 0 \vdash e' : \tau$ ensures the label written to is in $\mathrm{Dom}(H'_1)$ (since the typing rule for inrollback requires the log's domain to be a subset of the heap's domain). We choose $H_1$ to be the restriction of $H$ to the domain of $H'_1$.

2. Follows immediately from part (1) since the top-level steps when some thread takes a step

**Lemma 5.12 (Top-Level Equivalence)** *If* $\vdash \circ; \cdot; e_{\mathrm{src}}$ *and* $\circ; \cdot; e_{\mathrm{src}} \to^*_{\mathsf{rb}} a; H; T$*, then there exist* $H_1$*,* $H_2$*,* $H_3$*,* $\Gamma_1$*,* $\Gamma_2$*,* $e_1$*,* $e_2$*, and* $k$ *such that:*

1. $H$ *extends* $H_1$*,* $\Gamma_1 \vdash H_1 : \Gamma_1$*, and* $\Gamma_1; 0 \vdash T$
2. *If* $a = \circ$*, then* $\circ; \cdot; e_{\mathrm{src}} \to^*_{\mathsf{no}} \circ; H_1; T$
3. *If* $a = \bullet$*,* activeplus$(e; H_{\mathsf{log}}; e_0; \mathsf{ia})$*, and* $T = T_A \parallel e \parallel T_B$*, then*
   (a) *If* $r = \mathsf{ia}$*, then* translate$(e; e_1)$ *and* $\circ; \cdot; e_{\mathrm{src}} \to^*_{\mathsf{no}} \bullet; H_1; T_A \parallel e_1 \parallel T_B$*.*
   (b) *The assumed step-sequence ends with* $k$ *steps where the state has the form* $\bullet; H'; T'$ *(i.e., "a is* $\bullet$*") as follows:*
      i. $\circ; \cdot; e_{\mathrm{src}} \to^*_{\mathsf{rb}} \circ; H_2; T_A \parallel e_0 \parallel T_B$
      ii. $\circ; H_2; T_A \parallel e_0 \parallel T_B \to_{\mathsf{rb}} \bullet; H_2; T_A \parallel e_2 \parallel T_B$
      iii. $\bullet; H_2; T_A \parallel e_2 \parallel T_B \to^k_{\mathsf{rb}} \bullet; H; T_A \parallel e \parallel T_B$
   (c) $\Gamma_2 \vdash H_2 : \Gamma_2$ *and* $\Gamma_2; 0 \vdash T_A \parallel e_0 \parallel T_B$
   (d) rollsbackto$(H; H_{\mathsf{log}}; H_3)$ *and* $H_3$ *extends* $H_2$

**Proof** (Notice that (4b), (4c), and (4d) are nonvacuous for both values of $r$.)

The proof is by induction on the number of steps taken for $\circ; \cdot; e_{\mathrm{src}} \to^*_{\mathsf{rb}} a; H; T$.

For 0 steps, $a = \circ$, $H = \cdot$, and $T = e_{\mathrm{src}}$. Result (1) follows by letting $H_1 = \cdot$ and $\Gamma_1 = \cdot$. Result (2) is immediate by taking 0 steps. Results (3) and (4) hold vacuously.

For $n > 0$ steps, there exist $a'$, $H'$, and $T'$ such that $\circ; \cdot; e_{\mathrm{src}} \to^*_{\mathsf{rb}} a'; H'; T'$ and $a'; H'; T' \to_{\mathsf{rb}} a; H; T$. So by induction there exist $H'_1$, $H'_2$, $H'_3$, $\Gamma'_1$, $\Gamma'_2$, $e'_1$, $e'_2$, and $k'$ such that:

1. $H'$ extends $H_1'$, $\Gamma_1' \vdash H_1' : \Gamma_1'$, and $\Gamma_1'; 0 \vdash T'$
2. If $a' = \circ$, then $\circ; \cdot; e_{\text{src}} \rightarrow^*_{\mathsf{no}} \circ; H_1'; T'$
3. If $a' = \bullet$, $T' = T_A' \parallel e' \parallel T_B'$, and activeplus$(e'; H_{\mathsf{log}}'; e_0'; r)$, then
   (a) If $r = \mathsf{ia}$, then translate$(e'; e_1')$ and $\circ; \cdot; e_{\text{src}} \rightarrow^*_{\mathsf{no}} \bullet; H_1'; T_A' \parallel e_1' \parallel T_B'$.
   (b) The assumed step-sequence ends with $k'$ steps where the state has the form $\bullet; H''; T''$ (i.e., "$a$ is $\bullet$") as follows:
      i. $\circ; \cdot; e_{\text{src}} \rightarrow^*_{\mathsf{rb}} \circ; H_2'; T_A' \parallel e_0' \parallel T_B'$
      ii. $\circ; H_2'; T_A' \parallel e_0' \parallel T_B' \rightarrow_{\mathsf{rb}} \bullet; H_2'; T_A' \parallel e_2' \parallel T_B'$
      iii. $\bullet; H_2'; T_A' \parallel e_2' \parallel T_B' \rightarrow^{k'}_{\mathsf{rb}} \bullet; H'; T_A' \parallel e' \parallel T_B'$
   (c) $\Gamma_2' \vdash H_2' : \Gamma_2'$ and $\Gamma_2'; 0 \vdash T_A' \parallel e_0' \parallel T_B'$
   (d) rollsbackto$(H'; H_{\mathsf{log}}'; H_3')$ and $H_3'$ extends $H_2'$

For result (1): $\Gamma_1' \vdash H_1' : \Gamma_1'$, $\Gamma_1'; 0 \vdash T'$, $a'; H'; T' \rightarrow_{\mathsf{rb}} a; H; T$, and the Heap Strengthening Lemma ensure there exists an $H_0$ such that $a'; H_1'; T' \rightarrow_{\mathsf{rb}} a; H_0; T$ and $H$ extends $H_0$. Therefore, by Type Safety there exists $\Gamma_0$ such that $\Gamma_0 \vdash H_0 : \Gamma_0$ and $\Gamma_0; 0 \vdash T$. If we choose $H_1 = H_0$ and $\Gamma_1 = \Gamma_0$, then result (1) is satisfied. However, in the case below where $a' = \bullet$, $a = \circ$, and $r = \mathsf{ir}$, we will choose a different $H_1$ and $\Gamma_1$, requiring a different justification of result (1).

We continue by cases on $a'$ and $a$, using from the argument above that $a'; H_1'; T' \rightarrow_{\mathsf{rb}} a; H_1; T$.

- If $a' = a = \circ$, then result (3) holds vacuously. For result (2), by inversion on $\circ; H_1'; T' \rightarrow_{\mathsf{rb}} \circ; H_1; T$ and $\Gamma_1'; 0 \vdash T'$, there exist $T_A$, $T_B$, $e'$, $e$, $\tau$, $T''$, and $H_{\mathsf{log}}$ such that $T' = T_A \parallel e' \parallel T_B$, $T = T_A \parallel e \parallel T_B \parallel T''$, $\Gamma_1'; 0 \vdash e' : \tau$, not-active$(e')$, and $\circ; H_1'; e' \rightarrow \circ; H_1; e; T''; H_{\mathsf{log}}$. So by the Starting Inactive Lemma, $\circ; H_1'; e' \rightarrow_{\mathsf{no}} \circ; H_1; e''; T'' H_{\mathsf{log}}$ where translate$(e; e'')$. But Type-Safety and $a = \circ$ ensure not-active$(e)$, so $e'' = e$. So we can derive $\circ; H_1'; T' \rightarrow_{\mathsf{no}} \circ; H_1; T$. Together with inductive result (2), this gives us result (2).
- If $a' = \circ$ and $a = \bullet$, then result (2) holds vacuously. For result (3), by inversion on $\circ; H_1'; T' \rightarrow_{\mathsf{rb}} \bullet; H_1; T$ and $\Gamma_1'; 0 \vdash T'$, there exist $T_A$, $T_B$, $e'$, $e$, $\tau$, $T''$, and $H_{\mathsf{log}}$ such that $T' = T_A \parallel e' \parallel T_B$, $T = T_A \parallel e \parallel T_B \parallel T''$, $\Gamma_1'; 0 \vdash e' : \tau$, not-active$(e')$, not-active$(T_A)$, not-active$(T_B)$, and $\circ; H_1'; e' \rightarrow \bullet; H_1; e; T''; H_{\mathsf{log}}$. So by the Preservation Lemma, active$(e)$. So by the Becoming Active Lemma, activeplus$(e; \cdot; e'; \mathsf{ia})$ and by the Activeplus-Determinism Lemma and not-active$(T_A \parallel T_B)$, it suffices to show results (a)–(d) where $H_{\mathsf{log}} = \cdot$ and $e_0 = e'$.
  (a) Follows from the Starting Inactive Equivalence Lemma and inductive result (2) (like in the $a = a' = \circ$ case).
  (b) Follows immediately letting $e_0 = e'$, $H_2 = H'$, $e_2 = e$, and $k = 0$.
  (c) Follows from Type-Safety (inverting $\vdash \bullet; H'; T'$), i.e., that the state type-checked before the step, letting $\Gamma_2 = \Gamma'$.
  (d) Follows from $H_{\mathsf{log}} = \cdot$ and $H_2 = H' = H$ by letting $H_3 = H_2$.
- If $a = \bullet$ and $a' = \circ$, then result (3) holds vacuously. For result (2), we use inductive result (3) and the Only Active Steps Lemma to determine only thread $e'$ steps. We also use, from Type Safety, $\Gamma_1' \vdash H_1' : \Gamma_1'$ and $\Gamma_1'; 0 \vdash e' : \tau$. So the conditions of the Becoming Inactive Lemma apply, providing:
  – If $r = \mathsf{ir}$, then $H_1 = H_1'$, $H_{\mathsf{log}}' = \cdot$, and $e = e_0'$.
  – If $r = \mathsf{ia}$, then $H_1 = H_1'$ and $\bullet; H_1'; e'' \rightarrow_{\mathsf{no}} \circ; H_1; e; T''$ where translate$(e'; e)$.

  We need to show $\circ; \cdot; e_{\text{src}} \rightarrow^*_{\mathsf{no}} \circ; H_1; T$ We proceed by cases on $r$:
  – If $r = \mathsf{ir}$, then $T$ is $T_A' \parallel e_0' \parallel T_B'$. Furthermore activeplus$(e'; \cdot; e_0'; \mathsf{ir})$, so inductive result (3d) ensures $H_3' = H'$ so $H'$ extends $H_2'$ (in other words, we have correctly rolled back to the beginning of the transaction). So letting $H_1 = H_2'$ and invoking inductive results (3bi) and (3bc), we establish results (1) and (2).[1]

[1]Note this is the case where we do *not* choose $H_1$ to be the $H_0$ from the Heap Strengthening argument. That choice would not work because we *cannot* show $\circ; \cdot; e_{\text{src}} \rightarrow^*_{\mathsf{no}} \circ; H_1; T$. Intuitively, allocated labels in the now-aborted transaction still exist. In terms of the lemma, because $r = \mathsf{ir}$, the inductive result does *not* provide $\circ; \cdot; e_{\text{src}} \rightarrow^*_{\mathsf{no}} \bullet; H_1'; T'$. Hence we use inductive result (3c) instead, a fact we preserved from the time we entered the transaction.

  - If $r = \mathsf{ia}$, then inductive result (3a) and $\bullet; H_1'; e'' \rightarrow_{\mathsf{no}} \circ; H_1; e; T''$ ensure $\circ; \cdot; e_{\mathrm{src}} \rightarrow^*_{\mathsf{no}} \circ; H_1; T$.

- If $a = a' = \bullet$, then result (2) holds vacuously. For result (3), we use inductive result (3) and the Only Active Steps Lemma to determine only thread $e'$ steps. We also use, from Type Safety, $\Gamma_1' \vdash H_1' : \Gamma_1'$ and $\Gamma_1'; 0 \vdash e' : \tau$. So the conditions of the Staying Active Lemma apply (twice, once with $H$ and once with $H_1$ since we need different results from the two), providing:
  - rollsbackto$(H; H_{\mathsf{log}}; H_3)$ for some $H_3$ extending $H_3'$
  - activeplus$(e; H_{\mathsf{log}}; e_0; r)$
  - If $r' = \mathsf{ir}$, then $r = \mathsf{ir}$, i.e., if $r = \mathsf{ia}$, then $r' = \mathsf{ia}$.
  - If $r = r' = \mathsf{ia}$, then $\bullet; H_1'; e_{no}' \rightarrow_{\mathsf{no}} \bullet; H_1; e_{no}; \cdot$ where translate$(e'; e_{no}')$ and translate$(e; e_{no})$.

