# Contextual Effects for Version-Consistent Dynamic Software Updating and Safe Concurrent Programming *

## Department of Computer Science Technical Report CS-TR-4920

Iulian Neamtiu Michael Hicks Jeffrey S. Foster Polyvios Pratikakis

University of Maryland

{neamtiu,mwh,jfoster,polyvios}@cs.umd.edu

## Abstract

This paper presents a generalization of standard effect systems that we call *contextual effects*. A traditional effect system computes the effect of an expression $e$. Our system additionally computes the effects of the computational context in which $e$ occurs. More specifically, we compute the effect of the computation that has already occurred (the *prior effect*) and the effect of the computation yet to take place (the *future effect*).

Contextual effects are useful when the past or future computation of the program is relevant at various program points. We present two substantial examples. First, we show how prior and future effects can be used to enforce *transactional version consistency* (TVC), a novel correctness property for dynamic software updates. TVC ensures that programmer-designated transactional code blocks appear to execute entirely at the same code version, even if a dynamic update occurs in the middle of the block. Second, we show how future effects can be used in the analysis of multi-threaded programs to find thread-shared locations. This is an essential step in applications such as data race detection.

## 1. Introduction

Type and effect systems provide a framework for reasoning about the possible side effects of a program's executions. Effects traditionally consider assignments or allocations, but can also track other events, such as functions called or operations performed. A standard type and effect system (Lucassen 1987; Nielson et al. 1999) proves judgments $\varepsilon; \Gamma \vdash e : \tau$, where $\varepsilon$ is the effect of the expression $e$. For many applications, knowing the effect of the *context* in which $e$ appears is also useful. For example, if $e$ includes a security-sensitive operation, then knowing the effect of execution prior to evaluating $e$ could be used to support history-based access control (Abadi and Fournet 2003; Skalka et al. 2007). Conversely, knowing the effect of execution following $e$ could be used for some forms of static garbage collection, e.g., to free initialization functions once initialization is complete (Foster et al. 2006).

The core idea of this paper is a generalization of standard effect systems to compute what we call *contextual effects*. Our contextual effect system proves judgments of the form $\Phi; \Gamma \vdash e : \tau$, where $\Phi$ is a tuple $[\alpha; \varepsilon; \omega]$ containing $\varepsilon$, the standard effect of $e$, and $\alpha$ and $\omega$, the *prior effect* and *future effect*, respectively, of $e$'s context. For example, in an application $e_1\ e_2$, the prior effect of $e_2$ includes the effect of $e_1$, and likewise the future effect of $e_1$ includes the effect of $e_2$. The first contribution of this paper is a system for computing contextual effects and a proof that the system is sound (Section 2).

The inspiration for contextual effects arose from experience with two research projects. The first considers *dynamic software updating* (DSU), a technique by which running software can be updated on-the-fly with new code and data. In prior work we formalized and implemented Ginseng, a compiler and tool suite that supports DSU for C programs (Neamtiu et al. 2006; Stoyle et al. 2007). Updates are at the granularity of function calls, meaning that following an update, active code continues with the old version, while subsequent calls are to the new version. A key consideration for update correctness is timing, since, applied at the wrong time, the changes due to an update could conflict with processing in flight. For example, suppose the original code defined function `h() { f(); g(); }`, but then is changed to move the call to `g` from `h` to `f`, i.e., `h() { f(); }` and `f() { ...; g(); }`. Suppose the update occurs just prior to the original pair of calls. The call to `f` will be to the new version that calls `g`, but then returns to its caller, the *old* `h`, which then calls `g` again, potentially leading to an error.

We address this problem with a novel correctness property called *transactional version consistency* (TVC), the second contribution of this paper. In this approach, programmers designate blocks of code as *transactions* whose execution must always be attributable to a single program version. Thus an update is only allowed within a transaction if the transaction's execution still appears due to either the old or new program version. The problematic update above would be ruled out by placing the body of `h` in a transaction. We formalize this idea in a small language Proteus-tx, which extends our prior dynamic updating calculus Proteus (Stoyle et al. 2007) with transactions. Proteus-tx's type system uses contextual effects at candidate update points within a transaction to determine what updates are safe to apply. We have proven that Proteus-tx enforces transactional version consistency, and we have developed a preliminary implementation for C. In our prior work, we manually specified that updates could occur at one or two *quiescent* program points, typically at the conclusion of event processing loops (Neamtiu et al. 2006). Using these quiescent points to identify transaction boundaries, we discovered many additional version-consistent update points within transactions, which can be used to reduce the time from when an update becomes available to when it is applied (Section 3).

The second research effort from which contextual effects arose is Locksmith (Hicks et al. 2006; Pratikakis et al. 2006), a tool that can automatically detect data races in C programs. We found that we could use contextual effects to compute what memory locations are shared among threads in a multi-threaded program. The basic idea is that thread-shared locations are exactly those in the intersection of the standard effect of a child thread and the *future* effect of the parent thread at the point the child is created. Locations accessed prior to creating the child but not afterward are

* This paper completes the paper of the same name that appears in POPL 2008 (Neamtiu et al. 2008) with a full formal development and proofs.

| | | | |
|---|---|---|---|
| Expressions | $e$ | ::= | $v \mid x \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid e\ e$ |
| | | $\mid$ | $\mathsf{if0}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e$ |
| | | $\mid$ | $\mathsf{ref}^L\ e \mid !\,e \mid e := e$ |
| Values | $v$ | ::= | $n \mid \lambda x.e$ |
| Effects | $\alpha, \varepsilon, \omega$ | ::= | $\emptyset \mid 1 \mid \{L\} \mid \varepsilon \cup \varepsilon$ |
| Contextual Effs. | $\Phi$ | ::= | $[\alpha; \varepsilon; \omega]$ |
| Types | $\tau$ | ::= | $int \mid ref^{\varepsilon}\ \tau \mid \tau \longrightarrow^{\Phi} \tau$ |
| Labels | $L$ | | |

**Figure 1.** Contextual effects source language

not shared. The final contribution of this paper is a presentation of our algorithm for computing shared locations as an extension to contextual effects. We used this analysis to compute shared locations in a range of C programs. Our algorithm finds that many locations are thread-local, and hence cannot have races (Section 4).

We believe that contextual effects have many other uses, in particular any application in which the past or future computation of the program is relevant at various program points. The remainder of the paper presents our core contextual type and effect system, followed by its application to transactional version consistency and thread sharing analysis.

## 2. Contextual effects

To begin, we present a core contextual effect system for a simple imperative calculus and prove it sound. Figure 1 presents our source language, which contains expressions $e$ that consist of values $v$ (integers or functions); variables; let binding; function application; and the conditional if0, which tests its integer-valued guard against 0. Our language also includes updatable references $\mathsf{ref}^L\ e$ along with dereference and assignment. Here we annotate each syntactic occurrence of ref with a *label* $L$, which serves as the abstract name for the locations allocated at that program point. We use labels to define contextual effects. For simplicity we do not model recursive functions directly in our language, but they can be encoded using references.

Our system uses two kinds of effect information. An *effect*, written $\alpha$, $\varepsilon$, or $\omega$, is a possibly-empty set of labels, and may be 1, the set of all labels. A *contextual effect*, written $\Phi$, is a tuple $[\alpha; \varepsilon; \omega]$. In our system, if $e'$ is a subexpression of $e$, and $e'$ has contextual effect $[\alpha; \varepsilon; \omega]$, then

- The *current effect* $\varepsilon$ is the effect of evaluating $e'$ itself.
- The *prior effect* $\alpha$ is the effect of evaluating $e$ up until we begin evaluating $e'$.
- The *future effect* $\omega$ is the effect of the remainder of the evaluation of $e$ after $e'$ is fully evaluated.

Thus $\varepsilon$ is the effect of $e'$ itself, and $\alpha \cup \omega$ is the effect of the context in which $e'$ appears—and therefore $\alpha \cup \varepsilon \cup \omega$ contains all locations accessed during the entire reduction of $e$.

To make contextual effects easier to work with, we introduce some shorthand. We write $\Phi^{\alpha}$, $\Phi^{\varepsilon}$, and $\Phi^{\omega}$ for the prior, current, and future effect components, respectively, of $\Phi$. We also write $\Phi_{\emptyset}$ for the empty effect $[1; \emptyset; 1]$—by subsumption, discussed below, an expression with this effect may appear in any context. In what follows, we refer to contextual effects simply as *effects*, for brevity.

### 2.1 Typing

We now present a type and effect system to determine the contextual effect of every subexpression in a program. Types $\tau$, listed at the end of Figure 1, include the integer type $int$; reference types $ref^{\varepsilon}\ \tau$, which denote a reference to memory of type $\tau$ where the reference itself is annotated with a label $L \in \varepsilon$; and function types

Typing

$$(\textsc{TInt})\ \frac{}{\Phi_{\emptyset}; \Gamma \vdash n : int} \qquad (\textsc{TVar})\ \frac{\Gamma(x) = \tau}{\Phi_{\emptyset}; \Gamma \vdash x : \tau}$$

$$(\textsc{TLet})\ \frac{\Phi_1; \Gamma \vdash e_1 : \tau_1 \quad \Phi_2; \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2}$$

$$(\textsc{TIf})\ \frac{\Phi_1; \Gamma \vdash e_1 : int \quad \Phi_2; \Gamma \vdash e_2 : \tau \quad \Phi_2; \Gamma \vdash e_3 : \tau \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}$$

$$(\textsc{TRef})\ \frac{\Phi; \Gamma \vdash e : \tau}{\Phi; \Gamma \vdash \mathsf{ref}^L\ e : ref^{\{L\}}\ \tau}$$

$$(\textsc{TDeref})\ \frac{\Phi_1; \Gamma \vdash e : ref^{\varepsilon}\ \tau \quad \Phi_2^{\varepsilon} = \varepsilon \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash !\,e : \tau}$$

$$(\textsc{TAssign})\ \frac{\Phi_1; \Gamma \vdash e_1 : ref^{\varepsilon}\ \tau \quad \Phi_2; \Gamma \vdash e_2 : \tau \quad \Phi_3^{\varepsilon} = \varepsilon \quad \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi}{\Phi; \Gamma \vdash e_1 := e_2 : \tau}$$

$$(\textsc{TLam})\ \frac{\Phi; \Gamma, x : \tau' \vdash e : \tau}{\Phi_{\emptyset}; \Gamma \vdash \lambda x.e : \tau' \longrightarrow^{\Phi} \tau}$$

$$[\textsc{TApp}]\ \frac{\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \quad \Phi_2; \Gamma \vdash e_2 : \tau_1 \quad \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2}$$

$$(\textsc{TSub})\ \frac{\Phi'; \Gamma \vdash e : \tau' \quad \tau' \leq \tau \quad \Phi' \leq \Phi}{\Phi; \Gamma \vdash e : \tau}$$

Effect combinator

$$(\textsc{XFlow-Ctxt})\ \frac{\Phi_1 = [\alpha_1; \varepsilon_1; (\varepsilon_2 \cup \omega_2)] \quad \Phi_2 = [(\varepsilon_1 \cup \alpha_1); \varepsilon_2; \omega_2] \quad \Phi = [\alpha_1; (\varepsilon_1 \cup \varepsilon_2); \omega_2]}{\Phi_1 \rhd \Phi_2 \ \hookrightarrow\ \Phi}$$

Subtyping

$$(\textsc{SInt})\ \frac{}{int \leq int} \qquad (\textsc{SRef})\ \frac{\tau \leq \tau' \quad \tau' \leq \tau \quad \varepsilon \subseteq \varepsilon'}{ref^{\varepsilon}\ \tau \leq ref^{\varepsilon'}\ \tau'}$$

$$(\textsc{SFun})\ \frac{\tau_1' \leq \tau_1 \quad \tau_2 \leq \tau_2' \quad \Phi \leq \Phi'}{\tau_1 \longrightarrow^{\Phi} \tau_2 \leq \tau_1' \longrightarrow^{\Phi'} \tau_2'}$$

$$(\textsc{SCtxt})\ \frac{\alpha_2 \subseteq \alpha_1 \quad \varepsilon_1 \subseteq \varepsilon_2 \quad \omega_2 \subseteq \omega_1}{[\alpha_1; \varepsilon_1; \omega_1] \leq [\alpha_2; \varepsilon_2; \omega_2]}$$

**Figure 2.** Contextual effects type system

$\tau \longrightarrow^{\Phi} \tau'$, where $\tau$ and $\tau'$ are the domain and range types, respectively, and the function has contextual effect $\Phi$.

Figure 2 presents our contextual type and effect system. The rules prove judgments of the form $\Phi; \Gamma \vdash e : \tau$, meaning in type environment $\Gamma$, expression $e$ has type $\tau$ and contextual effect $\Phi$. The first two rules, (TInt) and (TVar), assign the expected types and the empty effect, since values have no effect.

(TLet) types subexpressions $e_1$ and $e_2$, which have effects $\Phi_1$ and $\Phi_2$, respectively, and requires that these effects combine to form $\Phi$, the effect of the entire expression. We use a call-by-value semantics, and hence the effect of the let should be the effect of $e_1$ followed by the effect of $e_2$. We specify the sequencing of effects

with the combinator $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$, defined by (XFLOW-CTXT) in the middle part of Figure 2. Since $e_1$ happens before $e_2$, this rule requires that the future effect of $e_1$ be $\varepsilon_2 \cup \omega_2$, i.e., everything that happens during the evaluation of $e_2$, captured by $\varepsilon_2$, plus everything that happens after, captured by $\omega_2$. Similarly, the past effect of $e_2$ must be $\varepsilon_1 \cup \alpha_1$, since $e_2$ happens just after $e_1$. Lastly, the effect $\Phi$ of the entire expression has $\alpha_1$ as its prior effect, since $e_1$ happens first; $\omega_2$ as its future effect, since $e_2$ happens last; and $\varepsilon_1 \cup \varepsilon_2$ as its current effect, since both $e_1$ and $e_2$ are evaluated. We write $\Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi$ as shorthand for $(\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi') \wedge (\Phi' \rhd \Phi_3 \hookrightarrow \Phi)$.

(TIF) requires that its branches have the same type $\tau$ and effect $\Phi_2$, which can be achieved with subsumption (below), and uses $\rhd$ to specify that $\Phi_1$, the effect of the guard, occurs before either branch. (TREF) types memory allocation, which has no effect but places the annotation $L$ into a singleton effect $\{L\}$ on the output type. This singleton effect can be increased as necessary by using subsumption.

(TDEREF) types the dereference of a memory location of type $ref^{\varepsilon}\ \tau$. In a standard effect system, the effect of $!\,e$ is the effect of $e$ plus the effect $\varepsilon$ of accessing the pointed-to memory. Here, the effect of $e$ is captured by $\Phi_1$, and because the dereference occurs after $e$ is evaluated, (TDEREF) puts $\Phi_1$ in sequence just before some $\Phi_2$ such that $\Phi_2$'s current effect is $\varepsilon$. Therefore by (XFLOW-CTXT), $\Phi^{\varepsilon}$ is $\Phi_1^{\varepsilon} \cup \varepsilon$, and $e$'s future effect $\Phi_1^{\omega}$ must include $\varepsilon$ and the future effect of $\Phi_2$. On the other hand, $\Phi_2^{\omega}$ is unconstrained by this rule, but it will be constrained by the context, assuming the dereference is followed by another expression. (TASSIGN) is similar to (TDEREF), combining the effects $\Phi_1$ and $\Phi_2$ of its subexpressions with a $\Phi_3$ whose current effect is $\varepsilon$.

(TLAM) types the function body $e$ and sets the effect on the function arrow to be the effect of $e$. The expression as a whole has no effect, since the function produces no run-time effects until it is actually called. (TAPP) types function application, which combines $\Phi_1$, the effect of $e_1$, with $\Phi_2$, the effect of $e_2$, and $\Phi_f$, the effect of the function.

The last rule in our system, (TSUB), introduces subsumption on types and effects. The judgments $\tau' \leq \tau$ and $\Phi' \leq \Phi$ are defined at the bottom of Figure 2. (SINT), (SREF), and (SFUN) are standard, with the usual co- and contravariance where appropriate. (SCTXT) defines subsumption on effects, which is covariant in the current effect, as expected, and contravariant in both the prior and future effects. To understand the contravariance, first consider an expression $e$ with future effect $\omega_1$. Since future effects should soundly approximate (i.e., be a superset of) the locations that may be accessed in the future, we can use $e$ in any context that accesses *at most* locations in $\omega_1$. Similarly, since past effects approximate locations that were accessed in the past, we can use $e$ in any context that accessed at most locations in $\alpha_1$.