  From inductive result (b), letting $k = k' + 1$ provides result (b). Inductive result (c) is result (c). From inductive result (d) we know $H_3'$ extends $H_2'$, so letting $H_2 = H_2'$ and noting heap extension is transitive means rollsbackto$(H; H_{\mathsf{log}}; H_3)$ for some $H_3$ extending $H_3'$ provides result (d). Finally, result (a) is vacuous if $r = \mathsf{ir}$, so assume $r = \mathsf{ia}$. Then $r' = \mathsf{ia}$ so inductive result (a) and $\bullet; H_1; e_{no} \rightarrow_{\mathsf{no}} \bullet; H_1'; e_{no}'; T$ provide $\circ; \cdot; e_{\mathrm{src}} \rightarrow^*_{\mathsf{no}} \bullet; H_1; T$.

**Theorem 5.13** (StrongUndo StrongBasic **Equivalence**) *If* $\vdash \circ; \cdot; e_{\mathrm{src}}$ *and* $\circ; \cdot; e_{\mathrm{src}} \rightarrow^*_{\mathsf{rb}} \circ; H; T$, *then there exists an* $H_1$ *such that* $H$ *extends* $H_1$ *and* $\circ; \cdot; e_{\mathrm{src}} \rightarrow^*_{\mathsf{no}} \circ; H_1; T$.

**Proof** This is a corollary of the Top-Level Equivalence Lemma, particularly results (1) and (2) (since our theorem assumes the resulting state includes $\circ$).

# 6 The WeakUndo language

One potential way to model a further weakening of software transactions is with eager update (transactions write to a globally visible version of $H$) and rollback (if a transaction aborts, it undoes those changes). In this model, a transaction can abort at any time.

## 6.1 Syntax

Syntax changes between WeakUndo and StrongBasic are minor. The changes in syntax for inatomic achieve the following goals:

1. We keep a log of the modifications made in the current transaction ($H_{\mathsf{log}}$) in case it aborts.
2. We use a nested value of $a$ to ensure that a transaction cannot rollback if the nested value of $e$ is executing a transaction. Once the nested transaction commits, then the parent transaction can roll back the changes made by the nested transaction as well as its own.
3. We keep the original expression $e_0$ to re-execute after rollback.

Similarly, we have added syntax for inrollback that keeps the remaining log entries to rollback as well as the original expression to re-execute once rollback completes.

$$\begin{array}{lll} e & ::= & c \mid l \mid x \mid e_1; e_2 \mid e_1 := e_2 \mid \text{ref } e \mid !e \mid \lambda x.e \mid e_1\ e_2 \\ & & \mid \mathsf{spawn}\, e \mid \mathsf{atomic}\ e \mid \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) \mid \mathsf{inrollback}(H_{\mathsf{log}}, e_0) \end{array}$$

## 6.2 Dynamic Evaluation

### 6.2.1 Whole Program Evaluation

Since we have removed the two internally parallel types of spawn, $e$ can only create a single $T$. In addition, introducing rollback has added the log ($H_{\mathsf{log}}$) to the evaluation form for $e$. The PROGRAM rule changes to accommodate the new evaluation form in two ways:

1. It is not necessary to keep $H_{\mathsf{log}}$ once we reach the top-level, so it is discarded.
2. We do the obvious thing with $T$.

$$\boxed{a; H; T \rightarrow a'; H'; T'}$$

PROGRAM

$$\frac{a; H; e \rightarrow a'; H'; e'; T; H_{\mathsf{log}}}{a; H; T_A \parallel e \parallel T_B \rightarrow a'; H'; T_A \parallel e' \parallel T_B \parallel T}$$

### 6.2.2 Expression Evaluation

Because we need to keep a log of heap changes during nested transactions, it must become part of the result of evaluation under eager update, yielding a slightly modified form. Many of the rules from Section 2.2 are only modified slightly. The SEQ-1, SET-1, SET-2, REF-1, GET-1, APP-1 and APP-2 rules all propagate the inductive value for $H_{\mathsf{log}}$ and $T$. (SEQ-1 is included in this section as an example). The SEQ-V, ALLOC and BETA rules all create an empty $H_{\mathsf{log}}$ and empty $T$. No other rules are adapted from the StrongNestedParallel language.

We have added new rules for heap accesses and modified or deleted rules to accommodate the syntax changes. The rules for WEAK-SET-EAGER and WEAK-GET-EAGER are included as a way of specifying how transactions modify and read $H$. We have also removed INATOMIC HELPER, SPAWN$_0$, SPAWN$_1$, and SPAWN$_2$ rules originally defined in StrongNestedParallel since internal parallelism and all supporting syntax has been deleted.

It is also of interest that this language includes more than one way to exit a transaction. We have removed the EXIT ATOMIC rule and introduced two rules that change $\bullet$ to $\circ$; COMMIT and COMPLETE ROLLBACK. As mentioned earlier, by including an internal value of $a$ in the inatomic syntax plays a critical role in the ENTER ROLLBACK rule because it prevents a transaction from entering rollback when there is an active nested transaction.

$\boxed{a; H; e \rightarrow a'; H'; e'; T; H_{\mathsf{log}}}$

SEQ-1
$$\frac{a; H; e_1 \rightarrow a'; H'; e_1'; T; H_{\mathsf{log}}}{a; H; e_1; e_2 \rightarrow a'; H'; e_1'; e_2; T; H_{\mathsf{log}}}$$

SPAWN
$$\frac{}{a; H; \mathsf{spawn}\, e \rightarrow a; H; 0; e; \cdot}$$

WEAK-SET-EAGER
$$\frac{}{a; H; l := v \rightarrow a; H, l \mapsto v; v; \cdot; l \mapsto H(l)}$$

WEAK-GET-EAGER
$$\frac{}{a; H; !l \rightarrow a; H; H(l); \cdot; \cdot}$$

ENTER ATOMIC
$$\frac{}{\circ; H; \mathsf{atomic}\ e \rightarrow \bullet; H; \mathsf{inatomic}(\circ, e, \cdot, e); \cdot; \cdot}$$

INATOMIC
$$\frac{a; H; e \rightarrow a'; H'; e'; \cdot; H'_{\mathsf{log}}}{\bullet; H; \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) \rightarrow \bullet; H'; \mathsf{inatomic}(a', e', H_{\mathsf{log}} H'_{\mathsf{log}}, e_0); \cdot; \cdot}$$

ENTER ROLLBACK
$$\frac{}{\bullet; H; \mathsf{inatomic}(\circ, e, H_{\mathsf{log}}, e_0) \rightarrow \bullet; H; \mathsf{inrollback}(H_{\mathsf{log}}, e_0); \cdot; \cdot}$$

DO ROLLBACK
$$\frac{}{\bullet; H; \mathsf{inrollback}(H_{\mathsf{log}}, l \mapsto v_{\mathsf{old}}, e_0) \rightarrow \bullet; H, l \mapsto v_{\mathsf{old}}; \mathsf{inrollback}(H_{\mathsf{log}}, e_0); \cdot; \cdot}$$

COMPLETE ROLLBACK
$$\frac{}{\bullet; H; \mathsf{inrollback}(\cdot, e_0) \rightarrow \circ; H; \mathsf{atomic}\ e_0; \cdot; \cdot}$$

COMMIT
$$\frac{}{\bullet; H; \mathsf{inatomic}(\circ, v, H_{\mathsf{log}}, e_0) \rightarrow \circ; H; v; \cdot; H_{\mathsf{log}}}$$

## 6.3 Typecheck $e$

In order to appropriately typecheck $e$, we use the T-CONST, T-VAR, T-SEQ, T-LAMBDA, T-APP, and T-ATOMIC rules from Section 2.3. These rules are unchanged.

As with all weak languages in the AtomsFamily, WeakUndo uses a type-and-effect system to partition $H$. The partition definition and rules are the same for Weak; T-SET-PARTITION, T-GET-PARTITION, and T-REF-PARTITION, T-LABEL-PARTITION are all repeated below; see Section 3.3 for intuition.

To handle syntax modifications we define T-SPAWN, T-INATOMIC and T-INROLLBACK and remove anything pertaining to syntax not in WeakUndo (e.g. other flavors of spawn that have been removed). removed, the rule for spawn defined for the The use of $\Gamma'$ in the T-INATOMIC rule is important because we would like $\Gamma'$ and in $H_{\mathsf{log}}$ to operate on the same set of labels (since this is what we require for $\Gamma$ and $H$). By stating that $\Gamma$ extends $\Gamma'$, we are enforcing that all labels defined in $H_{\mathsf{log}}$ are also defined in $H$. With eager update, even label allocation occurs directly in $H$.

$\boxed{\Gamma; \varepsilon \vdash e : \tau}$

T-SET-PARTITION
$$\frac{\Gamma; t \vdash e_1 : \mathrm{ref}_t \tau \qquad \Gamma; t \vdash e_2 : \tau}{\Gamma; t \vdash e_1 := e_2 : \tau}$$

T-GET-PARTITION
$$\frac{\Gamma; t \vdash e : \mathrm{ref}_t \tau}{\Gamma; t \vdash !e : \tau}$$

T-REF-PARTITION
$$\frac{\Gamma; \varepsilon \vdash e : \tau}{\Gamma; \varepsilon \vdash \mathrm{ref}\ e : \mathrm{ref}_t \tau}$$

T-LABEL-PARTITION
$$\frac{\Gamma(l) = (\tau, t)}{\Gamma; \varepsilon \vdash l : \mathrm{ref}_t \tau}$$

T-SPAWN
$$\frac{\Gamma; 0 \vdash e : \tau}{\Gamma; 0 \vdash \mathsf{spawn}\, e : \mathrm{int}}$$

T-INROLLBACK
$$\frac{\Gamma; 2 \vdash e_0 : \tau \qquad \Gamma \vdash H_{\mathsf{log}} : \Gamma' \qquad \Gamma \text{ extends } \Gamma' \qquad \mathsf{all2}(\Gamma') \qquad \text{not-active}(e_0)}{\Gamma; \varepsilon \vdash \mathsf{inrollback}(H_{\mathsf{log}}, e_0) : \tau}$$

T-INATOMIC
$$\frac{\text{correct-atomic}(a, e) \qquad \Gamma; 2 \vdash e_0 : \tau \qquad \begin{array}{c}\Gamma; 2 \vdash e : \tau \\ \text{not-active}(e_0)\end{array} \qquad \Gamma \vdash H_{\mathsf{log}} : \Gamma' \qquad \Gamma \text{ extends } \Gamma' \qquad \mathsf{all2}(\Gamma')}{\Gamma; \varepsilon \vdash \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) : \tau}$$

## 6.4 Other Typing Rules

Add a way to state that all labels in a context typecheck under $\varepsilon = 2$. (whenever $\mathsf{all2}(\Gamma)$ and $l \in Dom(\Gamma)$ then $\Gamma; \varepsilon \vdash l : \text{ref}_2\tau$)

$\boxed{\mathsf{all2}(\Gamma)}$

$$\frac{}{\mathsf{all2}(\cdot)} \qquad \frac{\mathsf{all2}(\Gamma')}{\mathsf{all2}(\Gamma', l = (\tau, 2))}$$

## 6.5 Activeness

### 6.5.1 Not-Active($e$)

Like in the StrongBasic language, we remove the not-active($e$) rules for deleted syntax ($\mathsf{spawn}_0$, $\mathsf{spawn}_1$, and $\mathsf{spawn}_2$). We add a rule for the new spawn syntax:

$\boxed{\text{not-active}(e)}$

$$\frac{\text{not-active}(e)}{\text{not-active}(\mathsf{spawn}\ e)}$$

### 6.5.2 Active($e$)

We update the active($e$) rule for inatomic and add a rule for inrollback.

$\boxed{\text{active}(e)}$

$$\frac{}{\text{active}(\mathsf{inrollback}(H_{\mathsf{log}}, e_0))} \qquad \frac{}{\text{active}(\mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0))}$$

### 6.5.3 Correct Atomic State of $T$

No Changes.

## 6.6 Type Safety

| Theorem/Lemma | Restated | Changes |
|---|---|---|
| Top-Level Progress | No | None |
| Single Thread Progress | See Lemma 6.1 below | See Proof below |
| Top-Level Preservation | No | None |
| Single Thread Preservation | See Lemma 6.2 below | See Proof below |
| Weakening Lemma | No | None |
| Canonical Forms Lemma | No | None |
| Substitution Lemma | No | T-SPAWN-0: Remove this case<br>T-SPAWN-1: Remove this case<br>T-SPAWN-2: Remove this case<br>T-SPAWN: This case is like the T-SEQ case.<br>T-INATOMIC: No changes<br>T-INROLLBACK: This case is like the T-INATOMIC case. |
| Values Effectless Lemma | No | None (see note in Sec. 3.3) |
| Variables not in $\Gamma'$ | No | None |
| Values Inactive | No | None |
| Effects Lemma | See Lemma 6.3 below | See Proof below |
| Heap Append Lemma | NEW See Lemma 6.4 below | See Proof below |

**Lemma 6.1 (Single Thread Progress)** *If $\Gamma \vdash H : \Gamma$, then each of the following must be true:*

1. *If $\Gamma; \varepsilon \vdash e : \tau$, and* active*(e) then $\exists e', a', H', T, H_{\mathsf{log}}$ such that $\bullet; H; e \rightarrow a'; H'; e'; T; H_{\mathsf{log}}$*
2. *If $\Gamma; \varepsilon \vdash e : \tau$, and* not-active*(e) then $e$ is a value or $\exists e', a', H', T, H_{\mathsf{log}}$ such that $\circ; H; e \rightarrow a'; H'; e'; T; H_{\mathsf{log}}$*
3. *If $\Gamma; \varepsilon \vdash T$ and correct-atomic(a, T) then $T$ is some $T_A \parallel e \parallel T_B$ such that $a; H; e \rightarrow a'; H'; e'; T'; H_{\mathsf{log}}$ or $T$ is all values.*

**Proof** of Lemma 6.1, updated cases only, organized as in Section 2.6. (No changes to proof of part 3).

- T-SPAWN-0, T-SPAWN-1, and T-SPAWN-2 are all removed.
- T-SPAWN: Add this case. $e = \mathsf{spawn}\ (e)$
  1. This case is vacuous because there is no way to derive active($e$).
  2. $e$ steps under the SPAWN rule.
- T-INATOMIC $e = \mathsf{inatomic}(a'', e_1, H_{\mathsf{log}}, e_0)$
  1. Inversion on the typing rule implies correct-atomic($a'', e_1$) and that $e_1$ type checks under $\varepsilon = 2$. There are three sub-cases to consider:
     (a) If $a'' = \bullet$, then it must be that active($e_1$). In this case, $e_1$ steps via INATOMIC. In this case, because $\Gamma; 2 \vdash \Gamma : e_1$ it must be that $\varepsilon \neq 0$, which by the effects lemma enforces that $T = \cdot$. Thus the entire inatomic expression can step.
     (b) If $a'' = \circ$ and $e_1 \neq v$, then there is a valid step for $e$ via ENTER ROLLBACK or by induction.
     (c) If $a'' = \circ$ and $e_1 = v$, then there is a valid step for $e$ via COMMIT.
  2. This case is vacuous because there is no way to derive not-active($e$).
- T-INROLLBACK $e = \mathsf{inrollback}(H_{\mathsf{log}}, e_0)$
  1. In this case, we can always derive active($e$). Inversion on the typing rule implies that $e_0$ type checks under $\varepsilon = 2$. There are two sub-cases to consider:
     (a) If $H_{\mathsf{log}} \neq \cdot$ then there is a valid step for $e$ under DO ROLLBACK.

(b) If $H_{\mathsf{log}} = \cdot$ then there is a valid step for $e$ under COMPLETE ROLLBACK.

2. This case is vacuous because there is no way to derive not-active($e$).

**Lemma 6.2 (Single Thread Preservation)** *If* $a; H; e \to a'; H'; e'; T; H_{\mathsf{log}}$ *and* $\Gamma; \varepsilon \vdash e : \tau$ *and* $\Gamma \vdash H : \Gamma$ *and one of the following:*

1. $a = \circ$ *and* not-active*(e)*
2. $a = \bullet$ *and* not-active*(e)*
3. $a = \bullet$ *and* active*(e)*

*then* $\exists \Gamma', \Gamma''$ *with* $\Gamma'$ *extending* $\Gamma$ *and* $\Gamma''$ *such that:*

1. $\Gamma'; \varepsilon \vdash e' : \tau$
2. $\Gamma' \vdash H' : \Gamma'$
3. $\Gamma' \vdash H_{\mathsf{log}} : \Gamma''$ *and if* $\varepsilon = 2$ *then* $\mathsf{all2}(\Gamma'')$
4. not-active*(T)*
5. $\Gamma'; 0 \vdash T$
6. *All of the following, although all but one case will be vacuous:*
   - *If* $a = a'$ *and* active*(e) then* active*(e')*
   - *If* $a = a'$ *and* not-active*(e) then* not-active*(e')*
   - *If* $a = \circ$ *and* $a' = \bullet$ *then* not-active*(e) and* active*(e')*
   - *If* $a = \bullet$ *and* $a' = \circ$ *then* active*(e) and* not-active*(e')*

**Proof** by induction on the typing derivation of $e$ by cases on the final rule used in the derivation. Because we have added the conclusion that $\Gamma' \vdash H_{\mathsf{log}} : \Gamma''$ with $\Gamma'$ extending $\Gamma''$, we extend all unlisted cases with the following statements: If $H_{\mathsf{log}} = \cdot$ then we have $\Gamma' \vdash \cdot : \cdot$ where any $\Gamma'$ extends $\cdot$. Also, if $H_{\mathsf{log}} \neq \cdot$ then we know that the desired $\Gamma''$ and $\Gamma'$ are obtained through induction. Only the interesting or relevant changes are listed here.

- T-SET-PARTITION $e = e_1 := e_2$. There are three ways $e$ could have become $e'$. Only one of these is different for WeakUndo.
  - Under WEAK-SET-EAGER $e = l := v$, and $e' = v$, and $T = \cdot$, $H' = H[l \mapsto v]$, and $H_{\mathsf{log}} = l \mapsto H(l)$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = l \mapsto (\tau, t)$.
    1. Inversion on the typing rule gives $\Gamma; t \vdash l : \mathrm{ref}_t\tau$ and $\Gamma; t \vdash v : \tau$. Since $\Gamma' = \Gamma$, it must be that $\Gamma'; t \vdash l : \mathrm{ref}_t\tau$ and $\Gamma'; t \vdash v : \tau$. Since $e' = v$, we have $\Gamma'; t \vdash e' : \tau$.
    2. Since $H' = H[l \mapsto v]$, then $H'(l) = v$. By the weakening lemma and the chosen $\Gamma$, $\Gamma'; t \vdash v : \tau$ and $\Gamma'(l) = (\tau, t)$. Thus, $\Gamma' \vdash H' : \Gamma'$.
    3. By assumption, $\Gamma; \varepsilon \vdash v : \tau$, and $\Gamma(l) = (\tau, t)$. Because $\Gamma' = \Gamma$, we also have that $\Gamma'(l) = (\tau, t)$. $\Gamma''(l) = (\tau, t)$ by definition. Since $H_{\mathsf{log}} = l \mapsto H(l)$, it must be that $\Gamma' \vdash H_{\mathsf{log}} : \Gamma''$. Also, *If* $\varepsilon = 2$ then $\Gamma'; 2 \vdash e' : \tau$ so $\Gamma'(l) = (\tau, 2)$ and because $\Gamma'$ extends $\Gamma''$ and using our choice for $\Gamma'''$, we know that $\Gamma'' = l{:}(\tau, 2)$, and therefore $\mathsf{all2}(\Gamma'')$.
    4. not-active($\cdot$)
    5. $\Gamma; 0 \vdash \cdot$ is always true.
    6. Because $e$ stepped under WEAK-SET-EAGER, $a = a'$, and not-active($e$) is derived from not-active($l$) and not-active($v$). Since $e' = v$, we know not-active($e'$). There is no way to derive active($e$), so that case is vacuous.
- T-GET-PARTITION $e = !e_0$ There are two evaluation rules that could take $e$ to $e'$. Only one of these is different for WeakUndo.
  - Under WEAK-GET-EAGER $e = !l$, $e' = H(l) = v$, $T = \cdot$, and $H_{\mathsf{log}} = \cdot$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = \cdot$.
    1. Inversion on the typing rule gives $\Gamma; t \vdash l : \mathrm{ref}_t\tau$, and thus Since $\Gamma' = \Gamma$, it must be that $\Gamma'; t \vdash v : \tau$.
    2. Since $H' = H$, and $\Gamma \vdash H : \Gamma$ by assumption, and $\Gamma' = \Gamma$, it must be that $\Gamma' \vdash H' : \Gamma'$.
    3. Since $H_{\mathsf{log}} = \cdot$, it must be that $\Gamma' \vdash H_{\mathsf{log}} : \cdot$ and $\mathsf{all2}(\cdot)$ when $\varepsilon = 2$.

4. not-active$(\cdot)$
5. $\Gamma; 0 \vdash \cdot$ is always true.
6. Because $e$ stepped under WEAK-GET-EAGER, $a = a'$, and not-active$(!l)$ is derived from not-active$(l)$. Since $e' = v$, we know not-active$(e')$. There is no way to derive active$(e)$, so that case is vacuous.