### 2.2 Semantics and Soundness

We now prove that our contextual effect system is sound. The top of Figure 3 gives some basic definitions needed for our operational semantics. We extend values $v$ to include the form $r_L$, which is a run-time heap location $r$ annotated with label $L$. We need to track labels through our operational semantics to formulate and prove soundness, but these labels need not exist at run-time. We define heaps $H$ to be maps from locations to values. Finally, we extend typing environments $\Gamma$ to assign types to heap locations.

The bottom part of Figure 3 defines a big-step operational semantics for our language. Reductions operate on *configurations* $\langle \alpha, \omega, H, e \rangle$, where $\alpha$ and $\omega$ are the sets of locations accessed before and after evaluation of $e$, respectively; $H$ is the heap; and $e$ is the expression to be evaluated. Evaluations have the form

$$\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', R \rangle$$

$$\begin{array}{lrcl} \text{Values} & v & ::= & \ldots \mid r_L \\ \text{Heaps} & H & ::= & \emptyset \mid H, r \mapsto v \\ \text{Environments} & \Gamma & ::= & \emptyset \mid \Gamma, x : \tau \mid \Gamma, r : \tau \end{array}$$

$$[\text{ID}]\ \frac{}{\langle \alpha, \omega, H, v \rangle \longrightarrow_{\emptyset} \langle \alpha, \omega, H, v \rangle}$$

$$[\text{CALL}]\ \frac{\begin{array}{c}\langle \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, \lambda x.e \rangle \\ \langle \alpha_1, \omega_1, H_1, e_2 \rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v_2 \rangle \\ \langle \alpha_2, \omega_2, H_2, e[x \mapsto v_2] \rangle \longrightarrow_{\varepsilon_3} \langle \alpha', \omega', H', v \rangle\end{array}}{\langle \alpha, \omega, H, e_1\ e_2 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle \alpha', \omega', H', v \rangle}$$

$$[\text{REF}]\ \frac{\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', v \rangle \quad r \notin \mathrm{dom}(H')}{\langle \alpha, \omega, H, \mathsf{ref}^{L}\ e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', (H', r \mapsto v), r_L \rangle}$$

$$[\text{DEREF}]\ \frac{\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega' \cup \{L\}, H', r_L \rangle \quad r \in \mathrm{dom}(H')}{\langle \alpha, \omega, H, !\,e \rangle \longrightarrow_{\varepsilon \cup \{L\}} \langle \alpha' \cup \{L\}, \omega', H', H'(r) \rangle}$$

$$[\text{ASSIGN}]\ \frac{\begin{array}{c}\langle \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, r_L \rangle \\ \langle \alpha_1, \omega_1, H_1, e_2 \rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2 \cup \{L\}, (H_2, r \mapsto v'), v \rangle\end{array}}{\begin{array}{r}\langle \alpha, \omega, H, e_1 := e_2 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \{L\}} \\ \langle \alpha_2 \cup \{L\}, \omega_2, (H_2, r \mapsto v), v \rangle\end{array}}$$

$$[\text{IF-T}]\ \frac{\begin{array}{c}\langle \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, v_1 \rangle \quad v_1 = 0 \\ \langle \alpha_1, \omega_1, H_1, e_2 \rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v \rangle\end{array}}{\langle \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle \alpha_2, \omega_2, H_2, v \rangle}$$

$$[\text{IF-F}]\ \frac{\begin{array}{c}\langle \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, v_1 \rangle \quad v_1 = n \neq 0 \\ \langle \alpha_1, \omega_1, H_1, e_3 \rangle \longrightarrow_{\varepsilon_3} \langle \alpha_3, \omega_3, H_3, v \rangle\end{array}}{\langle \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_3} \langle \alpha_3, \omega_3, H_3, v \rangle}$$

$$[\text{LET}]\ \frac{\begin{array}{c}\langle \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, v_1 \rangle \\ \langle \alpha_1, \omega_1, H_1, e_2[x \mapsto v_1] \rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v_2 \rangle\end{array}}{\langle \alpha, \omega, H, \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle \alpha_2, \omega_2, H_2, v_2 \rangle}$$

**Figure 3.** Contextual effects operational semantics (partial)

where $\varepsilon$ is the effect of evaluating $e$ and $R$ is the result of reduction, either a value $v$ or $\mathbf{err}$, indicating evaluation failed. Intuitively, $\alpha$ records a trace of what has happened in the past, and $\omega$ is a *capability* describing what locations may be accessed in the future. As evaluation proceeds, labels move from the capability $\omega$ to the trace $\alpha$.

The reduction rules are straightforward. [ID] reduces a value to itself without changing the state or the effects. [CALL] evaluates the first expression to a function, the second expression to a value, and then the function body with the formal argument replaced by the actual argument. [REF] generates a fresh location $r$, which is bound in the heap to $v$ and evaluates to $r_L$. [DEREF] reads the location $r$ in the heap and adds $L$ to the standard evaluation effect. This rule requires that the future effect after evaluating $e$ have the form $\omega' \cup \{L\}$, i.e., $L$ must be in the capability after evaluating $e$, but prior to dereferencing the result. Then $L$ is added to $\alpha'$ in the the output configuration of the rule. Notice that $\omega' \cup \{L\}$ is a standard union, and so $L$ may also be in $\omega'$. This allows the same location can be accessed multiple times. [ASSIGN] behaves similarly to [DEREF].

Lastly, [IF-T] and [IF-F] give the two cases for conditionals, and [LET] binds $x$ to the result of evaluating $e_1$ inside of $e_2$. Our semantics also includes rules (not shown) that produce $\mathbf{err}$ when the program tries to access a location that is not in the input capability, or when values are used at the wrong type.

Given this operational semantics, we can now prove that the contextual effect system in Figure 2 is sound. We only state

our lemmas and theorems here. The proofs can be found in Appendix A.

We begin with a standard definition of heap typing.

**Definition 2.1** (Heap Typing)**.** *We say heap $H$ is well-typed under $\Gamma$, written $\Gamma \vdash H$, if $\mathrm{dom}(\Gamma) = \mathrm{dom}(H)$ and if for every $r \in \mathrm{dom}(H)$, we have $\Phi_\emptyset; \Gamma \vdash H(r) : \Gamma(r)$.*

Given this definition, we show the standard effect soundness theorem, which states that the program does not go wrong and that the standard effect $\Phi^\varepsilon$ captures the effect of evaluation.

**Theorem 2.2** (Standard Effect Soundness)**.** *If $\Phi; \Gamma \vdash e : \tau$ and $\Gamma \vdash H$ and $\langle 1, 1, H, e\rangle \longrightarrow_\varepsilon \langle 1, 1, H', R\rangle$, then there is a $\Gamma' \supseteq \Gamma$ such that R is a value $v$ for which $\Phi_0; \Gamma' \vdash v : \tau$ where $\Gamma' \vdash H'$ and $\varepsilon \subseteq \Phi^\varepsilon$.*

Next, we show the operational semantics is adequate, in that it moves effects from the future to the past during evaluation.

**Lemma 2.3** (Adequacy of Semantics)**.** *If $\langle \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle \alpha', \omega', H', v\rangle$ then $\alpha' = \alpha \cup \varepsilon$ and $\omega = \omega' \cup \varepsilon$.*

Next we must define what it means for the statically-ascribed contextual effects of some expression $e$ to be sound with respect to the effects of $e$'s evaluation. Suppose that $e_p$ is a program that is well-typed according to typing derivation $\mathcal{T}$ and evaluates to some value $v$ as witnessed by an evaluation derivation $D$. Observe that each term $e_1$ that is reduced in a subderivation of $D$ is either a subterm of $e_p$, or is derived from a subterm $e_2$ of $e_p$ via reduction; in the latter case it is sound to give $e_1$ the same type and effect that $e_2$ has in $\mathcal{T}$. To reason about the soundness of the effects, therefore, we must track the static effect of expression $e_2$ as it is evaluated.

We do this by defining a new *typed operational semantics* that extends standard configurations with a typing derivation of the term in that configuration. The key property of this semantics is that it preserves the effect $\Phi$ of a term throughout its evaluation, and we prove that given standard evaluation and typing derivations of the original program, we can always construct a corresponding typed operational semantics derivation.

Finally, we prove that given a typed operational semantics derivation, the effect $\Phi$ in the typing in each configuration conservatively approximates the actual prior and future effect.

**Theorem 2.4** (Prior and Future Effect Soundness)**.** *If*

$$E :: \langle T, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle$$

*where $T :: \Phi; \Gamma \vdash e : \tau$, $\alpha \subseteq \Phi^\alpha$ and $\omega' \subseteq \Phi^\omega$ then for all sub-derivations $E_i$ of $E$, $E_i :: \langle T_i, \alpha_i, \omega_i, H_i, e_i\rangle \longrightarrow_\varepsilon \langle T_{v_i}, \alpha_i', \omega_i', H_i', v_i\rangle$ where $T_i :: \Phi_i; \Gamma_i \vdash e_i : \tau_i$, it will hold that $\alpha_i \subseteq \Phi_i^\alpha$ and $\omega_i' \subseteq \Phi_i^\omega$.*

The proof of the above theorem is by induction on the derivation, starting at the root and working towards the leaves, and relying on Theorem 2.2 and Lemma 2.3.

### 2.3 Contextual Effect Inference

The typing rules in Figure 2 form a checking system, but we would prefer to infer effect annotations rather than require the programmer to provide them. Here we sketch the inference process, which is straightforward and uses standard constraint-based techniques.

We change the rules in Figure 2 into inference rules by making three modifications. First, we make the rules syntax-driven by integrating (TSUB) into the other rules (Mitchell 1991); second, we add variables $\chi$ to represent as-yet-unknown effects; and third, we replace implicit equalities with explicit equality constraints.

The resulting rules are mostly as expected, with one interesting difference for (TAPP). We might expect inlining subsumption into (TAPP) to yield the following rule:

$$(*)\ \frac{\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \qquad \Phi_2; \Gamma \vdash e_2 : \tau_1' \qquad \tau_1' \le \tau_1 \qquad \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2}$$

However, this would cause the inferred $\Phi_f$ effect to be larger than necessary if there are multiple calls to the same function. For example, consider the following code, where f is some one-argument function, x, y, and z are references, and $A$ and $B$ identify two program points:

$$(\ \textbf{if0}\ \ldots\ \textbf{then}\ \ /\!*A*\!/\ (\mathsf{f}\ 1;\ !\mathsf{x})\ \textbf{else}\ \ /\!*B*\!/\ (\mathsf{f}\ 2;\ !\mathsf{y}));\ \ !\mathsf{z}$$

If we used rule (*), then from branch $A$, we would require $\{\mathsf{x}, \mathsf{z}\} \subseteq \Phi_f^\omega$, and from branch $B$, we would require $\{\mathsf{y}, \mathsf{z}\} \subseteq \Phi_f^\omega$, where $\Phi_f$ is the effect of function f. Putting these together, we would thus have $\Phi_f^\omega = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$. This result is appropriate, since any of those locations may be accessed after some call to f. However, consider the future effect $\Phi_A^\omega$ at program point $A$. By (XFLOW-CTXT), $\Phi_A^\omega$ would contain $\Phi_f^\omega$, and yet y will not be accessed once we reach $A$, since that access is on another branch. The analogous problem happens at program point $B$, whose future effect is polluted by x.

The problem is that our effect system conflates all calls to f. One solution would be to add Hindley-Milner style parametric polymorphism, which would address this particular example. However, even with Hindley-Milner polymorphism we would suffer the same problem at indirect function calls, e.g., in C, calls through function pointers would be monomorphic.

The solution is to notice that inlining subsumption into (TAPP) should not yield (*), but instead results in the following rule:

$$(\text{TAPP}')\ \frac{\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \qquad \Phi_2; \Gamma \vdash e_2 : \tau_1' \qquad \Phi_f \le \Phi_f' \qquad \tau_1' \le \tau_1 \qquad \Phi_f' \text{ fresh} \qquad \Phi_1 \rhd \Phi_2 \rhd \Phi_f' \hookrightarrow \Phi}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2}$$

Applied to the above example, (TAPP′) results in two constraints on the future effect of $\Phi_f$:

$$\Phi_f^\omega \supseteq \Phi_{fA}^\omega = \{\mathsf{x}, \mathsf{z}\} \qquad \Phi_f^\omega \supseteq \Phi_{fB}^\omega = \{\mathsf{y}, \mathsf{z}\}$$

Here $\Phi_{fA}$ and $\Phi_{fB}$ are the fresh function effects at the call to f in *A* and *B*, respectively. Notice that we have $\Phi_f^\omega = \{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$, as before, since f is called in both contexts. But now $\Phi_{fA}^\omega$ need not contain y, and $\Phi_{fB}$ need not contain x. Thus with (TAPP′), a function's effect summarizes all of its contexts, but does not cause the prior and future effects from different contexts to pollute each other.

To perform type inference, we apply our inference rules, viewing them as generating the *constraints* $C$ in their hypotheses, given by the following grammar:

$$C ::= \tau \le \tau' \mid \Phi \le \Phi' \mid \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$$

We can then solve the constraints by performing graph reachability to find, for each variable $\chi$, the set of base effects $\{L\}$ or 1 that reach it. In practice, these constraints can be solved very efficiently using a toolkit such as Banshee (Kodumal and Aiken 2005), which also can be used to build a context-sensitive version of inference using context-free language reachability (Pratikakis et al. 2006).

## 3. Transactional Version Consistency for DSU

In prior work we developed Ginseng, a dynamic software updating (DSU) system for C programs (Neamtiu et al. 2006; Stoyle et al. 2007). To use Ginseng, programmers construct *dynamic patches* that contain new or updated functions and data. Ginseng applies these patches to the running program, and then subsequent function calls and data accesses apply to the new versions. A key novelty of Ginseng is that it ensures dynamic updates do not violate type

```
1   int main() { ...                    19  conn accept_loop() {
2    conn = accept_loop();              20    ...
3    init_log ();                       21   while (1) {
4    handle_sess(conn);                 22    tx {
5   ... }                               23     addr = accept();
6                                       24     if (!fork())
7   void handle_sess(conn) {            25      /* child */
8     ...                               26      return conn;
9    while (1) {                        27     }
10    tx {                              28    } /* end tx */
11     cmd = get(conn.fd);              29   }
12     if (cmd == "LIST") {             30  }
13       handle_list (cmd);             31
14     } else ...                       32  void handle_list (cmd) {
15     /* new: log_log () */            33     start_log_entry ();
16    } /* end tx */                    34     ...
17   }                                  35     log_log (); /* old */
18  }                                   36  }
```

**Figure 4.** High-level structure of the `vsftpd` server

safety while still allowing patches to change function and data types (Neamtiu et al. 2006; Stoyle et al. 2007), which is often necessary (Baumann et al. 2007; Neamtiu et al. 2005).

However, while type safety is important, it is only one aspect of program correctness. In this section we introduce *transactional version consistency* (TVC), a new property that lets programmers reason more easily about the safety of updates, and show how contextual effects can enforce TVC.

To illustrate the problem with an example, consider Figure 4, which sketches the structure of `vsftpd`, an FTP server which we have dynamically updated using Ginseng (Neamtiu et al. 2006). For the moment, ignore the tx{} annotations in the code. In this program, main (lines 1–5) first calls accept_loop, whose body (lines 19–30) contains an infinite loop. Each iteration of the loop accepts a connection request (line 23) and forks a child process to handle the requested session (line 24). If forking succeeds, the child returns (line 26) to main, which initializes the session's log (line 3, function not shown) and processes the session (line 4) by calling handle_sess (lines 7–18). In turn, handle_sess processes client commands until the session is closed. We show one command processor, handle_list (lines 32–36), which is called on line 13. This function creates a log entry (line 33) that is flushed when processing is finished (line 35).

Consider the following update, inspired by actual changes to `vsftpd`. Rather than flush the log entry at the end of each command processing function (like handle_list), the update changes the code to do so in handle_sess instead, e.g., the call to log_log on line 35 is moved to line 15. Once this update is applied, the next iteration of the loop will execute the new version of the code.[1]

This update is type safe, but notice that its behavior may be incorrect depending on where it occurs. Suppose Ginseng applies the update after line 15, meaning that future calls to handle_list and the handle_sess loop body go to the new version. Then everything happens correctly: we have just called the old version of handle_list, so logging occurred, and we finish executing the old version of the handle_sess loop, and thus do not call log_log. The next iteration uses all new code, and so logging continues as usual.