- T-SPAWN $e = \mathsf{spawn}\, e_0$. Thus, $\Gamma; \varepsilon \vdash e : \text{int}$. There is only one evaluation rule for $\mathsf{spawn}\, e_0$, SPAWN. After $e$ steps to $e'$ using this rule, $e' = 0$, $H' = H$, $T = e_0$, and $H_{\mathsf{log}} = \cdot$. Pick $\Gamma' = \Gamma$, and $\Gamma'' = \cdot$.
  1. T-CONST provides $\Gamma'; \varepsilon \vdash 0 : \text{int}$.
  2. $\Gamma' = \Gamma$ and $H' = H$, and by assumption, $\Gamma \vdash H : \Gamma$. Thus $\Gamma' \vdash H' : \Gamma'$.
  3. $\Gamma' \vdash \cdot : \cdot$ and $\mathsf{all2}(\cdot)$ when $\varepsilon = 2$
  4. By assumption, either active$(e)$ or not-active$(e)$. There is no way to derive active$(e)$ for spawn expressions, so it must be that not-active$(e)$ and thus not-active$(e_0)$. Although $e_0$ was added to $T$, it still must be that not-active$(e_0)$, and therefore not-active$(T)$.
  5. By inversion on the typing rule T-SPAWN, it is also the case that $\Gamma; 0 \vdash e_0 : \tau$. Because $T = e_0$, and using the weakening lemma, it must be that $\Gamma'; 0 \vdash T$.
  6. The SPAWN rule enforces that $a = a'$. If not-active$(e)$ then it must be shown that not-active$(e')$. Since not-active$(0)$ requires no assumption and $e' = 0$, this is trivial. The case for active$(e)$ is vacuous because there is no way to derive active$(\mathsf{spawn}\, e_0)$.
- T-SPAWN-0 Remove this case.
- T-SPAWN-1 Remove this case.
- T-SPAWN-2 Remove this case.
- T-ATOMIC $e = \mathsf{atomic}\ e_0$. If $e$ steps, it must be via the evaluation rule ENTER ATOMIC, giving $e' = \mathsf{inatomic}(a'', e_0, \cdot, e_0)$, $T = \cdot$, $H' = H$, and $H_{\mathsf{log}} = \cdot$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = \cdot$.
  1. $\Gamma; 2 \vdash e_0 : \tau$ is true by inversion on the typing rule, and by induction $\Gamma'; 2 \vdash e_0 : \tau$ since $\Gamma'$ extends $\Gamma$.
  2. $H' = H$ is unchanged, and $\Gamma \vdash H : \Gamma$ by assumption, so $\Gamma' \vdash H' : \Gamma'$.
  3. $\Gamma' \vdash \cdot : \cdot$ and $\mathsf{all2}(\cdot)$ when $\varepsilon = 2$
  4. not-active$(\cdot)$
  5. $\Gamma'; 0 \vdash \cdot$ is always true.
  6. Here, $a = \circ$ and $a' = \bullet$. The correct atomicity is preserved because we can derive not-active$(e)$ from assumption 1 when $a = \circ$. active$(e')$ requires no assumptions.
- T-INATOMIC $e = \mathsf{inatomic}(a'', e_1, H'_{\mathsf{log}}, e_0)$ By inversion: $\Gamma; 2 \vdash e_1 : \tau$, correct-atomic$(a, e_1)$, $\Gamma; 2 \vdash e_0 : \tau$, not-active$(e_0)$, and $\Gamma \vdash H'_{\mathsf{log}} : \Gamma'''$. Proof is itemized by cases on the possible evaluation. If $e$ steps...
  - Under INATOMIC, then $e' = \mathsf{inatomic}(a''', e'_1, H_{\mathsf{log}} H''_{\mathsf{log}}, e_0)$, $H_{\mathsf{log}} = \cdot$, and $T = \cdot$. Pick $\Gamma'' = \cdot$, and $\Gamma'$ is obtained through induction.
    1. By induction, $\Gamma'; 2 \vdash e'_1 : \tau$, and correct-atomic$(a''', e'_1)$ are true. Also by induction, there is some $\Gamma''_{\mathsf{nested}}$ such that $\Gamma' \vdash H''_{\mathsf{log}} : \Gamma''_{\mathsf{nested}}$ and $\Gamma'$ extends $\Gamma''_{\mathsf{nested}}$. By inversion we already knew that $\Gamma \vdash H'_{\mathsf{log}} : \Gamma'''$, with $\mathsf{all2}(\Gamma''')$, and since $\Gamma'$ extends $\Gamma$ we know that $\Gamma' \vdash H'_{\mathsf{log}} : \Gamma'''$. Dy definition, $\Gamma$ extends $\Gamma'''$ and $\Gamma'$ extends $\Gamma$, thus $\Gamma'$ extends $\Gamma'''$. Lemma 6.4 provides that $\Gamma' \vdash H_{\mathsf{log}} H''_{\mathsf{log}} : \Gamma''' \Gamma''_{\mathsf{nested}}$ with $\Gamma'$ extends $\Gamma''' \Gamma''_{\mathsf{nested}}$. By induction and the weakening lemma, $\Gamma; 2 \vdash e_0 : \tau$ and not-active$(e_0)$ are also both true. Which gives everything we need to derive $\Gamma; 2 \vdash e' : \tau$
    2. $\Gamma' \vdash H' : \Gamma'$ is true by induction.
    3. $\Gamma' \vdash \cdot : \cdot$ and $\mathsf{all2}(\cdot)$ when $\varepsilon = 2$
    4. not-active$(\cdot)$ is always true.
    5. $\Gamma'; 0 \vdash \cdot$ is always true.
    6. Here, $a = a' = \bullet$, and active$(e)$ requires no assumptions. Similarly, active$(e')$ requires no assumptions.

– Under ENTER ROLLBACK, then $e' = \mathsf{inrollback}(H'_{\mathsf{log}}, e_0)$, $H_{\mathsf{log}} = \cdot$ and $T = \cdot$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = \cdot$.

1. $\Gamma \vdash H'_{\mathsf{log}} : \Gamma'''$ is true by inversion on the typing rule, and since $\Gamma' = \Gamma$, $\Gamma' \vdash H'_{\mathsf{log}} : \Gamma'''$ must also be true. $\Gamma; 2 \vdash e_0 : \tau$ and not-active($e_0$) are also true by inversion on the typing rule, which (when combined with the fact that $\Gamma' = \Gamma$) gives $\Gamma'; 2 \vdash e' : \tau$.
2. Since $\Gamma' = \Gamma$ and $H' = H$, and $\Gamma \vdash H : \Gamma$ is true by assumption, it must also be that $\Gamma' \vdash H' : \Gamma'$.
3. $\Gamma' \vdash \cdot : \cdot$ and $\mathsf{all2}(\cdot)$ when $\varepsilon = 2$
4. not-active($\cdot$) is always true.
5. $\Gamma'; 0 \vdash \cdot$ is always true.
6. Here, $a = a' = \bullet$, and active($e$) requires no assumptions. Similarly, active($e'$) requires no assumptions.

– Under COMMIT, then $e' = v$, $H_{\mathsf{log}} = H'_{\mathsf{log}}$ and $T = \cdot$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = \Gamma'''$

1. By inversion on the typing rule, (and since to take this step is must be that $e_1 = v$) we know that $\Gamma; 2 \vdash v : \tau$. Because $\Gamma' = \Gamma$ we also have $\Gamma'; 2 \vdash v : \tau$. Since $e' = v$, it must be that $\Gamma'; 2 \vdash e' : \tau$.
2. Since $\Gamma' = \Gamma$ and $H' = H$, and $\Gamma \vdash H : \Gamma$ is true by assumption, it must also be that $\Gamma' \vdash H' : \Gamma'$.
3. We know that $\Gamma \vdash H'_{\mathsf{log}} : \Gamma'''$ and $\mathsf{all2}(\Gamma''')$ are true by inversion on the typing rule. S ince $\Gamma' = \Gamma$ and $\Gamma'' = \Gamma'''$ we have $\Gamma' \vdash H'_{\mathsf{log}} : \Gamma'''$ and $\mathsf{all2}(\Gamma'')$.
4. not-active($\cdot$) is always true.
5. $\Gamma'; 0 \vdash \cdot$ is always true.
6. Here, $a = \bullet$ and $a' = \circ$, and the Values Inactive Lemma provides not-active($e'$).

- T-INROLLBACK $e = \mathsf{inrollback}(H'_{\mathsf{log}}, e_0)$ This typing rule gives us each of the following: $\Gamma; 2 \vdash e_0 : \tau$, not-active($e_0$), and $\Gamma \vdash H'_{\mathsf{log}} : \Gamma'''$. Proof is by cases on the possible evaluation. If $e$ steps...

– Under DO ROLLBACK, then $e' = \mathsf{inrollback}(H''_{\mathsf{log}}, e_0)$, where $H'_{\mathsf{log}} = H''_{\mathsf{log}}, l \mapsto v_{\mathsf{old}}$, $T = \cdot$, and $H_{\mathsf{log}} = \cdot$, and $H' = H, l \mapsto v_{\mathsf{old}}$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = \cdot$.

1. By inversion, we know $\Gamma; 2 \vdash e_0 : \tau$ and not-active($e_0$). Since $\Gamma \vdash H''_{\mathsf{log}}, l \mapsto v_{\mathsf{old}} : \Gamma'''$, then $\Gamma \vdash H''_{\mathsf{log}} : \Gamma'''$. Because $\Gamma' = \Gamma$, we have $\Gamma'; 2 \vdash e_0 : \tau$ and $\Gamma' \vdash H''_{\mathsf{log}} : \Gamma''$. Thus, $\Gamma'; 2 \vdash e' : \tau$ is derived.
2. Since $\Gamma' = \Gamma$ and $H' = H, l \mapsto v_{\mathsf{old}}$, and $\Gamma \vdash H : \Gamma$ is true by assumption, it must also be that $\Gamma' \vdash H : \Gamma'$. We also know that $\Gamma \vdash H''_{\mathsf{log}}, l \mapsto v_{\mathsf{old}} : \Gamma$ by inversion on the typing rule. Since $l \in \mathit{Dom}(H)$, we know that $\Gamma(l) = (\tau, \varepsilon)$, which when combined with the weakening lemma gives $\Gamma' \vdash l \mapsto v_{\mathsf{old}} : \Gamma'$. Thus $\Gamma' \vdash H' : \Gamma'$ must be true.
3. $\Gamma' \vdash \cdot : \cdot$ and $\mathsf{all2}(\cdot)$ when $\varepsilon = 2$
4. not-active($\cdot$) is always true.
5. $\Gamma'; 0 \vdash \cdot$ is always true.
6. Here, $a = a' = \bullet$. Fortunately, both active($e$) and active($e'$) are always true.

– Under COMPLETE ROLLBACK, then $e' = \mathsf{atomic}\ e_0$, $T = \cdot$, $H_{\mathsf{log}} = \cdot$, and $H' = H$. Pick $\Gamma' = \Gamma$ and $\Gamma'' = \cdot$.

1. By inversion on the typing judgment and choice of $\Gamma'$, we know that $\Gamma'; 2 \vdash e_0 : \tau$ and therefore $\Gamma'; \varepsilon \vdash e' : \tau$ must be true.
2. Since $\Gamma' = \Gamma$ and $H' = H$, and $\Gamma \vdash H : \Gamma$ is true by assumption, it must also be that $\Gamma' \vdash H' : \Gamma'$.
3. $\Gamma' \vdash \cdot : \cdot$ and $\mathsf{all2}(\cdot)$.
4. not-active($\cdot$) is always true.
5. $\Gamma'; 0 \vdash \cdot$ is always true.
6. Here, $a = \bullet$ and $a' = \circ$. Since active($e$) is always true, we need not-active($e_0$) (which is true by inversion on the typing rule) to derive not-active($e'$).

**Lemma 6.3 (Effects Lemma)** *Suppose* $\Gamma; \varepsilon \vdash e : \tau$ *and* $a; H; e \to a'; H'; e'; T; H_{\mathsf{log}}$. *If* $\varepsilon \neq 0$ *then* $T = \cdot$.

**Proof** of updated cases.

- T-SPAWN $e = \mathsf{spawn}\, e_1$. It must be that $\varepsilon = 0$, so this is vacuous.
- T-SPAWN-0, T-SPAWN-1, or T-SPAWN-2: Remove these cases.
- T-INATOMIC There are three possible sub-cases:
  1. Assume $e_1$ is not a value, and the chosen evaluation rule was INATOMIC. INATOMIC always creates $T = \cdot$.
  2. Assume $e_1$ is some $v$. $e$ steps under COMMIT, which always creates $T = \cdot$.
  3. Assume that the chosen evaluation rule was ENTER ROLLBACK (independent of $e_1$). ENTER ROLLBACK always creates $T = \cdot$.
- T-INROLLBACK has two subcases.
  1. Assume $H_{\mathsf{log}} \neq \cdot$, then the chosen evaluation rule was DO ROLLBACK, which always creates $T = \cdot$.
  2. Assume $H_{\mathsf{log}} = \cdot$, then the chosen evaluation rule was COMPLETE ROLLBACK, which always creates $T = \cdot$.

**Lemma 6.4 (Heap Append)** *If*

1. $\Gamma \vdash H_A : \Gamma_A$ *and* $\Gamma$ *extends* $\Gamma_A$
2. $\Gamma \vdash H_B : \Gamma_B$ *and* $\Gamma$ *extends* $\Gamma_B$

*Then* $\Gamma \vdash H_A H_B : \Gamma_A \Gamma_B$ *and* $\Gamma$ *extends* $\Gamma_A \Gamma_B$

**Proof** of Lemma 6.4:
Since $\Gamma$ extends $\Gamma_A$ and $\Gamma$ extends $\Gamma_B$, we know that for every $l \in \mathit{Dom}(\Gamma_A)$ that $l \in \mathit{Dom}(\Gamma)$, and for every $l \in \mathit{Dom}(\Gamma_B)$ that $l \in \mathit{Dom}(\Gamma)$, and as a result everywhere some label occurs in $\Gamma_A$ or $\Gamma_B$ then it has the same type as in $\Gamma$ and, where applicable, $\Gamma_B$ or $\Gamma_A$, respectively. Thus, there are no conflicts between $\Gamma_A$ and $\Gamma_B$ and $\Gamma$ extends $\Gamma_A \Gamma_B$. By trivial induction on the derivation of $\Gamma \vdash H : \Gamma$, it must also be true that $\Gamma \vdash H_A H_B : \Gamma_A \Gamma_B$.

## 6.7 Equivalence

In this section, we use $\rightarrow_s$ to refer to evaluation in StrongUndo and $\rightarrow_w$ to refer to evaluation in WeakUndo.

**Theorem 6.5 (Weak and Strong Rollback Equivalence)** *If* $\vdash \circ; \cdot; e$ *(using the type system from Section 6.3), then* $\circ; \cdot; e \rightarrow_s^n a; H; T$ *iff* $\circ; \cdot; e \rightarrow_w^n a; H; T$.

**Proof** There are two directions to prove for the iff statement:

- "strong implies weak": Every trace using the strong evaluation rules *is* an equivalent trace under the weak evaluation rules. This is because the weak rules allow a strict super set of traces.
- "weak implies strong": Follows as a corollary to Lemma 6.13 (the first conclusion of that lemma is what we need).

**Lemma 6.6 (Top-Level Reordering)** *(*revisited for WeakUndo and StrongUndo *). Suppose all of the following hold:*

1. $\Gamma \vdash H_0 : \Gamma$
2. $\Gamma; 0 \vdash e_A : \tau_A$
3. $\Gamma; 0 \vdash e_B : \tau_B$
4. active$(e_A)$
5. not-active$(e_B)$
6. $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_s \bullet; H_1; e'_A \parallel e_B \parallel T \rightarrow_w \bullet; H_2; e'_A \parallel e'_B \parallel T'$

*Then there exists some* $H'_1$ *such that* $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_w \bullet; H'_1; e_A \parallel e'_B \parallel T' \rightarrow_s \bullet; H_2; e'_A \parallel e'_B \parallel T'$.

**Top-level Reordering** Proof by induction on the form of $e_A$, organized by cases on the step taken by $e_A$. (*cases with minor changes or no changes are omitted for brevity. Similarly, note that the organization in this case is slightly different, but the induction is the same as Lemma 3.2.*)

- $e_A = \mathsf{spawn}_i e, 0 \leq i \leq 2$. *remove these cases*
- $e_A = \mathsf{spawn}\, e$. There is no way to derive that $e_A$ is active. This case is vacuous.
- $e_A = \mathsf{inatomic}(a'', e_1, H_{\mathsf{log}}, e_0)$. There are three ways in which $e_A$ could step:
  1. $e_A$ steps under INATOMIC. To apply Lemma 6.7, we need to show that each precondition applies:
     (a) $\Gamma \vdash H_0 : \Gamma$ by assumption.
     (b) By inversion on the typing rule for $e_A$, we know $\Gamma; 2 \vdash e_1 : \tau_A$.
     (c) $\Gamma; 0 \vdash e_B : \tau_B$ by assumption
     (d) not-active($e_B$) by assumption
     (e) Since $e_A$ stepped under the INATOMIC rule, it must be the case that $e_1$ is not a value, and that $e_1$ steps in the following manner to evaluate $e_A$: $a''; H_0; e_1 \rightarrow_s \;\; a'''; H_1; e_1'; \cdot; H_{\mathsf{log}A}$ By assumption, we know the following step was used to evaluate $e_B$: $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e_B'; T_B; H_{\mathsf{log}_B}$

     We can apply Lemma 6.7 to obtain $H_1'$.
  2. If $e_A$ stepped using the COMMIT rule, then $a = \bullet$ and $a' = \circ$. Since this violates the assumption that $a = a' = \bullet$, this case is vacuous.
     (a) not-active($e_B$) by assumption
     (b) Since $e_A$ stepped under the INATOMIC rule, it must be the case that $e_1$ is not a value, and that $e_1$ steps in the following manner to evaluate $e_A$: $a''; H_0; e_1 \rightarrow_s \;\; a'''; H_1; e_1'; \cdot; H_{\mathsf{log}A}$ By assumption, we know the following step was used to evaluate $e_B$: $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e_B'; T_B; H_{\mathsf{log}_B}$

     We can apply Lemma 6.7 with $e_1$ and $e_B$ to obtain $H_1'$.
  3. If $e_A$ stepped using the COMMIT rule, then $a = \bullet$ and $a' = \circ$. Since this violates the assumption that $a = a' = \bullet$, this case is vacuous.
  4. If $e_A$ stepped using the ENTER ROLLBACK rule, then $e_A$ does not change the heap on this step. Thus, $H_1 = H_0$ and $e_B$ was already using $H_0$ to step. Thus, $e_B$ can step as it did before. $e_A$ can step the same way under any $H$, so $e_A$ is similarly unaffected by reordering.
- $e_A = \mathsf{inrollback}(H_{\mathsf{log}}, e_0)$. There are two cases:
  1. If $e_A$ stepped using the DO ROLLBACK rule, (in which case $H_{\mathsf{log}} \neq \cdot$), we would like to apply Lemma 6.11 to obtain $H_1'$. Fist make sure every precondition applies:
     (a) $\Gamma \vdash H_0 : \Gamma$ is true by assumption.
     (b) $\Gamma; \varepsilon \vdash e_A : \tau_A$ is true by assumption. (In this case, $\varepsilon = 0$).
     (c) $\Gamma; 0 \vdash e_B : \tau_B$ is true by assumption.
     (d) not-active($e_B$) is true by assumption.
     (e) $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_s \;\; \bullet; H_1; e_A' \parallel e_B \parallel T$ and $\bullet; H_1; e_A' \parallel e_B \parallel T \rightarrow_w \;\; \bullet; H_2; e_A' \parallel e_B' \parallel T'$ are true by assumption.

     $H_1'$ is obtained by applying Lemma 6.11.
  2. If $e_A$ stepped using the COMPLETE ROLLBACK rule, then $a = \bullet$ and $a' = \circ$. Since this violates the assumption that $a = a' = \bullet$, this case is vacuous.

**Lemma 6.7 (Nested Reordering)** *(*revisited for WeakUndo and StrongUndo *). Suppose all the following hold:*

*1.* $\Gamma \vdash H_0 : \Gamma$

*2.* $\Gamma; 2 \vdash e_A : \tau_A$

*3.* $\Gamma; 0 \vdash e_B : \tau_B$

*4.* not-active*(*$e_B$*)*

*5. $a; H_0; e_A \rightarrow_s a'; H_1; e'_A; \cdot; H_{\mathsf{log}A}$ and $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$*

*Then there exists some $H'_1$ such that:*

*1. $\bullet; H_0; e_B \rightarrow_w \bullet; H'_1; e'_B; T_B; H_{\mathsf{log}_B}$ (i.e. $e_B$ steps under $H_0$ with no changes)*

*2. $a; H'_1; e_A \rightarrow_s a'; H_2; e'_A; \cdot; H_{\mathsf{log}A}$ (i.e. $e_A$ steps under $H'_1$ with no changes)*

**Proof** of Lemma 6.7 organized by cases on $e_A$. Only relevant changes are listed here.

- $e_A = \mathsf{spawn}_i e$, $0 \leq i \leq 2$. *remove these cases*
- $e_A = \mathsf{spawn}\, e$ Here, $e_A$ cannot typecheck under $\varepsilon = 2$. This case is vacuous.
- $e_A = e_1 := e_2$ Only one case needs to be revisited, when $e_A = l_A := v_A$. To apply Lemma 6.8, we must show that each precondition holds:
  1. $\Gamma \vdash H_0 : \Gamma$ is true by assumption.
  2. $H_1 = H_0, l_A \mapsto v_A$ is true by the only evaluation step available to $e_A$, STRONG-SET.
  3. $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$ is true by assumption.
  4. $\Gamma; 2 \vdash e_A : \tau_A$ is true by assumption.
  5. $\Gamma; 0 \vdash e_B : \tau_B$ is true by assumption.
  6. not-active($e_B$) is true by assumption.

  Applying Lemma 6.8 provides a valid $H'_1$.
- $e_A = !e_1$ Only one case needs to be revisited, when $e_A = !l_A$. To apply Lemma 6.9 we show that precondition holds:
  1. $\Gamma \vdash H_0 : \Gamma$ is true by assumption.
  2. $H_1 = H_0$ is true by the only evaluation step available to $e_A$, STRONG-GET. Also, $H_0(l_A) = v_A$
  3. $\bullet; H_0, l_A \mapsto v_A; e_B \rightarrow_w \bullet; H_2; e'_B; T; H_{\mathsf{log}_B}$ is true by assumption.
  4. $\Gamma; 2 \vdash e_A : \tau_A$ is true by assumption.
  5. $\Gamma; 0 \vdash e_B : \tau_B$ is true by assumption.
  6. not-active($e_B$) is true by assumption.

  Applying Lemma 6.9 provides a valid $H'_1$.
- $e_A = \mathsf{inatomic}(a'', e_1, H_{\mathsf{log}A}, e_0)$. There are three subcases:
  1. If $e_A$ steps under INATOMIC then $e_1$ is not a value, and so we use induction to obtain $H'_1$.
  2. If $e_A$ steps under COMMIT then $e_1 = v$, in which case we know that $H_1 = H_0$, $T_A = \cdot$, and $H_{\mathsf{log}A} = H_{\mathsf{log}}$. In this case, $e_B$ was already stepping using $H_0$, and so is not affected by reordering. (i.e. $e_B = e'_B$, $T_B = T_B$ as before, $H'_1 = H_2$ and $H_{\mathsf{log}_B} = H_{\mathsf{log}_B}$ as before. Similarly, $e_A$ steps under $H'_1$ and does not touch the heap, so $H_2$ is the same as before, and we know that $H_{\mathsf{log}A} = H_{\mathsf{log}}$ is the same as before.
  3. If $e_A$ steps under ENTER ROLLBACK then we know that $H_1 = H_0$, $T_A = \cdot$, and $H_{\mathsf{log}A} = \cdot$. In this case, $e_B$ was already stepping using $H_0$, and so is not affected by reordering. (i.e. $e_B = e'_B$, $T_B = T_B$ as before, $H'_1 = H_2$ and $H_{\mathsf{log}_B} = H_{\mathsf{log}_B}$ as before. Similarly, $e_A$ steps under $H'_1$ and does not touch the heap, so $H_2$ is the same as before, and we know that $H_{\mathsf{log}A} = \cdot$ and $T_A = \cdot$ are the same as before.
- $e_A = \mathsf{inrollback}(H_{\mathsf{log}}, e_0)$. There are two subcases:
  1. If $e_A$ steps under COMPLETE ROLLBACK then we know that $H_{\mathsf{log}} = \cdot$ and that $H_1 = H_0$, and that $H_{\mathsf{log}A} = \cdot$ and $T = \cdot$. In this case, $e_B$ was already stepping using $H_0$, and so is not affected by reordering. (i.e. $e_B = e'_B$, $T_B = T_B$ as before, $H'_1 = H_2$ and $H_{\mathsf{log}_B} = H_{\mathsf{log}_B}$ as before. Similarly, $e_A$ steps under $H'_1$ and does not touch the heap, so $H_2$ is the same as before, and we know that $H_{\mathsf{log}A} = H_{\mathsf{log}}$ is the same as before.
  2. If $e_A$ steps under DO ROLLBACK we know that $e_A = \mathsf{inrollback}(H'_{\mathsf{log}}, l_A \mapsto v_A, e_0)$. To apply Lemma 6.10 we first show that each precondition applies:
     (a) $\Gamma \vdash H_0 : \Gamma$ is true by assumption.
     (b) $\Gamma; 2 \vdash e_A : \tau'_A$ is true by assumption.
     (c) $\Gamma; 0 \vdash e_B : \tau_B$ is true by assumption.

(d) not-active($e_B$) is true by assumption.

(e) $\bullet; H_0; e_A \rightarrow_s \bullet; H_1; \mathsf{inrollback}(H'_{\mathsf{log}}, e_0); \cdot; H_{\mathsf{log}A}$ and $\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$ are both true by assumption.