In contrast, suppose the update was applied at line 11. At this point we are executing the old handle_sess, but we will call the *new* handle_list, which no longer calls log_log. Moreover, handle_list returns to its caller, i.e., the old handle_sess, which also does not call log_log. Thus no log entry is produced for the command.

[1] Updates to functions in Ginseng take effect the next time the function is called. To update the body of a long-running loop, programmers specify the loop should be treated as a tail-recursive function, which can then be updated using the normal function update mechanism. Thus the update takes effect at the next iteration of the loop (Neamtiu et al. 2006).

```
1   f()              // {f} = α        {v, g} = ω
2   v := 2;  (*)     // {f, v}         {g}
3   g();             // {f, v, g}      {}
```

**Figure 5.** A sample transaction

This example reveals a violation of what we call *version consistency*: due to the timing of the update, we execute part old code and part new code, and the overall result is inconsistent. It is easy to construct other problematic updates that violate program semantics in various ways depending where the update is applied.

Prior to the current work, we maintained version consistency in Ginseng by only allowing updates at programmer-specified positions. In `vsftpd`, we chose update points on lines 16 and 28, at the end of the connection acceptance and request processing code, just prior to the next iteration of the loop. However, this approach of having one or two well-placed update points may result in less *update availability*, meaning it may take longer for updates to occur once submitted to the program. While this may be reasonable for single-threaded programs with low-latency event processing code, in a multi-threaded program—like an operating system or embedded system—the problem could be quite serious. In particular, we would have to require that all threads reach their update points simultaneously for an update to take place. Since this is unlikely to happen naturally, we could treat update points as synchronization barriers, waiting until all threads have blocked before applying the update (Stoyle et al. 2007). However, this will degrade service while waiting for threads to block, and at worst could introduce deadlock.

### 3.1 Version Consistency via Contextual Effects

To increase update availability while ensuring version consistency, we draw inspiration from recent work on making multi-threaded programming simpler by using *atomic blocks* (Harris and Fraser 2003). The key benefit of atomic blocks is that the programmer can consider them in isolation, because the language guarantees they will be serializable with respect to the rest of the computation.

Analogously, we allow programmers to specify *transactions* within their code, and we enforce *transactional version consistency* (TVC), meaning that transactions execute as if they were entirely the old version or entirely the new version, no matter where an update actually occurs. For example, in Figure 4, we have marked the bodies of the two event-processing loops as transactions by placing them within tx{} blocks (lines 10 and 22), and thus the programmer can consider updates as occurring at lines 10 or 16 in handle_sess or lines 22 or 28 in accept_loop, which are equivalent to the manually specified locations we used earlier.

We can use contextual effects to enforce TVC while still allowing updates to occur within transactions. To see how, consider the code snippet in Figure 5. Comments show the prior and future effects after each statement, where we include both locations accessed and functions called in effects. Suppose we are running this code within a transaction and have just executed line 2, marked with a star, when an update becomes available. At this point the transaction has called f and written to v ($\alpha = \{f, v\}$), and will call g in the future ($\omega = \{g\}$). Consider the possible outcomes depending on the effect $\varepsilon$ of the update, where an update's effect consists of the set of functions and global variables that it adds or changes.

1. If $\alpha \cap \varepsilon = \emptyset$, e.g., the update only modifies g ($\varepsilon = \{g\}$), then the update is safe. In our example, this would result in using the

| | | | |
|---|---|---|---|
| Definitions | $d$ | ::= | $\mathsf{main}\ e$ |
| | | \| | $\mathsf{var}\ \mathsf{g} = v\ \mathsf{in}\ d$ |
| | | \| | $\mathsf{fun}\ \mathsf{f}(x) = e\ \mathsf{in}\ d$ |
| Expressions | $e$ | ::= | $v \mid x \mid \mathsf{let}\ x = e\ \mathsf{in}\ e \mid e\ e$ |
| | | \| | $\mathsf{if0}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e$ |
| | | \| | $\mathsf{ref}\ e \mid !\,e \mid e := e$ |
| | | \| | $\mathsf{tx}\ e \mid \mathsf{update}^{\alpha,\omega}$ |
| Values | $v$ | ::= | $n \mid \mathsf{z}$ |
| Effects | $\alpha, \omega, \varepsilon$ | ::= | $\emptyset \mid 1 \mid \{\mathsf{z}\} \mid \varepsilon \cup \varepsilon$ |
| Global symbols | $\mathsf{f}, \mathsf{g}, \mathsf{z}$ | $\in$ | $GSym$ |
| Dynamic updates | $upd$ | ::= | $\{chg, add\}$ |
| Additions | $add$ | $\in$ | $GSym \rightharpoonup (\tau \times b)$ |
| Changes | $chg$ | $\in$ | $GSym \rightharpoonup (\tau \times b)$ |
| Bindings | $b$ | ::= | $v \mid \lambda x.e$ |

**Figure 6.** Proteus-tx syntax, effects, and updates

new versions of f and v, which are the same as the old versions, with the new version of g. It is as if we applied the update at the beginning of the transaction, and the entire transaction runs under the new version.

2. If $\omega \cap \varepsilon = \emptyset$, e.g., the update only modifies f ($\varepsilon = \{\mathsf{f}\}$), then the update is also safe. In our example, this would result in using the old versions of f, v, and g. Notice that although we have called f ($\alpha \cap \varepsilon \neq \emptyset$), we never call it again after the update. It is as if we applied the update at the end of the transaction, and the whole transaction runs under the old version.
3. If $\alpha \cap \varepsilon \neq \emptyset$ and $\omega \cap \varepsilon \neq \emptyset$, e.g., the update modifies f and g, then we cannot apply the update, because that would result in using some old code (f) and some new code (g).

Putting these together, we can apply an update with effect $\varepsilon$ anywhere in a transaction such that $\alpha \cap \varepsilon = \emptyset$ or $\omega \cap \varepsilon = \emptyset$, where $\alpha$ and $\omega$ are the prior and future effects at the update point.

Our transactions are enforced using statically-determined contextual effects as defined in Section 2. Alternatively, we could log a transaction's actual effects at run-time and use these to check version consistency. Similar to common implementations of transactional memory (Harris and Fraser 2003; Herlihy and Moss 1993), we could optimistically apply an update when it becomes available, and then commit it or roll it back at the end of a transaction depending how the transaction's effect intersects with the effect of the update. This might improve update availability, since contextual effects are conservative approximations of the actual run-time effects. Nevertheless, we believe the benefits of a static approach outweigh the drawbacks. First, the static approach does not pay the overhead of logging, which is unnecessary most of the time since updates are infrequent. Second, there are no restrictions on the use of I/O within transactions; notice that network I/O occurs in both transactions in Figure 4. The optimistic approach generally must avoid I/O in transactions to properly roll back if version consistency is violated.

### 3.2 Syntax

Figure 6 presents Proteus-tx, which extends the language from Section 2 to model transactionally version-consistent dynamic updates, adapting the ideas of Proteus, our prior dynamic updating calculus (Stoyle et al. 2007). A Proteus-tx program is a definition $d$, which consists of an expression $\mathsf{main}\ e$, possibly preceded by definitions of *global symbols*, written f, g, or z and drawn from a set $GSym$. The definition $\mathsf{var}\ \mathsf{g} = v\ \mathsf{in}\ d$ binds mutable variable g to $v$ within the scope of $d$, and the definition $\mathsf{fun}\ \mathsf{f}(x) = e\ \mathsf{in}\ d$ binds f to a (possibly-recursive) function with formal parameter $x$ and body $e$.

$$(\text{TMAIN})\ \frac{\Phi; \Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{main}\ e}$$

$$(\text{TDVAR})\ \frac{\Phi_\emptyset; \Gamma \vdash v : \tau \qquad \Gamma, \mathsf{g} : \mathit{ref}^{\{\mathsf{g}\}}\ \tau \vdash d}{\Gamma \vdash \mathsf{var}\ \mathsf{g} = v\ \mathsf{in}\ d}$$

$$(\text{TDFUN})\ \frac{\begin{array}{cc}\Gamma' = \Gamma, \mathsf{f} : \tau \longrightarrow^{\Phi} \tau' & \Phi; \Gamma', x : \tau \vdash e : \tau' \\ \Gamma' \vdash d & \{\mathsf{f}\} \subseteq \Phi\end{array}}{\Gamma \vdash \mathsf{fun}\ \mathsf{f}(x) = e\ \mathsf{in}\ d}$$

$$(\text{TGVAR})\ \frac{\Gamma(\mathsf{f}) = \tau}{\Phi_\emptyset; \Gamma \vdash \mathsf{f} : \tau}$$

$$(\text{TUPDATE})\ \frac{\Phi^\alpha \subseteq \alpha' \qquad \Phi^\omega \subseteq \omega'}{\Phi; \Gamma \vdash \mathsf{update}^{\alpha',\omega'} : \mathit{int}}$$

$$(\text{TTRANSACT})\ \frac{\Phi_1; \Gamma \vdash e : \tau \qquad \Phi^\alpha \subseteq \Phi_1^\alpha \qquad \Phi^\omega \subseteq \Phi_1^\omega}{\Phi; \Gamma \vdash \mathsf{tx}\ e : \tau}$$

**Figure 7.** Proteus-tx typing (extends Figure 2)

Expressions $e$ in Proteus-tx have several small differences from the language of Figure 1. We add global symbols z to the set of values $v$. We also remove anonymous lambda bindings to keep things simpler, for reasons discussed in Section 3.4. To mark transactions, we add a form $\mathsf{tx}\ e$ for a transaction whose body is $e$.

We specify program points where dynamic updates may occur with the term $\mathsf{update}^{\alpha,\omega}$, where the annotations $\alpha$ and $\omega$ specify the prior and future effects at the update point, respectively. When evaluation reaches $\mathsf{update}^{\alpha,\omega}$, an available update is applied if its contents do not conflict with the future and prior effect annotations; otherwise evaluation proceeds without updating.

A *dynamic update* $upd$ consists of a pair of partial functions $chg$ and $add$ that describe the changes and additions, respectively, of global symbol bindings. The range of these functions is pairs $(\tau, b)$, where $b$ is the new or replacement value (which may be a function $\lambda x.e$) and $\tau$ is its type. Note that Proteus-tx disallows type-altering updates, though Section 3.6 explains how they can be supported by employing ideas from our earlier work (Stoyle et al. 2007). Also, although our implementation allows state initialization functions, for simplicity we do not model them in Proteus-tx.

Finally, effects in Proteus-tx consist of sets of global symbol names z, which represent either a dereference of or assignment to z (if it is a variable) or a call to z (if it is a function name). Because updates in Proteus-tx can only change global symbols (and do not read or write through their contents), we can ignore the effects of the latter (we use syntax $\mathsf{ref}\ e$ instead of $\mathsf{ref}^L\ e$).

### 3.3 Typing

Figure 7 extends the core contextual effect typing rules from Figure 2 to Proteus-tx. The first three rules define the judgment $\Gamma \vdash d$, meaning definition $d$ is well-typed in environment $\Gamma$. (TMAIN) types $e$ in $\Gamma$, where $e$ may have any effect and any type. (TDVAR) types the value $v$, which has the empty effect (since it is a value), and then types $d$ with g bound to a reference to $v$ labeled with effect $\{\mathsf{g}\}$. The last definition rule, (TDFUN), constructs a new environment $\Gamma'$ that extends $\Gamma$ with a binding of f to the function's type. The function body $e$ is type checked in $\Gamma'$, to allow for recursive calls. This rule also requires that f appear in all components of the function's effect $\Phi$, written $\{\mathsf{f}\} \subseteq \Phi$. We add f to the prior effect because f must have been called for its entry to be reached. We add f to the current effect so that it is included in the effect at a call site. Lastly, we add f to the future effect because f is on the call stack and we consider its continued execution to be an effect. Note that this prohibits updates to $\mathsf{main}()$, which is always on the stack. How-

ever, we can solve this problem by extracting portions of main() into separate functions, which can then be updated; Ginseng provides support to automate this process (Neamtiu et al. 2006). The next rule, (TGVAR), types global variables, which are bound in $\Gamma$. The last two rules type the dynamic updating-related elements of Proteus-tx. (TUPDATE) types update by checking that its prior and future effect annotations are supersets of (and thus conservatively approximate) the prior and future effects of the context.

Finally, (TTRANSACT) types transactions. A key design choice here is deciding how to handle nested transactions. In (TTRANSACT), we include the prior and future effects of $\Phi$, from the outer context, into those of $\Phi_1$, from the transaction body. This ensures that an update within a child transaction does not violate version consistency of its parent. However, we do not require the reverse—the components of $\Phi_1$ need not be included in $\Phi$. This has two consequences. First, sequenced transactions are free to commit independently. For example, consider the following code

tx { tx { /*A*/ }; /*B*/ tx { /*C*/ } }

According to (TTRANSACT), the effect at $B$ is included in the prior and future effects of $C$ and $A$, respectively, but not the other way around. Thus neither transaction's effect propagates into the other, and therefore does not influence any update operations in the other.

The second consequence is that version consistency for a parent transaction ignores the effects of its child transactions. This resembles *open nesting* in concurrency transactions (Ni et al. 2007). For example, suppose in the code above that $A$ and $C$ contain calls to a hash table $T$. Without the inner transaction markers, an update to $T$ available at $B$ would be rejected, because due to $A$ it would overlap with the prior effect, and due to $C$ it would overlap with the future effect. With the inner transactions in place, however, the update would be allowed. As a result, the parent transaction could use the old version of the hash table in $A$ and the new version in $C$.

This treatment of nested transactions makes sense when inner transactions contain code whose semantics is largely independent of the surrounding context, e.g., the abstraction represented by a hash table is independent of where, or how often, it is used. (Baumann et al. 2007) have applied this semantics to successfully partition dynamic updates to the K42 operating system into independent, object-sized chunks. While we believe open nesting makes sense, we can see circumstances in which closed nesting might be more natural, so we expect to refine our approach with experience.

### 3.4 Operational Semantics

Figure 8 defines a small-step operational semantics that reduces configurations $\langle n; \Sigma; H; e\rangle$, where $n$ defines the current program version (a successful dynamic update increments $n$), $\Sigma$ is the *transaction stack* (explained shortly), $H$ is the heap, and $e$ is the active program expression. Reduction rules have the form $\langle n; \Sigma; H; e\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e'\rangle$, where the event $\eta$ on the arrow is either $\mu$, a dynamic update that occurred (discussed below), or $\varepsilon$, the effect of the evaluation step.

In our semantics, heaps map references $r$ and global variables z to triples $(\tau, b, \nu)$ consisting of a type $\tau$, a binding $b$ (defined in Figure 6), and a *version set* $\nu$. The first and last components are relevant only for global symbols; the type $\tau$ is used to ensure that dynamic updates do not change the types of global bindings, and the version set $\nu$ contains all the program versions up to, and including, the current version since the corresponding variable was last updated. When an update occurs, new or changed bindings are given only the current version, while all other bindings have the current version added to their version set (i.e., we preserve the fact that the same binding was used in multiple program versions).

As evaluation proceeds, we maintain a transaction stack $\Sigma$, which is a list of pairs $(n, \sigma)$ that track the currently active transactions. Here $n$ is the version the program had when the transaction began, and $\sigma$ is a *trace*. A trace is a set of pairs $(\mathsf{z}, \nu)$, each of which represents a global symbol access paired with its version set at the time of use. The traces act as a log of dynamic events, and we track them in our semantics so we can prove that all global symbols accessed in a transaction come from the same version.

To evaluate a program $d$, we first compute $\mathcal{C}(\emptyset, d)$ using the function $\mathcal{C}$ shown at the top of Figure 8, which yields a pair $H; e$. This function implicitly uses the types derived by typing $d$ using the rules in Figure 7. Then we begin regular evaluation in the configuration $\langle 0; (0, \emptyset); H; e\rangle$, i.e., we evaluate $e$ at version 0, with initial transaction stack $(0, \emptyset)$, and with the declared bindings $H$. This causes the top-level expression $e$ in main $e$ to be treated as if it were enclosed in a transaction block.

The first several reduction rules in Figure 8 are straightforward. [LET], [REF], [DEREF], [ASSIGN], [IF-T], and [IF-F] are small-step versions of the rules in Figure 3, though normal references no longer have effects. None of these rules affects the current version or transaction stack. [CONG] is standard.