**Lemma 6.8 (Independent Write Under Partition)** *Let $e_A = l_A := v_A$, so under the strong semantics $a; H_0; l_A := v_A \rightarrow_s a'; H_1; v_A; \cdot; l_A \mapsto H_0(l_A)$. Suppose:*

1. *$H_1 = H_0, l_A \mapsto v_A$*
2. *$\bullet; H_0, l_A \mapsto v_A; e_B \rightarrow_w \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$*
3. *$\Gamma; 2 \vdash e_A : \tau_A$ (and by inverting on* T-SET-PARTITION *and* T-LABEL-PARTITION *we know that $\Gamma(l_A) = (\tau_A, 2)$)*
4. *$\Gamma; 0 \vdash e_B : \tau_B$*
5. not-active*($e_B$)*

*then there is some $H'_1$ such that*

1. *$H_2(l_A) = v_A$ (i.e. result prior to reorder is preserved for $e_A$)*
2. *$\bullet; H_0; e_B \rightarrow_w \bullet; H'_1; e'_B; T_B; H_{\mathsf{log}_B}$ (i.e. $e_B$ becomes the same $e'_B$ as before with the same $T_B$ and $H_{\mathsf{log}_B}$ and $H'_1$ is like $H_2$ except that $H'_1(l_A) = H_0(l_A)$)*

**Proof** of Lemma 6.8. Only relevant changes are listed here.

- $\Gamma \vdash H_0 : \Gamma$
- $e_B = \mathsf{spawn}_i e,\ 0 \leq i \leq 2$. *remove these cases*
- $e_B = \mathsf{spawn}\, e''_B$. The only step available to $e_B$ is $\bullet; H_0; \mathsf{spawn}\, e''_B \rightarrow_w \bullet; H_0; 0; e''_B; \cdot$. From this, it is clear that before reordering the heaps for evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$ (and won't change $H_0$'s mapping either) so $H_2(l_A) = v_A$ as before. Similarly, $e_B$ is unaffected by the values in $H_0$ and so $e'_B = 0$, $H_{\mathsf{log}_B} = \cdot$ and $T = e''_B$ as before.
- $e_B = \mathsf{inatomic}(a, e''_B, H_{\mathsf{log}_B}, e_{B0})$. By the assumption that not-active($e_B$), this case is vacuous.
- $e_B = \mathsf{inrollback}(H_{\mathsf{log}_B}, e_{B0})$. By the assumption that not-active($e_B$), this case is vacuous.

**Lemma 6.9 (Independent Read Under Partition)** *Let $e_A = !l_A$, so under the strong semantics $a; H_0; !l_A \rightarrow_s a'; H_0; H_0(l_A); \cdot; \cdot$. Suppose:*

1. *$\Gamma \vdash H_0 : \Gamma$*
2. *$H_1 = H_0$, and $H_0(l_A) = v_A$*
3. *$\bullet; H_1; e_B \rightarrow_w \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$*
4. *$\Gamma; 2 \vdash e_A : \tau_A$ (and by inverting on* T-GET-PARTITION *and* T-LABEL-PARTITION *we know that $\Gamma(l_A) = (\tau_A, 2)$).*
5. *$\Gamma; 0 \vdash e_B : \tau_B$*
6. not-active*($e_B$)*

*then there is some $H'_1$ such that*

1. *$H_2(l_A) = v_A$ (i.e. $e_A$ reads the same value from $H_2$ as it did from $H_0$)*
2. *$\bullet; H_0; e_B \rightarrow_w \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$ (i.e. $e_B$ evaluates to the same $T_B$, $H_{\mathsf{log}}$ and $e'_B$ under $H_0$ as it did under $H_1$. This is relatively obvious since we know $H_1 = H_0$).*

**Proof** of Lemma 6.9. Only relevant changes are listed here.

- $e_B = \mathsf{spawn}_i e,\ 0 \leq i \leq 2$. *remove these cases*
- $e_B = \mathsf{spawn}\, e''_B$. The only step available to $e_B$ is $\bullet; H_0; \mathsf{spawn}\, e''_B \rightarrow_w \bullet; H_0; 0; e''_B; \cdot$. From this, it is clear that before reordering the heaps for evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$ (and won't change $H_0$'s mapping either) so $H_2(l_A) = v_A$ as before. Similarly, $e_B$ is unaffected by the values in $H_0$ and so $e'_B = 0$, $H_{\mathsf{log}_B} = \cdot$ and $T = e''_B$ as before.
- $e_B = \mathsf{inatomic}(a, e''_B, H_{\mathsf{log}_B}, e_{B0})$. By the assumption that not-active($e_B$), this case is vacuous.

- $e_B = \mathsf{inrollback}(H_{\mathsf{log}_B}, e_{B0})$. By the assumption that not-active$(e_B)$, this case is vacuous.

**Lemma 6.10 (Nested Rollback)** *If* $e_A = \mathsf{inrollback}(H'_{\mathsf{log}}, l_A \mapsto v, e_0)$ *and*

1. $\Gamma \vdash H_0 : \Gamma$
2. $\Gamma; \varepsilon \vdash e_A : \tau'_A$. *Note that what we care about isn't the* $\varepsilon$ *under which* $e_A$ *typechecks, since we really need that* $\Gamma(l_A) = (\tau, 2)$ *for some* $\tau$*, and this comes from the typing rule for rollback.*
3. $\Gamma; 0 \vdash e_B : \tau_B$
4. not-active*(*$e_B$*)*
5. $\bullet; H_0; e_A \rightarrow_s \ \bullet; H_1; \mathsf{inrollback}(H'_{\mathsf{log}}, e_0); \cdot; H_{\mathsf{log}A}$ *and*
   $\bullet; H_1; e_B \rightarrow_w \ \bullet; H_2; e'_B; T_B; H_{\mathsf{log}_B}$.

*Then there is some* $H'_1$ *such that*

1. $\bullet; H_0; e_B \rightarrow_w \ \bullet; H'_1; e'_B; T'; H_{\mathsf{log}_B}$
2. $\bullet; H'_1; e_A \rightarrow_s \ \bullet; H_2; e'_A; \cdot; \cdot$.

**Proof** of Lemma 6.10, organized by cases on $e_B$. Note that because $e_A$ typechecks, we know that $\mathsf{all2}(H'_{\mathsf{log}}, l \mapsto v)$, and therefore $\Gamma(l_A) = (\tau_A, 2)$ for some $\tau_A$.

- $e_B = v_B$. $e_B$ does not step, so this case is vacuous.
- $e_B = x$. $e_B$ does not step, so this case is vacuous.
- $e_B = \mathsf{spawn}\, e''_B$. The only step available to $e_B$ is $\bullet; H_0; \mathsf{spawn}\, e''_B \rightarrow_w \ \bullet; H_0; 0; e''_B; \cdot$. From this, it is clear that before reordering the heaps for evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$ (and won't change $H_0$'s mapping either) so $H_2(l_A) = v_A$ as before. Similarly, $e_B$ is unaffected by the values in $H_0$ and so $e'_B = 0$ and $T_0 = e''_B$ are as before.
- $e_B = \mathsf{atomic}\ e$. $e_B$ cannot step under $\bullet$. This case is vacuous.
- $e_B = e_1; e_2$. There are two cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = v_B$ and $e_2$ is not a value. Here, $e_B$ steps using the SEQ-V rule, so $\bullet; H_0; v; e_2 \rightarrow_w \bullet; H_0; e_2; \cdot; \cdot$. Notice that $H'_1 = H_0$ and so $e_A$ executes under $H'_1$ to get $H_2$ as before. Thus, $H_2(l_A) = v_A$), and $e'_B = e_2$ with $T_0 = \cdot$ as before.
- $e_B = \mathsf{if}\ e_1\ e_2\ e_3$. There are two cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = c$. Here, $e_B$ steps using either the IF-Z rule or the IF-NZ rule, so $\bullet; H_0; e_B \rightarrow_w \ \bullet; H_0; e'_B; \cdot; \cdot$. Notice that $H'_1 = H_0$ and so $e_A$ executes under $H'_1$ to get $H_2$ as before. Thus, $H_2(l_A) = v_A$), and $e'_B$ with $T_0 = \cdot$ as before.
- $e_B = e_1 := e_2$. There are three cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = l_B$ and $e_2$ is not a value. By induction.
  - $e_1 = l_B$ and $e_2 = v_B$. By assumption, we have $\Gamma; 0 \vdash e_B : \tau_B$, and since $e_1$ is some $l_B$, we can invert on the typing rules to get: $\Gamma(l_B) = (\tau_B, 0)$. We know that $\Gamma(l_A) = (\tau, 2)$, therefore $l_A$ and $l_B$ must be distinct. We know from before reordering that $H_2 = H_0, l_A \mapsto v_A, l_B \mapsto v_B$, so we pick $H'_1 = H_0, l_B \mapsto v_B$ and after evaluating $e_A$ we have $H_2 = H_0, l_B \mapsto v_B, l_A \mapsto v_A$. As a result, $H_2(l_A) = v_A$. Additionally, $e'_B = v_B$ and $T_0 = \cdot$ as before.
- $e_B = !e_1$. There are two cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = l_B$ By assumption, we have $\Gamma; 0 \vdash e_B : \tau_B$, and since $e_1$ is some $l_B$, we can invert on the typing rules to get: $\Gamma(l_B) = (\tau_B, 0)$. It is given that $\Gamma(l_A) = (\tau_A, 2)$, therefore $l_A$ and $l_B$ must be distinct. By assumption, $H_1 = H_0, l_A \mapsto v_A$. Because the labels are distinct, we also know that $H_0(l_B) = v_B$ (and therefore $e'_B = v_B = H_0(l_B)$ are as before). By picking $H'_1 = H_0$, $e_A$ evaluates under $H_0$ as it did before, giving $H_2 = H_0, l_A \mapsto v_A$ with $H_2(l_A) = v_A$.

- $e_B = e_1\ e_2$. There are three cases:
  - $e_1$ is not a value. By induction.
  - $e_1 = \lambda x.e''_B$ and $e_2$ is not a value. By induction.
  - $e_1 = \lambda x.e''_B$ and $e_2 = v_B$. The only step available to $e_B$ is $\bullet; H_0; \lambda x.e''_B\ v_B \rightarrow_w\ \bullet; H_0; e''_B[v_B/x]; \cdot; \cdot$. From this, it is clear that before reordering the heaps for evaluation, $e_B$ did not change $H_2$'s mapping for $l_A$, so $H_2(l_A) = v_A$ as before. Similarly, $e_B$ does not use any values in $H$ and so $e'_B$ and $T_0$ are as before.
- $e_B = \mathsf{inatomic}(a, e''_B, H_{\mathsf{log}_B}, e_{B0})$. By the assumption that not-active$(e_B)$, this case is vacuous.
- $e_B = \mathsf{inrollback}(H_{\mathsf{log}_B}, e_{B0})$. By the assumption that not-active$(e_B)$, this case is vacuous.
- $e_B = \text{ref } e''_B$. There are two cases:
  - $e''_B$ is not a value. By induction.
  - $e''_B$ is some $v$ In this case, a label is created and $H_2 = H_0, l_A \mapsto v_A, l_{new} \mapsto v$ and $e'_B = l_{new}$. We know that $l_{new} \notin \mathit{Dom}(H_0)$, and since $\Gamma \vdash H_0 : \Gamma$ it must be that $l_{new} \notin \mathit{Dom}(\Gamma)$. Since we also assume that $\Gamma(l_A) = (\tau_A, 2)$, it must be that that $l_A \neq l_{new}$ (otherwise, $e_A$ could not type-check under $\Gamma$). As a result, $e_B$ can step under $H_0$ in the following way: $\bullet; H_0; e_B \rightarrow_w \bullet; H_0, l_{new} \mapsto v; l_{new}; \cdot; \cdot\cdot$, giving $e'_B = l_{new}$ and $T_0 = \cdot$ as before. By picking $H'_1 = H_0, l_{new} \mapsto v$, we have $H'_1(l_A) = H_0(l_A)$, and after the write to $l_A$ we have $H_2 = H_0, l_{new} \mapsto v, l_A \mapsto v_A$ with $H_2(l_A) = v_A$.

**Lemma 6.11 (Independent Rollback)** *If* $e_A = \mathsf{inrollback}(H'_{\mathsf{log}}, l \mapsto v, e_0)$ *and*

1. $\Gamma \vdash H_0 : \Gamma$
2. $\Gamma; \varepsilon \vdash e_A : \tau_A$
3. $\Gamma; 0 \vdash e_B : \tau_B$
4. not-active*(*$e_B$*)*
5. $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_s\ \bullet; H_1; e'_A \parallel e_B \parallel T$ *and*
   $\bullet; H_1; e'_A \parallel e_B \parallel T \rightarrow_w\ \bullet; H_2; e'_A \parallel e'_B \parallel T'$.