[GVAR-DEREF], [GVAR-ASSIGN], and [CALL] each have effect $\{\mathsf{z}\}$ and add $(\mathsf{z}, \nu)$ to the current transaction's trace, where $\nu$ is z's current version set. Notice that [CALL] performs dereference and application in one step, finding z in the heap and performing substitution. Since dynamic updates modify heap bindings, this ensures that every function call is to the most recent version. Notice that although both functions and variables are stored in the heap, we assign regular function types to functions ((TDFUN) in Figure 7) so that they cannot be assigned to within a program. Including $\lambda$-terms in the expression language would either complicate function typing or make it harder to define function updates so we omit them to keep things simpler.

The next several rules handle transactions. [TX-START] pushes the pair $(n, \emptyset)$ onto the right of the transaction stack, where $n$ is the current version and $\emptyset$ is the empty trace. The expression tx $e$ is reduced to intx $e$, which is a new form that represents an actively-evaluating transaction. The form intx $e$ does not appear in source programs, and its type rule matches that of tx $e$ (see Figure 9).

Next, [TX-CONG-1] and [TX-CONG-2] perform evaluation within an active transaction intx $e$ by reducing $e$ to $e'$. The latter rule applies if $e$'s reduction does not include an update, in which case the effect $\varepsilon$ of reducing $e$ is treated as $\emptyset$ in the outer transaction. This corresponds to our model of transaction nesting, which does not consider the effects of inner transactions when updating outer transactions. Otherwise, if an update occurs, then [TX-CONG-1] applies, and we use the function $\mathcal{U}$ to update version numbers on the outermost entry of the transaction stack. $\mathcal{U}$ is discussed shortly.

The key property guaranteed by Proteus-tx, that transactions are version consistent, is enforced by [TX-END], which gets stuck unless $traceOK(n'', \sigma'')$ holds. This predicate, defined just below the reduction rules, states that every element $(\mathsf{z}, \nu)$ in the transaction's trace $\sigma''$ satisfies $n'' \in \nu$, meaning that when z was used, it could be attributed to version $n''$, the version of the transaction. If this predicate is satisfied, [TX-END] strips off intx and pops the top (rightmost) entry on the transaction stack.

The last two rules handle dynamic updates. When update$^{\alpha,\omega}$ is in redex position, these rules try to apply an available update bundle $\mu$, which is a pair $(upd, dir)$ consisting of an update (from Figure 6) and a *direction* $dir$ that indicates whether we should consider the update as occurring at the beginning or end of the transaction, respectively. If $updateOK(upd, H, (\alpha, \omega), dir)$ is satisfied for some $dir$, then [UPDATE] applies and the update occurs. Otherwise [NO-UPDATE] applies, and the update must be delayed.

If [UPDATE] applies, we increment the program's version number and update the heap using $\mathcal{U}[H]^{upd}_{n+1}$, defined in the middle-right of Figure 8. This function replaces global variables and adds new

Definitions

| | | | |
|---|---|---|---|
| Heaps | $H$ | ::= | $\emptyset \mid r \mapsto (\cdot, b, \nu), H$ |
| | | $\mid$ | $\mathsf{z} \mapsto (\tau, b, \nu), H$ |
| Version sets | $\nu$ | ::= | $\emptyset \mid \{n\} \cup \nu$ |
| Traces | $\sigma$ | ::= | $\emptyset \mid (\mathsf{z}, \nu) \cup \sigma$ |
| Transaction stacks | $\Sigma$ | ::= | $\emptyset \mid (n, \sigma), \Sigma$ |
| Expressions | $e$ | ::= | $\ldots \mid r \mid \mathsf{intx}\ e$ |
| Events | $\eta$ | ::= | $\varepsilon \mid \mu$ |
| Update Direction | $dir$ | ::= | $bck \mid fwd$ |
| Update Bundles | $\mu$ | ::= | $(upd, dir)$ |

Compilation

$$\begin{array}{ll} \mathcal{C}(H; \mathsf{main}\ e) & = H; e \\ \mathcal{C}(H; \mathsf{fun}\ \mathsf{f}(x) = e\ \mathsf{in}\ d) & = \mathcal{C}(H, \mathsf{f} \mapsto (\tau \longrightarrow^{\Phi} \tau', \lambda x.e, \{0\}); d) \\ \mathcal{C}(H; \mathsf{var}\ \mathsf{g} = v\ \mathsf{in}\ d) & = \mathcal{C}(H, \mathsf{g} \mapsto (\tau, v, \{0\}); d) \end{array}$$

Evaluation Contexts

$$\begin{array}{rcl} \mathbb{E} & ::= & [\,] \mid \mathbb{E}\ e \mid v\ \mathbb{E} \mid \mathsf{let}\ x = \mathbb{E}\ \mathsf{in}\ e \\ & \mid & \mathsf{ref}\ \mathbb{E} \mid !\ \mathbb{E} \mid \mathbb{E} := e \mid r := \mathbb{E} \mid \mathsf{g} := \mathbb{E} \\ & \mid & \mathsf{if0}\ \mathbb{E}\ \mathsf{then}\ e\ \mathsf{else}\ e \end{array}$$

Computation

$$\begin{array}{rll}
[\textsc{let}] & \langle n; (n', \sigma); H; \mathsf{let}\ x = v\ \mathsf{in}\ e\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H; e[x \mapsto v]\rangle & \\
[\textsc{ref}] & \langle n; (n', \sigma); H; \mathsf{ref}\ v\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H[r \mapsto (\cdot, v, \emptyset)]; r\rangle & r \notin \mathrm{dom}(H) \\
[\textsc{deref}] & \langle n; (n', \sigma); H; !\, r\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H; v\rangle & H(r) = (\cdot, v, \emptyset) \\
[\textsc{assign}] & \langle n; (n', \sigma); H; r := v\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H[r \mapsto (\cdot, v, \emptyset)]; v\rangle & r \in \mathrm{dom}(H) \\
[\textsc{if-t}] & \langle n; (n', \sigma); H; \mathsf{if0}\ 0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H; e_1\rangle & \\
[\textsc{if-f}] & \langle n; (n', \sigma); H; \mathsf{if0}\ n''\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H; e_2\rangle & n'' \neq 0 \\
[\textsc{cong}] & \langle n; \Sigma; H; \mathbb{E}[e]\rangle \longrightarrow_{\eta} \langle n'; \Sigma'; H'; \mathbb{E}[e']\rangle & \langle n; \Sigma; H; e\rangle \longrightarrow_{\eta} \langle n'; \Sigma'; H'; e'\rangle \\
[\textsc{gvar-deref}] & \langle n; (n', \sigma); H; !\, \mathsf{z}\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); H; v\rangle & H(\mathsf{z}) = (\tau, v, \nu) \\
[\textsc{gvar-assign}] & \langle n; (n', \sigma); H; \mathsf{z} := v\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); H[\mathsf{z} \mapsto (\tau, v, \nu)]; v\rangle & H(\mathsf{z}) = (\tau, v', \nu) \\
[\textsc{call}] & \langle n; (n', \sigma); H; \mathsf{z}\ v\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); H; e[x \mapsto v]\rangle & H(\mathsf{z}) = (\tau, \lambda x.e, \nu) \\
[\textsc{tx-start}] & \langle n; (n', \sigma); H; \mathsf{tx}\ e\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma), (n, \emptyset); H; \mathsf{intx}\ e\rangle & \\
[\textsc{tx-cong-1}] & \langle n; (n'', \sigma), \Sigma; H; \mathsf{intx}\ e\rangle \longrightarrow_{\mu} \langle n'; \mathcal{U}[(n'', \sigma)]^{\mu}_{n'}, \Sigma'; H'; \mathsf{intx}\ e'\rangle & \langle n; \Sigma; H; e\rangle \longrightarrow_{\mu} \langle n'; \Sigma'; H'; e'\rangle \\
[\textsc{tx-cong-2}] & \langle n; \Sigma; H; \mathsf{intx}\ e\rangle \longrightarrow_{\emptyset} \langle n'; \Sigma'; H'; \mathsf{intx}\ e'\rangle & \langle n; \Sigma; H; e\rangle \longrightarrow_{\varepsilon} \langle n'; \Sigma'; H'; e'\rangle \\
[\textsc{tx-end}] & \langle n; ((n', \sigma'), (n'', \sigma'')); H; \mathsf{intx}\ v\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma'); H; v\rangle & \mathit{traceOK}(n'', \sigma'') \\
[\textsc{update}] & \langle n; (n', \sigma); H; \mathsf{update}^{\alpha,\omega}\rangle \longrightarrow_{(upd, dir)} \langle n+1; \mathcal{U}[(n', \sigma)]^{upd, dir}_{n+1}; \mathcal{U}[H]^{upd}_{n+1}; 1\rangle & \mathit{updateOK}(upd, H, (\alpha, \omega), dir) \\
[\textsc{no-update}] & \langle n; (n', \sigma); H; \mathsf{update}^{\alpha,\omega}\rangle \longrightarrow_{\emptyset} \langle n; (n', \sigma); H; 0\rangle &
\end{array}$$

Update Safety

$$\begin{array}{l} \mathit{updateOK}(upd, H, (\alpha, \omega), dir) = \\ \quad dir = bck \Rightarrow \alpha \cap \mathrm{dom}(upd^{chg}) = \emptyset \\ \wedge \quad dir = fwd \Rightarrow \omega \cap \mathrm{dom}(upd^{chg}) = \emptyset \\ \wedge \quad \Gamma = \mathit{types}(H) \\ \wedge \quad \Gamma_{upd} = \Gamma, \mathit{types}(upd^{add}) \\ \wedge \quad \forall \mathsf{z} \mapsto (\tau, b, \cdot) \in upd^{chg}. \\ \qquad \big(\Phi_{\emptyset}; \Gamma_{upd} \vdash b : \tau \wedge \mathit{heapType}(\tau, \mathsf{z}) = \Gamma(\mathsf{z})\big) \\ \wedge \quad \forall \mathsf{z} \mapsto (\tau, b, \cdot) \in upd^{add}. \\ \qquad \big(\Phi_{\emptyset}; \Gamma_{upd} \vdash b : \tau \wedge \mathsf{z} \notin \mathrm{dom}(H)\big) \end{array}$$

Trace Safety

$$\mathit{traceOK}(n, \sigma) = (\forall (\mathsf{z}, \nu) \in \sigma.\ n \in \nu)$$

Heap Updates

$$\mathcal{U}[(\mathsf{z} \mapsto (\tau, b, \nu), H)]^{upd}_n = \begin{cases} \mathsf{z} \mapsto (\tau, b', \{n\}), \mathcal{U}[H]^{upd}_n \\ \quad \text{if } upd^{chg}(\mathsf{z}) \mapsto (\tau, b') \\ \mathsf{z} \mapsto (\tau, b, \nu \cup \{n\}), \mathcal{U}[H]^{upd}_n \\ \quad \text{otherwise} \end{cases}$$

$$\mathcal{U}[(r \mapsto (\cdot, b, \emptyset), H)]^{upd}_n = (r \mapsto (\cdot, b, \emptyset)), \mathcal{U}[H]^{upd}_n$$

$$\mathcal{U}[\emptyset]^{upd}_n = \{\mathsf{z} \mapsto (\tau, b, \{n\}) \mid \mathsf{z} \mapsto (\tau, b) \in upd^{add}\}$$

Heap Typing Environments

$$\begin{array}{rcl} \mathit{types}(\emptyset) & = & \emptyset \\ \mathit{types}(\mathsf{z} \mapsto (\tau, b, \nu), H') & = & \mathsf{z} : \mathit{heapType}(\tau, \mathsf{z}), \mathit{types}(H') \\ \mathit{heapType}(\tau_1 \longrightarrow^{\Phi} \tau_2, \mathsf{z}) & = & \tau_1 \longrightarrow^{\Phi} \tau_2 \quad \mathsf{z} \in \Phi \\ \mathit{heapType}(\tau, \mathsf{z}) & = & \mathit{ref}^{\{\mathsf{z}\}}\ \tau \qquad \tau \neq (\tau_1 \longrightarrow^{\Phi} \tau_2) \end{array}$$

Trace Stack Updates

$$\begin{array}{rcl} \mathcal{U}[(n', \sigma)]^{upd, fwd}_n & = & (n', \sigma) \\ \mathcal{U}[(n', \sigma)]^{upd, bck}_n & = & (n, \mathcal{U}_t[\sigma]^{upd}_n) \\ \mathcal{U}_t[\sigma]^{upd}_n & = & \{(\mathsf{z}, \nu \cup \{n\} \mid \mathsf{z} \notin \mathrm{dom}(upd^{chg})\} \\ & \cup & \{(\mathsf{z}, \nu) \mid \mathsf{z} \in \mathrm{dom}(upd^{chg})\} \end{array}$$

**Figure 8.** Proteus-tx operational semantics

bindings according to the update. New and replaced bindings' version sets contain only the current version, while unchanged bindings add the current version to their existing version sets.

The $\mathit{updateOK}()$ predicate is defined just below the reduction rules in Figure 8. The first two conjuncts enforce the update safety requirement discussed in Section 3.1. There are two cases. If $dir = bck$, then we require that the update not intersect the prior effects, so that the update will appear to have happened at the beginning of the transaction. In this case, we need to update the version number of the transaction to be the new version, and any elements in the trace not modified by the update can have the new version added to their version sets, i.e., the past effect can be attributed to the new version. To do this, [UPDATE] applies the function $\mathcal{U}[(n', \sigma)]^{upd, dir}_{n+1}$, defined on the bottom right of Figure 8, with $dir = bck$. The update applies to outer transactions as well, and thus [TX-CONG-1] applies this same version number replacement process across the transaction stack.

In the other case, if $dir = fwd$, we require that the remainder of the transaction not be affected by the update, so the update will appear to have happened at the end of the transaction. In this case we need not modify the transaction stack, and hence $\mathcal{U}[(n', \sigma)]^{upd, dir}_n$ with $dir = fwd$ simply returns $(n', \sigma)$.

The remaining premises of $\mathit{updateOK}()$ determine whether the update itself is well-formed: each replacement binding must have

$$\text{TIntrans}\ \frac{\Phi_1;\Gamma \vdash e:\tau \quad \Phi^\alpha \subseteq \Phi_1^\alpha \quad \Phi^\omega \subseteq \Phi_1^\omega}{\Phi;\Gamma \vdash \mathsf{intx}\ e:\tau}$$

$$\frac{\begin{array}{l}\mathrm{dom}(\Gamma) = \mathrm{dom}(H)\\ \forall \mathsf{z} \mapsto (\tau \longrightarrow^\Phi \tau', \lambda x.e, \nu) \in H.\\ \quad \Phi;\Gamma, x:\tau \vdash e:\tau' \ \wedge\ \Gamma(\mathsf{z}) = \tau \longrightarrow^\Phi \tau' \ \wedge\ \mathsf{z} \in \Phi\\ \forall \mathsf{z} \mapsto (\tau, v, \nu) \in H. \quad \Phi_\emptyset;\Gamma \vdash v:\tau \ \wedge\ \Gamma(\mathsf{z}) = \mathit{ref}^\varepsilon\ \tau \ \wedge\ \mathsf{z} \in \varepsilon\\ \forall r \mapsto (\cdot, v, \nu) \in H. \quad \Phi_\emptyset;\Gamma \vdash v:\tau \ \wedge\ \Gamma(r) = \mathit{ref}^\varepsilon\ \tau\\ \forall \mathsf{z} \mapsto (\tau, b, \nu) \in H. \quad n \in \nu\end{array}}{n;\Gamma \ \vdash\ H}$$

$$(\text{TC1})\ \frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha\\ \mathsf{f} \in \varepsilon \Rightarrow n \in \mathit{ver}(H, \mathsf{f})\end{array}}{[\alpha;\varepsilon;\omega], \cdot; H \vdash (n, \sigma)}$$

$$(\text{TC2})\ \frac{\begin{array}{c}\Phi', \mathcal{R}; H \vdash \Sigma\\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha\\ \mathsf{f} \in \varepsilon \Rightarrow n \in \mathit{ver}(H, \mathsf{f})\end{array}}{[\alpha;\varepsilon;\omega], \Phi', \mathcal{R}; H \vdash (n, \sigma), \Sigma}$$

$$\text{where} \quad \mathit{ver}(H, \mathsf{f}) = \nu \ \mathit{iff}\ H(\mathsf{f}) = (\tau, b, \nu)$$

**Figure 9.** Proteus-tx typing extensions for proving soundness

the same type as the original, and new and added bindings must type check in the context of the updated heap.

### 3.5 Soundness

We have proven that well-typed Proteus-tx programs are version-consistent. The main result is that a well-typed, well-formed program either reduces to a value or evaluates indefinitely while preserving typing and version consistency. To prove this we need two additional judgments, shown in Figure 9. Heap typing $n;\Gamma \vdash H$ extends Definition 2.1 from the core system, where the additional conditions ensure that global symbols are well-typed, have well-formed effects, and include version $n$ (presumed to be the current version) in their version sets.

Stack well-formedness $\mathcal{R}; H \vdash \Sigma$ checks that a transaction stack $\Sigma$ is correctly approximated by a *transaction effect* $\mathcal{R}$, which consists of a list of contextual effects $\Phi$, one for each nested transaction. $\mathcal{R}$ is computed from a typing derivation in a straightforward way according to the function $[\![\Phi;\Gamma \vdash e:\tau]\!] = \mathcal{R}$, extracting $\Phi_1$ from each occurrence of (TINTRANS) recursively; the rules are not shown due to space constraints. Stack well-formedness ensures two properties. First, it ensures that each element in the trace $\sigma$ is included in the corresponding prior effect $\alpha$ (i.e., $\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha$). As a result, we know that *bck* updates that rewrite the stack will add the new version to all elements of the trace, since none have changed. Second, it ensures that elements in each transaction's current effect (i.e., the part yet to be executed) have the version of that transaction: $\mathsf{f} \in \varepsilon \Rightarrow n \in \mathit{ver}(H, \mathsf{f})$.

With this we can prove the core result:

**Theorem 3.1** (Single-step Soundness)**.** *If $\Phi;\Gamma \vdash e:\tau$ where $[\![\Phi;\Gamma \vdash e:\tau]\!] = \mathcal{R}$; and $n;\Gamma \vdash H$; and $\Phi, \mathcal{R}; H \vdash \Sigma$; and $\mathit{traceOK}(\Sigma)$, then either $e$ is a value, or there exist $n'$, $H'$, $\Sigma'$, $\Phi'$, $e'$, and $\eta$ such that $\langle n;\Sigma;H;e\rangle \longrightarrow_\eta \langle n';\Sigma';H';e'\rangle$ and $\Phi';\Gamma' \vdash e':\tau$ where $[\![\Phi';\Gamma' \vdash e':\tau]\!] = \mathcal{R}'$; and $n';\Gamma' \vdash H'$; and $\Phi', \mathcal{R}'; H' \vdash \Sigma'$; and $\mathit{traceOK}(\Sigma')$ for some $\Phi', \Gamma', \mathcal{R}'$.*

The proof is based on progress and preservation lemmas, as is standard. Details are in Appendix B.

From this lemma we can prove soundness:

**Corollary 3.2** (Soundness)**.** *If $\Phi;\Gamma \vdash e:\tau$ and $0;\Gamma \vdash H$ then $\langle 0;(0,\emptyset);H;e\rangle \rightsquigarrow_A \langle n';(n'',\sigma);H';v\rangle$ for some value $v$ or else evaluates indefinitely, where $\rightsquigarrow_A$ is the reflexive, transitive closure of the $\longrightarrow_\eta$ relation such that $A$ is a set of events $\eta$.*

### 3.6 Transactional Version Consistency for C Programs

We extended Ginseng to implement transactional version consistency for C using contextual effects. In our implementation, transactional blocks are indicated with a special label, and are written __DSU_TX : {...}. Candidate update points can be inserted either manually or, in our experiment, automatically, as discussed below. To perform effect inference, we first compute a context-sensitive points-to analysis using CIL (Necula et al. 2002). Then we generate (context-insensitive) effect constraints (following Section 2.3) using labels derived from the points-to analysis, and we solve the constraints with Banshee (Kodumal and Aiken 2005).

After computing the contextual effects, Ginseng transforms the program to make it updatable, and transforms each update point into a call to a function $\mathsf{update}(\Delta, \alpha, \omega)$. Here $\alpha$ and $\omega$ are the prior and future effects at the update point, pre-computed by our contextual effect inference, and $\Delta$ is a set of type names whose definitions cannot be modified and variables whose types cannot be modified. More specifically, $\Delta$ contains all of those entities that could be accessed—functions f that might be called, variables g that might be dereferenced, and types t whose values might be destructed—by code possibly on the stack at the time of the update, since that code will expect the old version's type (Neamtiu et al. 2006; Stoyle et al. 2007). When update is called at run time, it checks to see whether an update is available and, if so, applies the update if it is both type safe (i.e., no variable or type in $\Delta$ has been changed by the update to have a different type) and version consistent (given $\alpha$ and $\omega$). If an update is not safe, it is delayed and execution continues at the old version.

***Type-altering Updates*** Since a function f's type is annotated with its contextual effect $\Phi$, a modification to the program that causes f's effect $\Phi$ to change must be considered a change to f's type. This can occur even when f's code has not changed, e.g., if f calls g and an update changes g's effect. Our implementation handles such changes following the approach of our earlier work (Stoyle et al. 2007). In particular, if a variable f's type changes from $\tau$ to $\tau'$ due to an update, then if either $\tau' \leq \tau$ or $\tau' \not\leq \tau$ and $\mathsf{f} \notin \Delta$, the update is safe and can be applied. In the latter case, although f's type changes in an incompatible way, no active code depends on its type. On the other hand, if $\tau' \not\leq \tau$ and $\mathsf{f} \in \Delta$ then the update may be unsafe, since active code may rely on its type, and thus the update must be delayed.

***State Transformation*** Our version consistency condition is slightly more complicated in practice due to *state transformers*. A state transformer is an optional function, supplied by the programmer, that is called at update time to transform old program state into new program state. The programmer writes the state transformer as if it will be run at the beginning or end of a transaction, and our system must ensure that this appearance is true. That is, to allow an update to occur within a transaction, we must ensure that (1) the writes performed by the state transformer do not violate the version consistency of the current program transactions, and (2) the effects of the current transactions do not violate the version consistency of the state transformer itself. We achieve both ends by considering the update changes $(\mathrm{dom}(\mathit{upd}^{\mathit{chg}}))$ *and* the state transformer's current effect $\varepsilon_{xf}$ as the effect of the update when performing the usual checks for version consistency.

For example, if an update point $\mathsf{update}(\Delta, \alpha, \omega)$ is reached within a transaction, then if $\omega \cap (\varepsilon_{xf} \cup \mathrm{dom}(\mathit{upd}^{\mathit{chg}})) = \emptyset$ then the remaining actions of the transaction will not affect the state transformer, and vice versa, and so it is as if the update occurred at the end of the transaction. Likewise, if $\alpha \cap (\varepsilon_{xf} \cup \mathrm{dom}(\mathit{upd}^{\mathit{chg}})) =$

$\emptyset$ then the effect of the transaction to this point has no bearing on the execution of the state transformer, and vice versa, so it is as if the update occurred at the beginning of the transaction. Note that because state transformers can also access the heap from global variables we need to include accesses to standard heap references (i.e., names $L$ as in Section 2) in our effects.

***Non-updatable transactions*** When writing a state transformer, the programmer needs to anticipate where it might be applied in the program, i.e., the transformer might need to do different things depending on which transactions have completed (as evidenced by the current state). Thus we have found it is convenient to rule out updates in some transactions, to limit the amount of location-dependent code in a state transformer. For example, in Figure 4, we would like to forbid updates from the program start up to the first transaction on line 22, and from the end of the transaction on line 28 to the beginning of the transaction on line 10. Since this code is not run very often, prohibiting transactions in it should not significantly reduce update availability.

Formally, we could support this notion by adding a new form $\mathsf{tx}^*\ e$ that has the same type rule and operational semantics as a transaction $\mathsf{tx}\ e$, but in which no updates are allowed at run time. In the example, however, the regions to which we would like to apply $\mathsf{tx}^*$ are non-lexically scoped. We could refactor the code to form lexical blocks, but ultimately we plan to support non-lexical transactions in our implementation. For our experiments below, we simulate $\mathsf{tx}^*\ e$ transactions by simply not flowing the prior and future effect of the outer context into the $\mathsf{tx}\ e$ blocks (i.e., eliminating the constraints in the hypothesis of (TTRANSACT)). This produces the result we want and is safe because the $\mathsf{tx}\ e$ blocks in `vsftpd` are only nested inside the transaction for the top-level expression, and everything else in that expression is effectively in a $\mathsf{tx}^*$ block.

### 3.7 Experiments

We measured the potential benefits of transactional version consistency by analyzing 12 dynamic updates to `vsftpd`. The updates correspond to versions 1.1.0 through 2.0.2. For our experiment, we modified Ginseng to seed the transactions in each `vsftpd` version with candidate update points. While we could conceivably insert update points at every statement, we found through manual examination that inserting update points just before the `return` statement of any function reachable from within a transaction provides good coverage. Then we used Ginseng to infer the contextual effects and type modification restrictions at each update point, and computed at how many of them we could safely apply the update.

We conducted our experiments on an Athlon 64 X2 dual core 4600 machine with 4GB of RAM, running Debian, kernel version 2.6.18. Figure 10 summarizes our results. For each version, we list its size, the time Ginseng takes to pre-compute contextual effects and type modification restrictions, and the number of candidate update points that were automatically inserted. The analysis takes around 10 minutes for the largest example, and we expect that time could be reduced with more engineering effort. The last two columns indicate how many update points are type safe, and how many are both type safe and version consistent, with respect to the update from the version in that row to the next version. Note that determining whether an update is type safe and version consistent is very fast, and so we do not report the time for that computation.

Recall that in our prior work with `vsftpd` we manually added two update points. From the table, we can see that several additional update points are type safe and version consistent. We manually examined all of these update points. For all program versions except 1.1.0, 1.2.1, and 2.0.2pre2, we found that roughly one-third of the VC-safe update points occur somewhere in the middle of a transaction, providing better potential update availability. Another third occur close to or just before the end of a transaction, and the last third occur in dead code, providing no advantage. For the remaining versions, 1.1.0, 1.2.1, and 2.0.2pre2, we found that roughly 10% of the update points are in the middle of transactions, and almost all the remaining ones are close to the end of a transaction, with a few more in dead code.

| Version | LoC | Time (s) | Upds | Type-safe | VC-safe |
|---|---|---|---|---|---|
| 1.1.0 | 10,157 | 193 | 344 | 300 | 33 |
| 1.1.1 | 10,245 | 196 | 346 | 19 | 9 |
| 1.1.2 | 10,540 | 234 | 350 | 25 | 8 |
| 1.1.3 | 10,723 | 238 | 354 | 19 | 8 |
| 1.2.0 | 12,027 | 326 | 413 | 31 | 9 |
| 1.2.1 | 12,662 | 264 | 438 | 368 | 146 |
| 1.2.2 | 12,691 | 278 | 439 | 32 | 9 |
| 2.0.0 | 13,465 | 440 | 471 | 392 | 9 |
| 2.0.1 | 13,478 | 420 | 471 | 459 | 9 |
| 2.0.2pre2 | 13,531 | 632 | 471 | 471 | 9 |
| 2.0.2pre3 | 14,712 | 686 | 484 | 484 | 8 |
| 2.0.2 | 17,386 | 649 | 471 | 468 | 9 |

**Figure 10.** Version consistency analysis results

```
1  let x = ref 0, y = ref 1,
2      z = ref 2, w = ref 3 in
3    y := 4;
4    fork (!x; !y; !z);
5    x := 5;
6    fork (z := 2);
7    while (...) fork(w := (!w) + 1)
```

$$(\text{TFORK})\ \frac{\Phi_{ei}; \Gamma \vdash e : \tau \qquad \Phi^{\varepsilon}_{ei} \subseteq \Phi^{\varepsilon}_{i}}{\Phi_i; \Gamma \vdash \mathsf{fork}^i\ e : \tau}$$

**Figure 11.** Thread sharing analysis: example and type rule

One reason so many update points tend to occur toward the end of the transaction is due to the field-insensitivity of the alias analysis we used. In `vsftpd`, the type `vsf_session` contains a multitude of fields and is used pervasively throughout the code. The field-insensitive analysis causes spurious conflicts when one field is accessed early in the transaction but others are accessed later on, as is typical. This pushes the update points to the end of the transaction, following `vsf_session`'s last use. We plan to integrate a field-sensitive alias analysis into Ginseng to remedy this problem.

Interestingly, there are generally far more updates that are exclusively type safe than those that are both type safe and version consistent. We investigated some of these, and we found that the reasons for this varied with the update. For example, the updates that do not change `vsf_session` (e.g., 1.1.0) have a high number of type-safe update points, while those that do (e.g., 1.1.1) have far fewer. This makes sense, given `vsf_session`'s frequent use.

In summary, these results show that many update points are both type safe and version consistent, providing greater availability of updates than via manual placement. We expect still more update availability with a more accurate alias analysis.

## 4. Thread Sharing Analysis

Data race detectors (Savage et al. 1997; Flanagan and Freund 2000; Pratikakis et al. 2006; Naik et al. 2006) and other static analysis tools often perform *thread sharing analysis* to identify memory locations that may be accessed by multiple threads (and conversely those locations that are purely thread-local) in a concurrent program. This is useful because only shared locations need to be protected from concurrent access. In this section we show how contextual effects can be used to implement a thread sharing analysis.

We illustrate our analysis with an example. Suppose we have a language construct fork $e$, which creates a new thread that evaluates $e$, and consider the code in the top of Figure 11, written in an ML-like language with a while loop. This program creates four updatable references and then manipulates them in the parent thread and various child threads.

One simple but incorrect method for computing sharing would be to compute the (standard) effects of each thread and then intersect them; any location in the intersection of two threads would be considered thread-shared. In this case the effect of the main thread is $\{\mathsf{x}, \mathsf{y}\}$ (the writes on lines 3 and 5), and the effects of the threads on lines 4, 6, and 7 are $\{\mathsf{x}, \mathsf{y}, \mathsf{z}\}$, $\{\mathsf{z}\}$, and $\{\mathsf{w}\}$, respectively. Thus we would compute that x, y, and z are shared, and that w is not.

The most obvious problem here is that w is determined thread-local, even though it is shared among the several threads created on line 7. We could solve this problem by performing some kind of analysis to determine which calls to fork might be invoked multiple times, but that adds complexity and does not solve another problem involving precision.

Observe that although x and y are accessed both by the main thread and the thread created on line 4, their sharing is different. The main thread writes to x on line 5 after the child thread on line 4 may have started, hence the read and write may be simultaneous. On the other hand, the write to y on line 3 happens before (Lamport 1978; Manson et al. 2005) the child thread is created on line 4, and since there is no other write in the parent thread, we can consider y to be thread-local—there are no possible concurrent accesses from different threads.

We can solve both of these problems by using contextual effects rather than regular effects to determine sharing. The idea is simple: a location is thread-shared if it may be accessed by a child thread and by the parent thread, but only *after* the child thread is created—and we can use the future effect to compute what happens after thread creation. This takes care of the problems with loops, since the future effect of a fork in a loop will include the back-edge in the loop; and it allows the parent to modify data before a child thread is created without forcing that data to be considered shared. This technique is even more useful when we distinguish read and write effects, which we do in practice, in which case data need only be considered shared if at least one of the potential simultaneous accesses is a write.

### 4.1 Typing

The bottom of Figure 11 gives the new type rule (TFORK) needed for sharing analysis. This rule types thread creation, where each syntactic occurrence $\mathsf{fork}^i\ e$ has been named with an index $i$. In this rule, we compute the effect $\Phi_{ei}$ of the child thread separately from the effect $\Phi_i$ of the parent thread. Once we have all such effects, we can compute the set of shared locations as $\mathit{shared} = \bigcup_i (\Phi_{ei}^{\varepsilon} \cap \Phi_i^{\omega})$. In other words, a location is shared if it is accessed both in the child thread and in the parent thread after a call to fork.

There is one catch, however. Consider z from the example program in Figure 11. This particular location is not accessed by the parent thread after it is created, and hence is not (yet) in $\Phi_i^{\omega}$, and thus will not be considered shared. To handle this case, sharing among two child threads, we simply add the child's effects to the parent's effects with the constraint $\Phi_{ei}^{\varepsilon} \subseteq \Phi_i^{\varepsilon}$. Considering the example again, this causes the write to z on line 6 to be added to the parent thread's effect on the same line, and therefore it will be in the parent's future effect on line 4.