*Then there is some* $H'_1$ *such that:*

1. $\bullet; H_0; e_A \parallel e_B \parallel T \rightarrow_w\ \bullet; H'_1; e_A \parallel e'_B \parallel T'$
2. $\bullet; H'_1; e_A \parallel e'_B \parallel T' \rightarrow_s\ \bullet; H_2; e'_A \parallel e'_B \parallel T'$.

**Proof** of Lemma 6.11: Since we know that Lemma 6.10 handles this case whenever $e_A$ takes a single step in rollback while $e_B$ takes a single step, we just apply that lemma here.

**Lemma 6.12 (Topmost Enter-Atomic)** *Suppose*

1. $\Gamma \vdash H : \Gamma$
2. $\Gamma; 0 \vdash e_B : \tau'$
3. $\Gamma; 0 \vdash e_A : \tau$
4. not-active*(*$e_A$*)*
5. not-active*(*$e_B$*)*
6. $\circ; H; e_A \parallel e_B \parallel T \rightarrow_s\ \bullet; H; e'_A \parallel e_B \parallel T$ *and*
   $\bullet; H; e'_A \parallel e_B \parallel T \rightarrow_w\ \bullet; H'; e'_A \parallel e'_B \parallel T'$.

*Then the following must be true:*

1. active*(*$e'_A$*)*
2. $\circ; H; e_A \parallel e_B \parallel T \rightarrow_w\ \circ; H'; e_A \parallel e'_B \parallel T$ *and*
   $\circ; H'; e_A \parallel e'_B \parallel T' \rightarrow_s\ \bullet; H'; e'_A \parallel e'_B \parallel T'$

**Proof** of Lemma 6.12 Restricted to the cases that need to be added or updated.

- $e_A = \mathsf{spawn}\, e$. The only step for $e_A$ is SPAWN, under which $a = a'$. This violates the assumption that $a = \circ$ and $a' = \bullet$. This case is vacuous.

- $e_A = \mathsf{spawn}_i(e)$. With $0 \leq i \leq 2$. Remove these cases.
- $e_A = \mathsf{atomic}\ e_0$. Here, only one evaluation step applies, ENTER-ATOMIC. Thus, $\circ; H; e_A \rightarrow_s \bullet; H; \mathsf{inatomic}(\circ, e_0, \cdot, \cdot)$. Deriving active($e'_A$) is obvious. Showing that $e_B$ can be moved to before $e_A$ entered a transaction requires cases on $e_B$. For each case, we must show: that $e_B$ is unaffected by $e_A$'s evaluation in terms of $H$ and $T$ and $a$. Notice that by assumption, $\bullet; H; T \parallel e_B \rightarrow_w \bullet; H'; T' \parallel e'_B$. Or rather, that $e_B$ stepped using the weak rules under $\bullet$. Also by assumption, $e_A$ does not change $H$ or $T$.
  - $e_B = \mathsf{spawn}_i(e)$. With $0 \leq i \leq 2$. Remove these cases.
  - $e_B = \mathsf{spawn}\, e$. $e_B$ steps in the following manner: $\bullet; H; e_B \rightarrow_w \bullet; H; (0 \parallel e \parallel T)$. Since $e_B$ creates $H' = H$ and $T' = e \parallel T$, while $e_A$ steps under ENTER-ATOMIC (and, as stated earlier, preserves $H$ and $T$), it must be the case that $e_B$ neither affects $e_A$ nor is affected by it. Similarly, $e_A$ is not affected by $e_B$.
  - $e_B = \mathsf{inatomic}(a, e''_B, H_{\mathsf{log}A}, e_{B0})$. This case violates the assumption that not-active($e_B$). This case is vacuous.
  - $e_B = \mathsf{inrollback}(H_{\mathsf{log}_B}, e''_{B0})$. This case violates the assumption that not-active($e_B$). This case is vacuous.
- $e_A = \mathsf{inatomic}(a, e''_A, H_{\mathsf{log}A}, e_0)$. This case violates the assumption that not-active($e_A$), so this case is vacuous.
- $e_A = \mathsf{inrollback}(H_{\mathsf{log}A}, e_0)$. This case violates the assumption that not-active($e_A$), so this case is vacuous.

**Lemma 6.13 (Serializability of Transactions)** *Suppose:*

1. $\vdash \circ; \cdot; e$
2. $\circ; \cdot; e \rightarrow_w^n \ a; H; T$

*then there exists some sequence such that*

1. $\circ; \cdot; e \rightarrow_s^n \ a; H; T$
2. *If* $a = \bullet$ *then the sequence ends with* $\circ; H; T_A \parallel e_i \parallel T_B \rightarrow_s \bullet; H; (T_A \parallel e'_i \parallel T_B)$ $\bullet; H; T \parallel e'_i \rightarrow_s^k \bullet; H; (T_A \parallel e''_i \parallel T_B)$ *where* $T_A \parallel e''_i \parallel T_B = T$

**Proof of Lemma 6.13** This proof is just like Lemma 3.8 except that it uses the appropriate lemmas in Section 6.7 where needed.

# 7 The WeakOnCommit language

As an alternative to eager update, we present a final model for software transactions with lazy update (log the desired changes to $H$) and a commit phase when the desired changes are written to the global version of $H$.

## 7.1 Syntax

Syntax changes between WeakOnCommit and StrongBasic are minor. The changes in syntax for inatomic achieve the following goals:

1. We keep a log of the modifications we would like to commit ($H_{\mathsf{oc}}$) should this transaction succeed and enter a commit phase.
2. We keep the original expression $e_0$ to re-execute in case of rollback. Both WeakUndo and WeakOn-Commit model commit and rollback in their operational semantics, although the tricky cases are different.

Similarly, we have added syntax for incommit that keeps track of the result value $v$ as well as a record of the heap values to commit to $H$.

In this situation, we must also add syntax for a stack $S$ of heaps. This way, when a nested transaction commits, we can to limit the scope of visible changes. This also allows us to define how heap accesses and memory allocation should work within transactions.

$$\begin{array}{rcl} e & ::= & c \mid l \mid x \mid e_1; e_2 \mid e_1 := e_2 \mid \text{ref } e \mid !e \mid \lambda x.e \mid e_1\ e_2 \\ & & \mid \mathsf{spawn}\, e \mid \mathsf{atomic}\ e \mid \mathsf{inatomic}(e, H_{\mathsf{oc}}, e_0) \mid \mathsf{incommit}(H_{\mathsf{oc}}, v) \\ S & ::= & \cdot \mid S{::}H \end{array}$$

## 7.2 Dynamic Evaluation

### 7.2.1 Whole Program Evaluation

Again, we adjust the PROGRAM rule to handle the new form for expression evaluation. Since there is no need to stratify $H$ using $S$ at the top-level, we "get $S$ started" by using $\cdot$ for $S$ at the top-level.

$\boxed{a; H; T \to a'; H'; T'}$

PROGRAM

$$\frac{a; \cdot{::}H; e \to a'; \cdot{::}H'; e'; T}{a; H; T_A \parallel e \parallel T_B \to a'; H'; T_A \parallel e' \parallel T_B \parallel T}$$

### 7.2.2 Expression Evaluation

Because we are now using a stack to stratify heap values in various levels of nesting, we have modified the form of evaluation under lazy update in a minor way. The SEQ-1, SPAWN, and WEAK-GET-LAZY rules are included as examples of how we have modified the rules from StrongNestedParallel to handle the stack. We have also removed the INATOMIC HELPER,We have also removed the INATOMIC HELPER, SPAWN$_0$, SPAWN$_1$, and SPAWN$_2$ rules originally defined in StrongNestedParallel.

The rules WEAK-ALLOC-LAZY and WEAK-SET-LAZY define how transactions modify $S$. The WEAK-GET-LAZY rule and the definitions of lookup define how transactions read $S$.

As with eager update, we have removed the EXIT ATOMIC rule and introduced two rules that end transactions; ROLLBACK and COMPLETE COMMIT. Note that when we commit, if the label was unallocated before the transaction began then we have no problem because we are using both "branches" of heap updates.

$\boxed{a; S; e \to a'; S'; e'; T}$

$$\textsc{seq-1}\quad \frac{a; S; e_1 \to a'; S'; e_1'; T}{a; S; e_1; e_2 \to a'; S'; e_1'; e_2; T} \qquad \textsc{spawn}\quad \frac{}{a; S; \mathsf{spawn}\, e \to a; S; 0; e} \qquad \textsc{weak-get-lazy}\quad \frac{\mathsf{lookup}(S, l) = v}{a; S; !l \to a; S; v; \cdot}$$

$$\textsc{weak-alloc-lazy}\quad \frac{}{a; S{::}H; \text{ref } v \to a; S{::}H, l \mapsto v; l; \cdot} \qquad \textsc{weak-set-lazy}\quad \frac{}{a; S{::}H; l := v \to a; S{::}H, l \mapsto v; v; \cdot}$$

$$\textsc{enter atomic}\quad \frac{}{\circ; S; \mathsf{atomic}\ e \to \bullet; S; \mathsf{inatomic}(e, \cdot, e); \cdot} \qquad \textsc{inatomic}\quad \frac{a; S{::}H_{\mathsf{oc}}; e \to a'; S{::}H'_{\mathsf{oc}}; e'; \cdot}{\bullet; S; \mathsf{inatomic}(e, H_{\mathsf{oc}}, e_0) \to \bullet; S; \mathsf{inatomic}(e', H'_{\mathsf{oc}}, e_0); \cdot}$$

$$\textsc{enter commit}\quad \frac{}{\bullet; S; \mathsf{inatomic}(v, H_{\mathsf{oc}}, e_0) \to \bullet; S; \mathsf{incommit}(H_{\mathsf{oc}}, v); \cdot}$$

$$\textsc{do commit}\quad \frac{l \in \mathit{Dom}(H_{\mathsf{oc}}) \qquad H''_{\mathsf{oc}} = H_{\mathsf{oc}} - \{l \mapsto H_{\mathsf{oc}}(l)\}}{\bullet; S{::}H; \mathsf{incommit}(H_{\mathsf{oc}}, v) \to \bullet; S{::}(H[l \mapsto H_{\mathsf{oc}}(l)]); \mathsf{incommit}(H''_{\mathsf{oc}}, v); \cdot} \qquad \textsc{complete commit}\quad \frac{}{\bullet; S; \mathsf{incommit}(\cdot, v) \to \circ; S; v; \cdot}$$

$$\textsc{rollback}\quad \frac{}{\bullet; S; \mathsf{inatomic}(H_{\mathsf{oc}}, e, e_0) \to \circ; S; \mathsf{atomic}\ e_0; \cdot}$$

We define heap lookup as follows:

$$\frac{l \in \mathit{Dom}(H)}{\mathsf{lookup}(S{::}H, l) = H(l)} \qquad \frac{l \notin \mathit{Dom}(H) \qquad \mathsf{lookup}(S, l) = v}{\mathsf{lookup}(S{::}H, l) = v}$$

## 7.3 Typecheck $e$

As with all weak languages in the AtomsFamily, WeakOnCommit uses a type-and-effect system to partition $H$. The partition definition and rules are the same for Weak. Type-checking rules for syntax not in WeakOnCommit have been removed, the rule for spawn defined for the StrongBasic language is repeated, the rule for inatomic handles the extended syntax, and a rule for incommit has been added.