### 4.2 Implementation and Experiments

We have incorporated our thread sharing analysis into Locksmith (Pratikakis et al. 2006), a static analysis tool we developed to find data races in C programs. Locksmith works by enforcing the *guarded-by* pattern: Every shared location in the program must be consistently guarded by some lock. Locksmith requires essentially no annotations, and uses several other analyses in addition to the thread sharing analysis described above. Previous presentations of Locksmith's sharing analysis were informal (Pratikakis et al. 2006) or used a different formulation (Hicks et al. 2006) (see Section 5), and did not report the effectiveness of the analysis.

| Name | LoC | Time (s) | Shared | Total | % |
|---|---|---|---|---|---|
| aget | 1,914 | 0.40 | 60 | 338 | 18% |
| ctrace | 2,212 | 0.25 | 21 | 307 | 7% |
| engine | 2,608 | 0.49 | 10 | 390 | 3% |
| knot | 1,985 | 0.35 | 29 | 319 | 9% |
| pfscan | 1,948 | 0.25 | 26 | 238 | 11% |
| smtprc | 8,630 | 2.46 | 128 | 1077 | 12% |
| eql | 16,568 | 1.68 | 22 | 240 | 9% |
| 3c501 | 17,441 | 0.64 | 23 | 353 | 7% |
| plip | 19,141 | 0.88 | 64 | 402 | 16% |
| sundance | 19,951 | 1.05 | 25 | 633 | 4% |
| wavelan | 20,085 | 1.14 | 123 | 660 | 19% |
| hp100 | 20,368 | 1.16 | 24 | 450 | 5% |
| synclink | 24,691 | 2.65 | 219 | 1158 | 19% |

**Figure 12.** Sharing analysis results

Figure 12 shows the results. We measure the running time, the number of shared locations, and the total number of locations. Here locations include all global variables, syntactic occurrences of malloc, local variables whose address is taken, or fields of locations (our analysis is field sensitive). The results show that contextual effect inference runs very quickly, and is able to determine that many locations in the program are thread-local. On average, only 11% of the total locations are determined to be shared. Thus Locksmith can safely assume that accesses to the remaining 89% of locations need not be guarded by locks, greatly improving the precision of race detection.

## 5. Related Work

### 5.1 Effect Systems

Several researchers have proposed extending standard effect systems (Lucassen 1987; Nielson et al. 1999) to model more complex properties. One common approach is to use traces of actions for effects rather than sets of actions. These traces can be used to check that resources are accessed in the correct order (Igarashi and Kobayashi 2002), to statically enforce history-based access control (Skalka et al. 2007), and to check communication sequencing (Nielson et al. 1999). While these systems can model the ordering of events, they do not compute the prior or future effect at a program point. We believe we could combine trace effects with our approach to create a contextual trace effect system, which we leave for future work.

In prior work (Hicks et al. 2006) we introduced *continuation effects* $\gamma$, which resemble the union $\varepsilon \cup \omega$ of our standard and future effects. Judgments in this system have the form $\gamma; \Gamma \vdash e : \tau; \gamma'$, where $\gamma'$ describes the effect of $e$'s continuation in the remainder of the program, and $\gamma$ is equivalent to $\varepsilon \cup \gamma'$ where $\varepsilon$ is the standard effect of $e$. The drawback of this formulation is that the standard effect $\varepsilon$ of $e$ cannot be recovered from $\gamma$, since $(\varepsilon \cup \gamma') - \gamma' \neq \varepsilon$ when $\varepsilon \cap \gamma' \neq \emptyset$. This system also does not include prior effects.

Capabilities in Alias Types (Smith et al. 2000) and region systems like CL (Walker et al. 2000) are likewise related to our standard and future effects. A capability consists of static approximation of the memory locations that are live in the program, and thus may be dereferenced in the current expression or in later evaluation. Because these systems assume their inputs are in continuation passing style (CPS), the effect of a continuation is equivalent to our

future effects. The main differences are that we compute future effects at every program point (rather than only for continuations), that we compute prior effects, and that we do not require the input program to be CPS-converted.

### 5.2 Correctness of Dynamic Software Updating

Several systems for on-line updates have been proposed. Here we focus on how prior work controls an update's timing to assure its effects are correct.

Most work disallows updates to code that is active, i.e., actually running or referenced by the call stack. The simplest approach to updating active code, taken by several recent systems (Gilmore et al. 1997; Makris and Ryu 2007; Chen et al. 2006), is to passively wait for it to become inactive. This can be problematic for multi-threaded programs, since there is a greater possibility that active threads reference a to-be-updated object. To address this problem, (Soules et al. 2003) developed a quiescence protocol to support dynamic updating in the object-oriented K42 operating system (Baumann et al. 2005, 2007). Once an update for an object is proposed, an adaptor object is interposed to block subsequent callers of the object. Once the active threads have exited, the object is upgraded and the blocked callers are resumed. The danger is that dependencies between updated objects could result in a deadlock. While code inactivity is useful, it is not sufficient for ensuring higher-level properties like version consistency. In particular, version consistency may require delaying an update if to-be-updated objects are not currently active but were during the transaction.

(Lee 1983) proposed a generalization of the quiescence condition by allowing programmers to specify timing constraints on when elements of an update should occur; recent work by (Chen et al. 2007) is similar. As an example, the condition `update P, Q when P, M, S idle` specifies that procedures `P` and `Q` should be updated only when procedures `P`, `M`, and `S` are not active. Lee provides some guidance for using these conditions. For example, if procedure `P`'s type has changed, then an update to it and its callers should occur when all are inactive. Our work uses a program analysis to discover conditions such as these to establish the higher-level transactional version consistency property.

Prior work with Ginseng focused on ensuring that a dynamic update does not introduce type errors when it is applied (Neamtiu et al. 2006; Stoyle et al. 2007). For example, the program point just before a call to a function `f` is restricted from changing the type of `f`—to allow the update would result in the old code calling the new `f` at an incompatible type. We developed a static *updatability analysis* that gathers type constraints imposed by the active (old) code at each program point and only allows an update to take place if it satisfies the constraints. This is more fine-grained than Lee's constraints—if the type of a function changes, we can update it even when its callers are active so long as they will not call the updated function directly. Our current work is complementary to this work, as a type-safe update will not necessarily be version-consistent (as illustrated by the example in Section 3), and depending on how transactions are specified the reverse may also be true.

Our use of transactions to ensure version consistency resembles work by (Boyapati et al. 2003) on lazily upgrading objects in a *persistent object store* (POS). Using a type system that guarantees object encapsulation, their system ensures that an object's transformation function, used to initialize the state of a new version based on old state, sees only objects of the old version, which is similar to our version consistency property. How updates interact with application-level transactions is less clear to us. The assumption seems to be that updates to objects are largely semantically-independent, so there is less concern about version-related dependencies between objects within a transaction.

### 5.3 Thread Sharing Analysis

Thread sharing analysis is a key part of several tools for analyzing multi-threaded programs. Eraser (Savage et al. 1997), a dynamic data race detector, assumes locations are shared after they have been accessed by at least two threads. Our thread-sharing analysis can be seen as a static version of this approach. RCC Java (Flanagan and Freund 2000) allows programmers to manually add annotations to mark classes as thread local, so that their fields need not be guarded by locks when accessed.

Chord (Naik et al. 2006) uses a thread escape analysis to find shared locations; a location is considered shared if it is reachable from a thread object. This is more conservative than our approach, which allows data to be thread-local as long as it is not used in the parent after a child thread is forked. Chord avoids this problem by discounting constructors when determining thread sharing or data races. A newer version of Chord includes a flow-sensitive sharing analysis (Naik and Aiken 2007), but it is not described in detail.

(von Praun and Gross 2003) propose a thread sharing analysis for Java. Their system determines that an object is shared if it is accessed by multiple threads, and includes additional reasoning to reduce sharing by taking thread creation and joining into account.

RacerX (Engler and Ashcraft 2003) performs deadlock and data race detection for C. RacerX uses a heuristic, statistical approach to decide whether data is likely to be shared, based on how often it is accessed when a lock is held. This is in contrast to our approach, which tries to compute thread sharing in a sound manner.

## 6. Conclusion

We have introduced contextual effects, which extend standard effect systems to capture the effect of the context in which each subexpression appears, i.e., the effect of evaluation both before and after the evaluation of the subexpression. We formalized a core contextual type and effect system and proved it sound. We then used extensions of our core system in two applications. First, we proposed transactional version consistency, a new correctness condition for dynamic software updates. We showed how to use contextual effects to enforce this property while allowing updates to occur more frequently within programs. Second, we used contextual effects to compute locations shared between threads in concurrent programs. We determined shared locations by intersecting the future effect of the parent at thread creation time with the effect of the child. Our experimental results show that our static contextual effect system is useful in both applications. These examples show the utility of contextual effects, and we anticipate they will also prove useful in a variety of other applications.

## Acknowledgments

The authors thank Peter Sewell, Nikhil Swamy, and the anonymous referees for their insightful comments on drafts of this paper. This research was supported in part by National Science Foundation grants CCF-0541036 and CCF-0346989, and the University of Maryland Partnership with the Laboratory for Telecommunications Sciences.

## References

Martin Abadi and Cedric Fournet. Access control based on execution history. In *NDSS*, 2003.

Andrew Baumann, Gernot Heiser, Jonathan Appavoo, et al. Providing dynamic update in an operating system. In *USENIX*, 2005.

Andrew Baumann, Jonathan Appavoo, Robert W. Wisniewski, et al. Reboots are for hardware: Challenges and solutions to updating an operating system on the fly. In *USENIX*, 2007.

Chandrasekhar Boyapati, Barbara Liskov, Liuba Shrira, Chuang-Hue Moh, and Steven Richman. Lazy modular upgrades in persistent object stores. In *OOPSLA*, 2003.

Haibo Chen, Rong Chen, Fengzhe Zhang, Binyu Zang, and Pen-Chung Yew. Live updating operating systems using virtualization. In *VEE*, 2006.

Haibo Chen, Jie Yu, Rong Chen, Binyu Zang, and Pen-Chung Yew. POLUS: A POwerful Live Updating System. In *ICSE*, pages 271–281, 2007.

Dawson Engler and Ken Ashcraft. RacerX: effective, static detection of race conditions and deadlocks. In *SOSP*, 2003.

Cormac Flanagan and Stephen N. Freund. Type-based race detection for Java. In *PLDI*, 2000.

Jeffrey S. Foster, Robert Johnson, John Kodumal, and Alex Aiken. Flow-Insensitive Type Qualifiers. *TOPLAS*, 28(6):1035–1087, November 2006.

Stephen Gilmore, Dilsun Kirli, and Chris Walton. Dynamic ML without dynamic types. Technical Report ECS-LFCS-97-378, LFCS, University of Edinburgh, 1997.

Tim Harris and Keir Fraser. Language support for lightweight transactions. In *OOPSLA*, 2003.

M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA*, 1993.

Michael Hicks, Jeffrey S. Foster, and Polyvios Pratikakis. Lock Inference for Atomic Sections. In *TRANSACT*, 2006.

Atsushi Igarashi and Naoki Kobayashi. Resource Usage Analysis. In *POPL*, Portland, Oregon, 2002.

John Kodumal and Alexander Aiken. Banshee: A scalable constraint-based analysis toolkit. In *SAS*, 2005.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, 1978.

Insup Lee. *DYMOS: A Dynamic Modification System*. PhD thesis, Dept. of Computer Science, University of Wisconsin, Madison, April 1983.

John M. Lucassen. *Types and Effects: Towards the Integration of Functional and Imperative Programming*. PhD thesis, MIT Laboratory for Computer Science, August 1987. MIT/LCS/TR-408.

Kristis Makris and Kyung Dong Ryu. Dynamic and adaptive updates of non-quiescent subsystems in commodity operating system kernels. In *Proc. EuroSys*, March 2007.

Jeremy Manson, William Pugh, and Sarita V. Adve. The Java Memory Model. In *POPL*, 2005.

John C. Mitchell. Type inference with simple subtypes. *JFP*, 1(3):245–285, July 1991.

Mayur Naik and Alex Aiken. Conditional must not aliasing for static race detection. In *POPL*, 2007.

Mayur Naik, Alex Aiken, and John Whaley. Effective static race detection for java. In *PLDI*, 2006.

Iulian Neamtiu, Jeffrey S. Foster, and Michael Hicks. Understanding Source Code Evolution Using Abstract Syntax Tree Matching. In *MSR'05*, 2005. URL `http://www.cs.umd.edu/~mwh/papers/evolution.pdf`.

Iulian Neamtiu, Michael Hicks, Gareth Stoyle, and Manuel Oriol. Practical dynamic software updating for C. In *PLDI*, 2006.

Iulian Neamtiu, Michael Hicks, Jeffrey S. Foster, and Polyvios Pratikakis. Contextual effects for version-consistent dynamic software updating and safe concurrent programming. In *Proc. POPL*, January 2008. URL `http://www.cs.umd.edu/~mwh/papers/contexteffs.pdf`.

George C. Necula, Scott McPeak, Shree P. Rahul, and Westley Weimer. CIL: Intermediate language and tools for analysis and transformation of C programs. *LNCS*, 2304:213–228, 2002.

Yang Ni, Vijay S. Menon, Ali-Reza Adl-Tabatabai, et al. Open nesting in software transactional memory. In *PPoPP*, 2007.

Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999. ISBN 3540654100.

Polyvios Pratikakis, Jeffrey S. Foster, and Michael Hicks. Context-sensitive correlation analysis for detecting races. In *PLDI*, 2006.

Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. Eraser: A Dynamic Data Race Detector for Multi-Threaded Programs. In *SOSP*, 1997.

Christian Skalka, Scott Smith, and David Van Horn. Types and trace effects of higher order programs. *JFP*, July 2007. Forthcoming; available online at `http://www.journals.cambridge.org`.

Fred Smith, David Walker, and Greg Morrisett. Alias types. In *ESOP*, 2000.

Craig A. N. Soules, Jonathan Appavoo, Kevin Hui, et al. System support for online reconfiguration. In *USENIX*, 2003.

Gareth Stoyle, Michael Hicks, Gavin Bierman, Peter Sewell, and Iulian Neamtiu. *Mutatis Mutandis*: Safe and flexible dynamic software updating (full version). *TOPLAS*, 29(4):22, August 2007.

Christoph von Praun and Thomas R. Gross. Static conflict analysis for multi-threaded object-oriented programs. In *PLDI '03*, 2003.

David Walker, Karl Crary, and Greg Morrisett. Typed memory management in a calculus of capabilities. *TOPLAS*, 24(4):701–771, July 2000.

## A. Core Effect System Proofs

We add a typing rule for heap locations:

$$[\text{TLoc}]\ \frac{\Gamma(r) = \tau}{\Phi_\emptyset; \Gamma \vdash r_L : \mathit{ref}^{\{L\}}\ \tau}$$

We also add a list of evaluation rules that define when a program goes wrong:

$$[\text{Call-W}]\ \frac{\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha', \omega', H', v\rangle \quad v \neq \lambda x.e}{\langle \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_\emptyset \langle \alpha, \omega, H, \mathbf{err}\rangle}$$

$$[\text{If-W}]\ \frac{\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, v_1\rangle \quad v_1 \neq n}{\langle \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_\emptyset \langle \alpha, \omega, H, \mathbf{err}\rangle}$$

$$[\text{Deref-H-W}]\ \frac{\langle \alpha, \omega, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', r_L\rangle \quad r \notin \mathrm{dom}(H')}{\langle \alpha, \omega, H, !\,e\rangle \longrightarrow_\emptyset \langle \alpha, \omega, H, \mathbf{err}\rangle}$$

$$[\text{Deref-L-W}]\ \frac{\langle \alpha, \omega, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', r_L\rangle \quad r \in \mathrm{dom}(H') \quad L \notin \omega'}{\langle \alpha, \omega, H, !\,e\rangle \longrightarrow_\emptyset \langle \alpha, \omega, H, \mathbf{err}\rangle}$$

$$[\text{Assign-H-W}]\ \frac{\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, r_L\rangle \quad \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v\rangle \quad r \notin \mathrm{dom}(H_2)}{\langle \alpha, \omega, H, e_1 := e_2\rangle \longrightarrow_\emptyset \langle \alpha, \omega, H, \mathbf{err}\rangle}$$

$$[\text{Assign-L-W}]\ \frac{\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, r_L\rangle \quad \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, (H_2, r \mapsto v'), v\rangle \quad L \notin \omega_2}{\langle \alpha, \omega, H, e_1 := e_2\rangle \longrightarrow_\emptyset \langle \alpha, \omega, H, \mathbf{err}\rangle}$$

**Definition A.1** (Heap Typing)**.** *We say heap $H$ is well-typed under $\Gamma$, written $\Gamma \vdash H$, if*

1. $\mathrm{dom}(\Gamma) = \mathrm{dom}(H)$ *and*
2. *for every $r \in \mathrm{dom}(H)$, we have $\Phi_\emptyset; \Gamma \vdash H(r) : \Gamma(r)$.*

**Theorem A.2** (Standard Effect Soundness)**.** *If*

1. $\Phi; \Gamma \vdash e : \tau$*,*
2. $\Gamma \vdash H$ *and*
3. $\langle 1, 1, H, e\rangle \longrightarrow_\varepsilon \langle 1, 1, H', R\rangle$

*then there is a $\Gamma' \supseteq \Gamma$ such that:*

1. *$R$ is a value $v$ for which $\Phi_\emptyset; \Gamma' \vdash v : \tau$,*
2. *$\Gamma' \vdash H'$ and*
3. *$\varepsilon \subseteq \Phi^\varepsilon$.*

*Proof.* Proof by induction on the evaluation derivation.

**case** [Id] **:**

$$[\text{Id}]\ \frac{}{\langle 1, 1, H, v\rangle \longrightarrow_\emptyset \langle 1, 1, H, v\rangle}$$

Then obviously $v$ is a value and $\Phi; \Gamma \vdash v : \tau$ is given from hypothesis.

**case** [Call] **:**

From the assumptions, we have an evaluation derivation:

$$[\text{Call}]\ \frac{\langle 1, 1, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1, 1, H_1, \lambda x.e\rangle \quad \langle 1, 1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1, 1, H_2, v_2\rangle \quad \langle 1, 1, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle 1, 1, H', v\rangle}{\langle 1, 1, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle 1, 1, H', v\rangle}$$

and also a typing derivation $\Phi; \Gamma \vdash e_1\ e_2 : \tau_2$. The typing derivation could be produced either by [TSUB] or [TAPP]. We show how to handle [TSUB] now and ignore it for the other cases:

$$[\text{TSUB}]\frac{\begin{array}{c}\Phi'; \Gamma \vdash e_1\ e_2 : \tau' \\ \tau' \leq \tau \\ \Phi' \leq \Phi\end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau}$$

Assuming the theorem holds for the premise:

$$\Gamma' \supseteq \Gamma \tag{1}$$
$$\Gamma' \vdash H_1 \tag{2}$$
$$\varepsilon \subseteq \Phi'^{\varepsilon} \tag{3}$$

Then from $\Phi' \leq \Phi$ we get that $\varepsilon \subseteq \Phi'^{\varepsilon} \subseteq \Phi^{\varepsilon}$.
In the case that the last rule of the typing derivation is [TAPP]:

$$[\text{TAPP}]\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ \Phi_2; \Gamma \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2}$$

We inductively apply the theorem on the first premise:

$$\Gamma_1 \supseteq \Gamma \tag{4}$$
$$\Phi_\emptyset; \Gamma_1 \vdash \lambda x.e : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \tag{5}$$
$$\Gamma_1 \vdash H_1 \tag{6}$$
$$\varepsilon_1 \subseteq \Phi_1^{\varepsilon} \tag{7}$$

From the second premise of [TAPP], $\Gamma_1 \supseteq \Gamma$ and Lemma A.7 we get $\Phi_2; \Gamma_1 \vdash e_2 : \tau_1$. From this, $\Gamma_1 \vdash H_1$ and the second premise of the [CALL] rule, we apply the theorem inductively and get:

$$\Gamma_2 \supseteq \Gamma_1 \tag{8}$$
$$\Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1 \tag{9}$$
$$\Gamma_2 \vdash H_2 \tag{10}$$
$$\varepsilon_2 \subseteq \Phi_2^{\varepsilon} \tag{11}$$

Finally, we apply Lemma A.7 to get:

$$\Phi_\emptyset; \Gamma_2 \vdash \lambda x.e : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \tag{12}$$
$$\Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1 \tag{13}$$
$$\tag{14}$$

From the first we get:

$$[\text{TLAM}]\frac{\Phi_f; \Gamma_2, x : \tau_1 \vdash e : \tau_2}{\Phi_\emptyset; \Gamma_2 \vdash \lambda x.e : \tau_1 \longrightarrow^{\Phi_f} \tau_2}$$

From the premise and (13) it follows from Lemma A.6 that $\Phi_f; \Gamma_2 \vdash e[x \mapsto v_2] : \tau_2$. Inductively applying the theorem then gives:

$$\Gamma_3 \supseteq \Gamma_2 \tag{15}$$
$$\Gamma_3 \vdash H' \tag{16}$$
$$\Phi_\emptyset; \Gamma_3 \vdash v : \tau_2 \tag{17}$$
$$\varepsilon_3 \subseteq \Phi_f^{\varepsilon} \tag{18}$$

From $\Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi$ we get $\Phi_1^{\varepsilon} \cup \Phi_2^{\varepsilon} \cup \Phi_f^{\varepsilon} = \Phi^{\varepsilon}$.
Finally, we have shown that:

$$\Gamma_3 \supseteq \Gamma_2 \supseteq \Gamma_1 \supseteq \Gamma \tag{19}$$
$$\Gamma_3 \vdash H' \tag{20}$$
$$\Phi_\emptyset; \Gamma_3 \vdash v : \tau_2 \tag{21}$$
$$\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3 \subseteq \Phi_1^{\varepsilon} \cup \Phi_2^{\varepsilon} \cup \Phi_f^{\varepsilon} = \Phi^{\varepsilon} \tag{22}$$

**case** [REF] **:**

From assumptions:

$$[\text{TREF}]\frac{\Phi;\Gamma \vdash e : \tau}{\Phi;\Gamma \vdash \mathsf{ref}^{L}\ e : ref^{\{L\}}\ \tau} \tag{23}$$

$$\Gamma \vdash H \tag{24}$$

$$[\text{REF}]\frac{\langle 1,1,H,e\rangle \longrightarrow_{\varepsilon} \langle 1,1,H',v\rangle \quad r \notin \mathrm{dom}(H')}{\langle 1,1,H,\mathsf{ref}^{L}\ e\rangle \longrightarrow_{\varepsilon} \langle 1,1,(H', r \mapsto v), r_L\rangle} \tag{25}$$

Let $H'' = (H', r \mapsto v)$.
From the premises of the above derivations we can apply the theorem inductively and get

$$\Gamma' \supseteq \Gamma \tag{26}$$

$$\Phi_{\emptyset};\Gamma' \vdash v : \tau \tag{27}$$

$$\Gamma' \vdash H' \tag{28}$$

$$\varepsilon \subseteq \Phi^{\varepsilon} \tag{29}$$

From $\Gamma' \vdash H'$ and $r \notin \mathrm{dom}(H')$ we have $r \notin \mathrm{dom}(\Gamma')$. So, we select $\Gamma'' = \Gamma', r \mapsto \tau$. Obviously $\Gamma'' \supseteq \Gamma' \supseteq \Gamma$, and $\Gamma''(r) = \tau$, from which we get

$$[\text{TLOC}]\frac{\Gamma''(r) = \tau}{\Phi_{\emptyset};\Gamma'' \vdash r_L : ref^{\{L\}}\ \tau}$$

Moreover, $H''(r) = v$. Also, $\mathrm{dom}(\Gamma'') = \mathrm{dom}(\Gamma') \cup \{r\} = \mathrm{dom}(H') \cup \{r\} = \mathrm{dom}(H'')$. From Lemma A.7 we get $\Phi_{\emptyset};\Gamma'' \vdash v : \tau$, which means $\Phi_{\emptyset};\Gamma'' \vdash H''(r) : \Gamma''(r)$. It follows that $\Gamma'' \vdash H''$.

**case** [DEREF] **:**
From assumptions:

$$[\text{TDEREF}]\frac{\Phi_1;\Gamma \vdash e : ref^{\varepsilon_1}\ \tau \quad \Phi_2^{\varepsilon} = \varepsilon_1 \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi;\Gamma \vdash !\,e : \tau} \tag{30}$$

$$\Gamma \vdash H \tag{31}$$

$$[\text{DEREF}]\frac{\langle 1,1,H,e\rangle \longrightarrow_{\varepsilon} \langle 1,1 \cup \{L\},H',r_L\rangle \quad r \in \mathrm{dom}(H')}{\langle 1,1,H,!\,e\rangle \longrightarrow_{\varepsilon \cup \{L\}} \langle 1 \cup \{L\},1,H',H'(r)\rangle} \tag{32}$$

By applying the theorem inductively we get

$$\Gamma' \supseteq \Gamma \tag{33}$$

$$\Gamma' \vdash H' \tag{34}$$

$$\Phi_{\emptyset};\Gamma' \vdash r_L : ref^{\varepsilon}\ \tau \tag{35}$$

$$\varepsilon \subseteq \Phi_1^{\varepsilon} \tag{36}$$

From (35) and [TLOC] we have:

$$[\text{TLOC}]\frac{\Gamma'(r) = \tau}{\Phi_{\emptyset};\Gamma' \vdash r_L : ref^{\{L\}}\ \tau}$$

which means $\{L\} = \varepsilon_1$ and $\Gamma'(r) = \tau$. From $\Phi_2^{\varepsilon} = \varepsilon_1 = \{L\}$ and $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$ we have $\Phi^{\varepsilon} = \Phi_1^{\varepsilon} \cup \Phi_2^{\varepsilon} = \Phi_1^{\varepsilon} \cup \{L\}$. Also, from $\Gamma' \vdash H'$ and $r \in \mathrm{dom}(H')$ we have $\Phi_{\emptyset};\Gamma' \vdash H'(r) : \Gamma(r)$.
Then, for $\Gamma'$ it is the case that:

$$\Gamma' \supseteq \Gamma \tag{37}$$

$$\Gamma' \vdash H' \tag{38}$$

$$\Phi_{\emptyset};\Gamma' \vdash H'(r) : \tau \tag{39}$$

$$\varepsilon \cup \{L\} \subseteq \Phi_1^{\varepsilon} \cup \{L\} = \Phi^{\varepsilon} \tag{40}$$

**case** [ASSIGN] **:**

From assumption

$$\frac{\begin{array}{c}\Phi_1;\Gamma \vdash e_1 : \mathit{ref}^{\,\varepsilon}\ \tau \\ \Phi_2;\Gamma \vdash e_2 : \tau \\ \Phi_3^{\varepsilon} = \varepsilon \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi\end{array}}{\Phi;\Gamma \vdash e_1 := e_2 : \tau}\ [\textsc{TAssign}] \tag{41}$$

$$\Gamma \vdash H \tag{42}$$

$$\frac{\begin{array}{c}\langle 1,1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1,1,H_1,r_L\rangle \\ \langle 1,1,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1,1\cup\{L\},(H_2,r\mapsto v'),v\rangle\end{array}}{\langle 1,1,H,e_1 := e_2\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2\cup\{L\}} \langle 1\cup\{L\},1,(H_2,r\mapsto v),v\rangle}\ [\textsc{Assign}] \tag{43}$$

We apply the theorem inductively on the first premise:

$$\Gamma_1 \supseteq \Gamma \tag{44}$$

$$\Gamma_1 \vdash H_1 \tag{45}$$

$$\Phi_\emptyset;\Gamma_1 \vdash r_L : \mathit{ref}^{\,\varepsilon}\ \tau \tag{46}$$

$$\varepsilon_1 \subseteq \Phi_1^{\varepsilon} \tag{47}$$

From (46) and [TLoc] we have

$$\frac{\Gamma_1(r) = \tau}{\Phi_\emptyset;\Gamma_1 \vdash r_L : \mathit{ref}^{\,\{L\}}\ \tau}\ [\textsc{TLoc}]$$

So $\Phi_3^{\varepsilon} = \varepsilon = \{L\}$ and $\Gamma_1(r) = \tau$. From $\Gamma_1 \vdash H_1$ we have $\Phi_\emptyset;\Gamma_1 \vdash H_1(r) : \tau$.
We then apply the theorem inductively to the second premise:

$$\Gamma_2 \supseteq \Gamma_1 \tag{48}$$

$$\Gamma_2 \vdash (H_2, r \mapsto v') \tag{49}$$

$$\Phi_\emptyset;\Gamma_2 \vdash v : \tau \tag{50}$$

$$\varepsilon_2 \subseteq \Phi_2^{\varepsilon} \tag{51}$$

Using Lemma A.7 we get $\Phi_1;\Gamma_2 \vdash r_L : \mathit{ref}^{\,\{L\}}\ \tau$. From (49) we get $\Phi_\emptyset;\Gamma_2 \vdash v' : \tau$. Therefore, $\mathrm{dom}(\Gamma_2) = \mathrm{dom}(H_2, r \mapsto v') = \mathrm{dom}(H_2, r \mapsto v)$. and for all $r' \in \mathrm{dom}(H_2) \cup \{r\}.\Phi_\emptyset;\Gamma_2 \vdash (H_2, r \mapsto v)(r') : \Gamma_2(r)$.
Finally:

$$\Gamma_2 \supseteq \Gamma_1 \supseteq \Gamma \tag{52}$$

$$\Gamma_2 \vdash (H_2, r \mapsto v) \tag{53}$$

$$\Phi_\emptyset;\Gamma_2 \vdash v : \tau \tag{54}$$

$$\varepsilon_1 \cup \varepsilon_2 \cup \{L\} \subseteq \Phi_1^{\varepsilon} \cup \Phi_2^{\varepsilon} \cup \Phi_3^{\varepsilon} = \Phi^{\varepsilon} \tag{55}$$

**case** [If-T] **:**
From assumption

$$\frac{\begin{array}{c}\Phi_1;\Gamma \vdash e_1 : \mathit{int} \\ \Phi_2;\Gamma \vdash e_2 : \tau \\ \Phi_2;\Gamma \vdash e_3 : \tau \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi;\Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}\ (\textsc{TIf}) \tag{56}$$

$$\Gamma \vdash H \tag{57}$$

$$\frac{\begin{array}{c}\langle 1,1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1,1,H_1,v_1\rangle \\ v_1 = 0 \\ \langle 1,1,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1,1,H_2,v\rangle\end{array}}{\langle 1,1,H,\mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2} \langle 1,1,H_2,v\rangle}\ [\textsc{If-T}] \tag{58}$$

We apply the theorem inductively to the first premise:

$$\Gamma_1 \supseteq \Gamma \tag{59}$$

$$\Gamma_1 \vdash H_1 \tag{60}$$

$$\Phi_\emptyset;\Gamma_1 \vdash v_1 : \mathit{int} \tag{61}$$

$$\varepsilon_1 \subseteq \Phi_1^{\varepsilon} \tag{62}$$

By Lemma A.7 and the second premise of [TIf] we have $\Phi_2;\Gamma_1 \vdash e_2 : \tau$. So, we can apply the theorem inductively to get:

$$\Gamma_2 \supseteq \Gamma_1 \tag{63}$$

$$\Gamma_2 \vdash H_2 \tag{64}$$

$$\Phi_\emptyset;\Gamma_2 \vdash v : \tau \tag{65}$$

$$\varepsilon_2 \subseteq \Phi_2^{\varepsilon} \tag{66}$$

From $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$ we have $\Phi^\varepsilon = \Phi_1^\varepsilon \cup \Phi_2^\varepsilon$
Finally we show

$$\Gamma_2 \supseteq \Gamma_1 \supseteq \Gamma \tag{67}$$
$$\Gamma_2 \vdash H_2 \tag{68}$$
$$\Phi_\emptyset; \Gamma_2 \vdash v : \tau \tag{69}$$
$$\varepsilon_1 \cup \varepsilon_2 \subseteq \Phi_1^\varepsilon \cup \Phi_2^\varepsilon = \Phi^\varepsilon \tag{70}$$

**case** [IF-F] **:**
Similar to the above.
**case** [LET] **:**
Similar to [CALL].
**case** [CALL-W] **:**
Assume:

$$\text{[TApp]}\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ \Phi_2; \Gamma \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2} \tag{71}$$
$$\Gamma \vdash H \tag{72}$$
$$\text{[Call-W]}\frac{\begin{array}{c}\langle 1, 1, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1, 1, H', v\rangle \\ v \neq \lambda x.e\end{array}}{\langle 1, 1, H, e_1\ e_2\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle} \tag{73}$$