$\boxed{\Gamma; \varepsilon \vdash e : \tau}$

$$\textsc{t-set-partition}\quad \frac{\Gamma; t \vdash e_1 : \text{ref}_t\tau \qquad \Gamma; t \vdash e_2 : \tau}{\Gamma; t \vdash e_1 := e_2 : \tau} \qquad \textsc{t-get-partition}\quad \frac{\Gamma; t \vdash e : \text{ref}_t\tau}{\Gamma; t \vdash !e : \tau} \qquad \textsc{t-ref-partition}\quad \frac{\Gamma; \varepsilon \vdash e : \tau}{\Gamma; \varepsilon \vdash \text{ref } e : \text{ref}_t\tau}$$

$$\textsc{t-label-partition}\quad \frac{\Gamma(l) = (\tau, t)}{\Gamma; \varepsilon \vdash l : \text{ref}_t\tau} \qquad \textsc{t-spawn}\quad \frac{\Gamma; 0 \vdash e : \tau}{\Gamma; 0 \vdash \mathsf{spawn}\, e : \text{int}} \qquad \textsc{t-incommit}\quad \frac{\Gamma; \varepsilon \vdash v : \tau \qquad \Gamma \vdash H_{\mathsf{oc}} : \Gamma}{\Gamma; \varepsilon \vdash \mathsf{incommit}(H_{\mathsf{log}}, v) : \tau}$$

$$\textsc{t-inatomic}\quad \frac{\Gamma\Gamma'; 2 \vdash e : \tau \qquad \Gamma; 2 \vdash e_0 : \tau \qquad \text{not-active}(e_0) \qquad \text{correct-atomic}(a, e) \qquad \Gamma\Gamma' \vdash H_{\mathsf{oc}} : \Gamma\Gamma'}{\Gamma; \varepsilon \vdash \mathsf{inatomic}(e, H_{\mathsf{oc}}, e_0) : \tau}$$

## 7.4 Other Typing Rules

Since we have added new syntax for stacks of heaps, we must define how to typecheck $S$. There are no other changes to this section.

$\boxed{\Gamma \vdash S : \Gamma'}$

$$\frac{\Gamma \vdash H : \Gamma'}{\Gamma \vdash \cdot{::}H : \Gamma'} \qquad \frac{\Gamma \vdash H : \Gamma' \qquad \Gamma \vdash S : \Gamma'}{\Gamma \vdash S{::}H : \Gamma'}$$

## 7.5 Activeness

### 7.5.1 Not-Active($e$)

Like in the StrongBasic language, we remove the not-active(e) rules for deleted syntax ($\mathsf{spawn}_0$, $\mathsf{spawn}_1$, and $\mathsf{spawn}_2$). We add a rule for the added syntax ($\mathsf{spawn}$):

$\boxed{\text{not-active}(e)}$

$$\frac{\text{not-active}(e)}{\text{not-active}(\mathsf{spawn}\ e)}$$

### 7.5.2 Active($e$)

We update the active($e$) rule for $\mathsf{inatomic}$ and add a rule for $\mathsf{incommit}$.

$\boxed{\text{active}(e)}$

$$\frac{}{\text{active}(\mathsf{incommit}(H_{\mathsf{oc}}, v))} \qquad \frac{}{\text{active}(\mathsf{inatomic}(e, H_{\mathsf{oc}}, e_0))}$$

### 7.5.3 Correct Atomic State of $T$

No Changes.

## 7.6 Type Safety

Future Work.

## 7.7 Equivalence

Future Work.

# 8 Languages at a Glance

## 8.1 Syntax and Expressions

The syntax and semantics differences between members of the AtomsFamily are summarized in Fig. 2.

## 8.2 Typing

The differences in type-checking $e$ and the activeness of $e$ for different members of the AtomsFamily are summarized in Fig. 3.

## 8.3 Unchanged Judgment Forms

The following are the same for every language in the AtomsFamily:

| Judgment | Section |
|---|---|
| $a; H; T \to a'; H'; T'$* | 2.2 |
| $\varepsilon \leq \varepsilon'$ | 2.4 |
| $\Gamma; \varepsilon \vdash T$ | 2.4 |
| $\Gamma \vdash H : \Gamma$ | 2.4 |
| $\vdash a; H; T$ | 2.4 |
| Initial Configuration | 2.4 |
| Terminal State | 2.4 |
| not-active$(T)$, active$(T)$, correct-atomic$(a, T)$ | 2.5.3 |

**Note:* Although the PROGRAM rule is different,
the form of the judgment never changes.

## 8.4 Type Safety Lemmas with no Significant Changes

Beyond deleting cases from the proofs in Section 2.6 for typing rules that no longer apply (e.g. T-SPAWN-2), the following lemmas have no significant changes:

Top Level Progress
Top Level Preservation
Weakening Lemma (and Context Extension Definition)
Canonical Forms
Values Effectless (note that a label may occur anywhere even with
a partition, only reading or writing its contents is restricted)
Variables Not in $\Gamma'$
Values Inactive
Effects Lemma (part 2 becomes trivial or is removed for most languages)

| Language | Syntax $\Delta$ from Section 2 | Expression Evaluation | Rules and Section | Qty. |
|---|---|---|---|---|
| StrongNestedParallel from Sec. 2 | This language is the basis for all members of the AtomsFamily | $a; H; e \to a'; H'; e'; T_0; T_1; T_2$ | SEQ-1, SEQ-V, SET-1, SET-2, STRONG-SET, REF-1, ALLOC, GET-1, STRONG-GET, APP-1, APP-2, BETA, SPAWN 0, SPAWN 1, SPAWN 2, ENTER ATOMIC, INATOMIC, INATOMIC HELPER, EXIT ATOMIC. All rules are from Section 2.2 | 19 |
| Weak from Sec. 3 | None | $a; H; e \to a'; H'; e'; T_0; T_1; T_2$ | SEQ-1, SEQ-V, SET-1, SET-2, REF-1, ALLOWS, GET-1, APP-1, APP-2, BETA, SPAWN 0, SPAWN 1, SPAWN 2, ENTER ATOMIC, INATOMIC, INATOMIC HELPER, EXIT ATOMIC from Section 2.2. WEAK-SET and WEAK-GET from Section 3.2. | 19 |
| StrongBasic from Sec. 4 | $e \ ::= \ \ldots \mid \mathsf{spawn}\, e \mid \mathsf{inatomic}(e)$<br>*Note: Also remove the syntax for* inatomic, spawn$_0$, spawn$_1$, *and* spawn$_2$ *from* StrongNestedParallel | $a; H; e \to a'; H'; e'; T$ | SEQ-1, SEQ-V, SET-1, SET-2, STRONG-SET, REF-1, ALLOC, GET-1, STRONG-GET, APP-1, APP-2, and BETA from Section 2.2, with modifications appropriate for a single $T$. SPAWN, ENTER ATOMIC, EXIT ATOMIC and INATOMIC from Section 4.2. | 16 |
| StrongUndo from Sec. 5 | $e \ ::= \ \ldots \mid \mathsf{spawn}\, e \mid \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) \mid \mathsf{inrollback}(H_{\mathsf{log}}, e_0)$<br>*Note: Also remove the syntax for* inatomic, spawn$_0$, spawn$_1$, *and* spawn$_2$ *from* StrongNestedParallel | $a; H; e \to a'; H'; e'; T; H_{\mathsf{log}}$ | SEQ-1, SEQ-V, SET-1, SET-2, REF-1, ALLOC, GET-1, APP-1, APP-2, BETA, SPAWN, SET-EAGER, GET-EAGER, ENTER ATOMIC, INATOMIC, ENTER ROLLBACK, DO ROLLBACK, COMPLETE ROLLBACK, and COMMIT from Section 5.2. | 19 |
| WeakUndo from Sec. 6 | $e \ ::= \ \ldots \mid \mathsf{spawn}\, e \mid \mathsf{inatomic}(a, e, H_{\mathsf{log}}, e_0) \mid \mathsf{inrollback}(H_{\mathsf{log}}, e_0)$<br>*Note: Also remove the syntax for* inatomic, spawn$_0$, spawn$_1$, *and* spawn$_2$ *from* StrongNestedParallel | $a; H; e \to a'; H'; e'; T; H_{\mathsf{log}}$ | SEQ-1, SEQ-V, SET-1, SET-2, REF-1, ALLOC, GET-1, APP-1, APP-2, and BETA from Section 2.2, with modifications appropriate for a single $T$ and an empty or inductive $H_{\mathsf{log}}$. SPAWN, WEAK-SET-EAGER, WEAK-GET-EAGER, ENTER ATOMIC, INATOMIC, ENTER ROLLBACK, DO ROLLBACK, COMPLETE ROLLBACK and COMMIT from Section 6.2 | 19 |
| WeakOnCommit from Sec. 7 | $e \ ::= \ \ldots \mid \mathsf{spawn}\, e \mid \mathsf{inatomic}(a, e, H_{\mathsf{oc}}, e_0) \mid \mathsf{incommit}(H_{\mathsf{oc}}, v)$<br>$S \ ::= \ \cdot \mid S{::}H$<br>*Note: Also remove the syntax for* inatomic, spawn$_0$, spawn$_1$, *and* spawn$_2$ *from* StrongNestedParallel | $a; S; e \to a'; S'; e'; T$ | SEQ-1, SEQ-V, SET-1, SET-2, REF-1, GET-1, APP-1, APP-2, and BETA from Section 2.2, with modifications appropriate for a single $T$ and using $S$ instead of $H$. SPAWN, WEAK-SET-LAZY, WEAK-GET-LAZY, WEAK-ALLOC-LAZY, ENTER ATOMIC, INATOMIC, ENTER COMMIT, DO COMMIT, COMPLETE COMMIT, and ROLLBACK from Section 7.2. We defined lookup$(S, l)$ in Section 7.2. | 19 |

Figure 2: Expression Syntax and Evaluation

| Language | Typing Rules: $\Gamma; \varepsilon \vdash e : \tau$ | Qty. | Activeness $\Delta$ |
|---|---|---|---|
| StrongNestedParallel from Sec. 2 | T-CONST, T-VAR, T-LABEL, T-SEQ, T-SET, T-REF, T-GET, T-LAMBDA, T-APP, T-SPAWN-0, T-SPAWN-1, T-SPAWN-2, T-ATOMIC, T-INATOMIC from Section 2.3 | 14 | n/a |
| Weak from Sec. 3 | T-CONST, T-VAR, T-SEQ, T-LAMBDA, T-APP, T-SPAWN-0, T-ATOMIC, T-INATOMIC from Section 2.3. T-SET-PARTITION, T-GET-PARTITION, T-REF-PARTITION, T-LABEL-PARTITION from Section 3.3. | 12 | n/a |
| StrongBasic from Sec. 4 | T-CONST, T-VAR, T-LABEL, T-SEQ, T-SET, T-GET, T-REF, T-LAMBDA, T-APP and T-ATOMIC from Section 2.3. T-INATOMIC and T-SPAWN from Section 4.3. | 12 | **Removed** not-active(spawn$_0$), not-active(spawn$_1$), and not-active(spawn$_2$). **Added** not-active(spawn). **Updated** active(inatomic). |
| StrongUndo from Sec. 5 | T-CONST, T-VAR, T-SEQ, T-LAMBDA, T-APP, from Section 2.3. T-ATOMIC, T-INATOMIC and T-INROLLBACK from Section 5.3 T-SET-PARTITION, T-GET-PARTITION, T-REF-PARTITION, T-LABEL-PARTITION and T-SPAWN from Section 6.3. | 13 | **Removed** not-active(spawn$_0$), not-active(spawn$_1$), and not-active(spawn$_2$). **Added** not-active(spawn) and active(inrollback). **Updated** active(inatomic). |
| WeakUndo from Sec. 6 | T-CONST, T-VAR, T-SEQ, T-LAMBDA, T-APP, and T-ATOMIC from Section 2.3. T-SET-PARTITION, T-GET-PARTITION, T-REF-PARTITION, T-LABEL-PARTITION T-SPAWN, T-INATOMIC and T-INROLLBACK from Section 6.3. *Added* all2($\Gamma$) *in Section 6.4.* | 13 | **Removed** not-active(spawn$_0$), not-active(spawn$_1$), and not-active(spawn$_2$). **Added** not-active(spawn) and active(inrollback). **Updated** active(inatomic). |
| WeakOnCommit from Sec. 7 | T-CONST, T-VAR, T-SEQ, T-LAMBDA, T-APP, and T-ATOMIC from Section 2.3. T-SET-PARTITION, T-GET-PARTITION, T-REF-PARTITION, T-LABEL-PARTITION T-SPAWN, T-INATOMIC and T-INCOMMIT from Section 7.3. *Added* $\Gamma \vdash S : \Gamma$ *in Section 7.4.* | 13 | **Removed** not-active(spawn$_0$), not-active(spawn$_1$), and not-active(spawn$_2$). **Added** not-active(spawn) and active(incommit). **Updated** active(inatomic). |

Figure 3: Activeness and Typing Rules