We apply the theorem recursively to the first premise:

$$\Gamma' \supseteq \Gamma \tag{74}$$
$$\Gamma' \vdash H' \tag{75}$$
$$\Phi_\emptyset; \Gamma' \vdash v : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \tag{76}$$
$$\varepsilon_1 \subseteq \Phi_1^\varepsilon \tag{77}$$

Then the only rule that can create a derivation for (76) is [TLAM], meaning $v = \lambda x.e$. But we have $v \neq \lambda x.e$, a contradiction. Therefore, there is no derivation for $\langle 1, 1, H, e_1\ e_2\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle$ when $\Gamma \vdash H$ and $\Phi; \Gamma \vdash e_1\ e_2 : \tau_2$.
**case** [IF-W] **:**
Assume:

$$\text{(TIf)}\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : int \\ \Phi_2; \Gamma \vdash e_2 : \tau \\ \Phi_2; \Gamma \vdash e_3 : \tau \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau} \tag{78}$$
$$\Gamma \vdash H \tag{79}$$
$$\text{[If-W]}\frac{\begin{array}{c}\langle 1, 1, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1, 1, H_1, v_1\rangle \\ v_1 \neq n\end{array}}{\langle 1, 1, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle} \tag{80}$$

We apply the theorem recursively to the first premise:

$$\Gamma' \supseteq \Gamma \tag{81}$$
$$\Gamma' \vdash H' \tag{82}$$
$$\Phi_\emptyset; \Gamma' \vdash v_1 : int \tag{83}$$
$$\varepsilon_1 \subseteq \Phi_1^\varepsilon \tag{84}$$

Then the only rule that can create a derivation for (83) is [TINT], meaning $v_1 = n$. But we have $v_1 \neq n$, a contradiction.
**case** [DEREF-H-W] **:**
Proof by contradiction, similar to the previous case. Assume there is a derivation that evaluates to $\mathbf{err}$, under the assumptions:

$$\text{[TDeref]}\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e : ref^{\varepsilon_1}\ \tau \\ \Phi_2^\varepsilon = \varepsilon_1 \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash !\,e : \tau} \tag{85}$$
$$\Gamma \vdash H \tag{86}$$
$$\text{[Deref-H-W]}\frac{\begin{array}{c}\langle 1, 1, H, e\rangle \longrightarrow_\varepsilon \langle 1, 1, H', r_L\rangle \\ r \notin \mathrm{dom}(H')\end{array}}{\langle 1, 1, H, !\,e\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle} \tag{87}$$

We apply the theorem inductively to the premise:

$$\Gamma' \supseteq \Gamma \tag{88}$$
$$\Gamma' \vdash H' \tag{89}$$
$$\Phi_\emptyset; \Gamma' \vdash r_L : \mathit{ref}^{\varepsilon_1}\ \tau \tag{90}$$
$$\varepsilon \subseteq \Phi_1^\varepsilon \tag{91}$$
$$\tag{92}$$

From (89) and [TLOC] we have $r \in \mathrm{dom}(\Gamma')$. Then (89) gives $r \in \mathrm{dom}(H')$. But we have that $r \notin \mathrm{dom}(H')$, a contradiction. Therefore, there is no derivation for $\langle 1, 1, H, !\, e\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle$ that ends by [DEREF-H-W] when $\Gamma \vdash H$ and $\Phi; \Gamma \vdash e_1\ e_2 : \tau_2$.

**case** [DEREF-L-W] **:**

$$\text{[DEREF-L-W]}\frac{\begin{array}{c}\langle 1, 1, H, e\rangle \longrightarrow_\varepsilon \langle 1, 1, H', r_L\rangle \\ r \in \mathrm{dom}(H') \\ L \notin 1\end{array}}{\langle 1, 1, H, !\, e\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle}$$

It is obvious that this rule cannot be applied, as $L \notin 1$ is tautologically false.

**case** [ASSIGN-H-W] **:**

$$\text{[TASSIGN]}\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \mathit{ref}^{\varepsilon}\ \tau \\ \Phi_2; \Gamma \vdash e_2 : \tau \\ \Phi_3^\varepsilon = \varepsilon \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash e_1 := e_2 : \tau} \tag{93}$$
$$\Gamma \vdash H \tag{94}$$
$$\text{[ASSIGN-H-W]}\frac{\begin{array}{c}\langle 1, 1, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1, 1, H_1, r_L\rangle \\ \langle 1, 1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1, 1, H_2, v\rangle \\ r \notin \mathrm{dom}(H_2)\end{array}}{\langle 1, 1, H, e_1 := e_2\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle} \tag{95}$$

Similarly to the case for [DEREF-H-W], we apply the theorem inductively on the first premise:

$$\Gamma_1 \supseteq \Gamma \tag{96}$$
$$\Gamma_1 \vdash H_1 \tag{97}$$
$$\Phi_\emptyset; \Gamma_1 \vdash r_L : \mathit{ref}^{\varepsilon}\ \tau \tag{98}$$
$$\varepsilon_1 \subseteq \Phi_1^\varepsilon \tag{99}$$

From Lemma A.7 we get $\Phi_2; \Gamma_1 \vdash e_2 : \tau$ and apply the theorem on the second premise:

$$\Gamma_2 \supseteq \Gamma_1 \tag{100}$$
$$\Gamma_2 \vdash H_2 \tag{101}$$
$$\Phi_\emptyset; \Gamma_2 \vdash v : \tau \tag{102}$$
$$\varepsilon_2 \subseteq \Phi_2^\varepsilon \tag{103}$$

From (98) and [TLOC] we have $r \in \mathrm{dom}(\Gamma_1)$. Then from (100) we have that $\mathrm{dom}(\Gamma_1) \subseteq \mathrm{dom}(\Gamma_2)$ and from (101) we get $\mathrm{dom}(\Gamma_2) = \mathrm{dom}(H_2)$. It follows that $r \in \mathrm{dom}(H_2)$. But we have $r \notin \mathrm{dom}(H_2)$ from hypothesis, a contradiction. Therefore, there is no derivation for $\langle 1, 1, H, e_1 := e_2\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle$ that ends by [ASSIGN-H-W] when $\Gamma \vdash H$ and $\Phi; \Gamma \vdash e_1 := e_2 : \tau$.

**case** [ASSIGN-L-W] **:**

$$\text{[ASSIGN-L-W]}\frac{\begin{array}{c}\langle 1, 1, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1, 1, H_1, r_L\rangle \\ \langle 1, 1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1, 1, (H_2, r \mapsto v'), v\rangle \\ L \notin 1\end{array}}{\langle 1, 1, H, e_1 := e_2\rangle \longrightarrow_\emptyset \langle 1, 1, H, \mathbf{err}\rangle}$$

It is obvious that this rule cannot be applied, as $L \notin 1$ is tautologically false.

□

**Lemma A.3** (Weakening of evaluation sub-derivations)**.** *Given a derivation $\langle \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle \alpha', \omega', H', v\rangle$, there exists a derivation $\langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, e\rangle \longrightarrow_\varepsilon \langle \alpha' \cup \alpha_w, \omega' \cup \omega_w, H', v\rangle$,*

*Proof.* Proof by induction on the derivation.

**case** [ID] **:**

Given

$$[\textsc{Id}]\frac{}{\langle \alpha, \omega, H, v\rangle \longrightarrow_{\emptyset} \langle \alpha, \omega, H, v\rangle}$$

then we can apply [ID] again:

$$[\textsc{Id}]\frac{}{\langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, v\rangle \longrightarrow_{\emptyset} \langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, v\rangle}$$

**case** [CALL] **:**

$$[\textsc{Call}]\frac{\begin{array}{c}\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, \lambda x.e\rangle \\ \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v_2\rangle \\ \langle \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle \alpha', \omega', H', v\rangle\end{array}}{\langle \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle \alpha', \omega', H', v\rangle}$$

Assuming the lemma holds for the premises:

$$\langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1 \cup \alpha_w, \omega_1 \cup \omega_w, H_1, \lambda x.e\rangle \tag{104}$$

$$\langle \alpha_1 \cup \alpha_w, \omega_1 \cup \omega_w, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2 \cup \alpha_w, \omega_2 \cup \omega_w, H_2, v_2\rangle \tag{105}$$

$$\langle \alpha_2 \cup \alpha_w, \omega_2 \cup \omega_w, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle \alpha' \cup \alpha_w, \omega' \cup \omega_w, H', v\rangle \tag{106}$$

From these, we can apply [CALL] again to get the wanted

$$\langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle \alpha' \cup \alpha_w, \omega' \cup \omega_w, H', v\rangle$$

**case** [REF] **:**

$$[\textsc{Ref}]\frac{\begin{array}{c}\langle \alpha, \omega, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', v\rangle \\ r \notin \mathrm{dom}(H')\end{array}}{\langle \alpha, \omega, H, \mathsf{ref}^{L}\ e\rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', (H', r \mapsto v), r_L\rangle}$$

Assuming the lemma holds for the premise:

$$\langle \alpha \cup \alpha_w, \omega \cup \alpha_w, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha' \cup \alpha_w, \omega' \cup \alpha_w, H', v\rangle$$

we can then apply [REF] again to get

$$\langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, \mathsf{ref}^{L}\ e\rangle \longrightarrow_{\varepsilon} \langle \alpha' \cup \alpha_w, \omega' \cup \omega_w, (H, r \mapsto v), r_L\rangle$$

**case** [DEREF] **:**

$$[\textsc{Deref}]\frac{\begin{array}{c}\langle \alpha, \omega, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega' \cup \{L\}, H', r_L\rangle \\ r \in \mathrm{dom}(H')\end{array}}{\langle \alpha, \omega, H, !\, e\rangle \longrightarrow_{\varepsilon \cup \{L\}} \langle \alpha' \cup \{L\}, \omega', H', H'(r)\rangle}$$

Assuming the lemma holds for the premise we have:

$$\langle \alpha \cup \alpha_w, \omega \cup \omega_w, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha' \cup \alpha_w, \omega' \cup \{L\} \cup \omega_w, H', r_L\rangle$$

Then we can apply [DEREF] again to get the weakened derivation.

**case** [ASSIGN] **:**

**case** [IF-T] **:**

**case** [IF-F] **:**

**case** [LET] **:**

These cases are similar.

□

**Lemma A.4** (Canonical Derivation)**.** *If and only if* $\langle 1, 1, H, e\rangle \longrightarrow_{\varepsilon} \langle 1, 1, H', v\rangle$ *then there exists a derivation* $\langle \emptyset, \omega, H, e\rangle \longrightarrow_{\varepsilon} \langle \alpha, \emptyset, H', v\rangle$*, and also* $\omega = \alpha = \varepsilon$*.*

*Proof.* The only-if case trivially follows from A.3 by adding 1 to both $\alpha$ and $\omega$ in the given derivation. We prove the if case by induction on the derivation:

**case** [ID] **:**

Given

$$[\textsc{Id}]\frac{}{\langle 1, 1, H, v\rangle \longrightarrow_{\emptyset} \langle 1, 1, H, v\rangle}$$

then it is also the case that

$$[\textsc{Id}]\frac{}{\langle \emptyset, \emptyset, H, v\rangle \longrightarrow_{\emptyset} \langle \emptyset, \emptyset, H, v\rangle}$$

**case** [CALL] **:**

$$[\text{CALL}]\frac{\begin{array}{c}\langle 1,1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1,1,H_1,\lambda x.e\rangle \\ \langle 1,1,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1,1,H_2,v_2\rangle \\ \langle 1,1,H_2,e[x\mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle 1,1,H',v\rangle\end{array}}{\langle 1,1,H,e_1\ e_2\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2\cup\varepsilon_3} \langle 1,1,H',v\rangle}$$

Assuming the lemma holds for the premises:

$$\langle \emptyset,\varepsilon_1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle \varepsilon_1,\emptyset,H_1,\lambda x.e\rangle \tag{107}$$
$$\langle \emptyset,\varepsilon_2,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle \varepsilon_2,\emptyset,H_2,v_2\rangle \tag{108}$$
$$\langle \emptyset,\varepsilon_3,H_2,e[x\mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle \varepsilon_3,\emptyset,H',v\rangle \tag{109}$$

Then from Lemma A.3 we get:

$$\langle \emptyset,\varepsilon_1\cup\varepsilon_2\cup\varepsilon_3,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle \varepsilon_1,\varepsilon_2\cup\varepsilon_3,H_1,\lambda x.e\rangle \tag{110}$$
$$\langle \varepsilon_1,\varepsilon_2\cup\varepsilon_3,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle \varepsilon_1\cup\varepsilon_2,\varepsilon_3,H_2,v_2\rangle \tag{111}$$
$$\langle \varepsilon_1\cup\varepsilon_2,\varepsilon_3,H_2,e[x\mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle \varepsilon_1\cup\varepsilon_2\cup\varepsilon_3,\emptyset,H',v\rangle \tag{112}$$

Then we can apply [CALL] again to get

$$\langle \emptyset,\varepsilon_1\cup\varepsilon_2\cup\varepsilon_3,H,e_1\ e_2\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2\cup\varepsilon_3} \langle \varepsilon_1\cup\varepsilon_2\cup\varepsilon_3,\emptyset,H',v\rangle$$

**case** [REF] **:**

$$[\text{REF}]\frac{\begin{array}{c}\langle 1,1,H,e\rangle \longrightarrow_{\varepsilon} \langle 1,1,H',v\rangle \\ r\notin \mathrm{dom}(H')\end{array}}{\langle 1,1,H,\mathsf{ref}^L\ e\rangle \longrightarrow_{\varepsilon} \langle 1,1,(H',r\mapsto v),r_L\rangle}$$

Assuming the lemma holds for the premise, we can apply [REF] again and get

$$\langle \emptyset,\varepsilon,H,\mathsf{ref}^L\ e\rangle \longrightarrow_{\varepsilon} \langle \varepsilon,\emptyset,(H,r\mapsto v),r_L\rangle$$

**case** [DEREF] **:**

$$[\text{DEREF}]\frac{\begin{array}{c}\langle 1,1,H,e\rangle \longrightarrow_{\varepsilon} \langle 1,1\cup\{L\},H',r_L\rangle \\ r\in \mathrm{dom}(H')\end{array}}{\langle 1,1,H,!\,e\rangle \longrightarrow_{\varepsilon\cup\{L\}} \langle 1\cup\{L\},1,H',H'(r)\rangle}$$

We write $1\cup\{L\}$ for clarity in the premise, however $L\in 1$ so $1\cup\{L\}=1$. Assuming the lemma holds for the premise we have:

$$\langle \emptyset,\varepsilon,H,e\rangle \longrightarrow_{\varepsilon} \langle \varepsilon,\emptyset,H',r_L\rangle$$

We apply Lemma A.3 to get

$$\langle \emptyset,\varepsilon\cup\{L\},H,e\rangle \longrightarrow_{\varepsilon} \langle \varepsilon,\emptyset\cup\{L\},H',r_L\rangle$$

Then, we can apply [DEREF] again to get:

$$\langle \emptyset,\varepsilon\cup\{L\},H,!\,e\rangle \longrightarrow_{\varepsilon\cup\{L\}} \langle \varepsilon\cup\{L\},\emptyset,H',H'(r)\rangle$$

**case** [ASSIGN] **:**

$$[\text{ASSIGN}]\frac{\begin{array}{c}\langle 1,1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1,1,H_1,r_L\rangle \\ \langle 1,1,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1,1\cup\{L\},(H_2,r\mapsto v'),v\rangle\end{array}}{\langle 1,1,H,e_1:=e_2\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2\cup\{L\}} \langle 1\cup\{L\},1,(H_2,r\mapsto v),v\rangle}$$

where obviously $1\cup\{L\}=1$, since $L\in 1$. Assuming the lemma holds for the premises we get:

$$\langle \emptyset,\varepsilon_1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle \varepsilon_1,\emptyset,H_1,r_L\rangle \tag{113}$$
$$\langle \emptyset,\varepsilon_2,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle \varepsilon_2,\emptyset,(H_2,r\mapsto v'),v\rangle \tag{114}$$

Applying Lemma A.3 we get:

$$\langle \emptyset,\varepsilon_1\cup\varepsilon_2\cup\{L\},H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle \varepsilon_1,\varepsilon_2\cup\{L\},H_1,r_L\rangle \tag{115}$$
$$\langle \varepsilon_1,\varepsilon_2\cup\{L\},H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle \varepsilon_1\cup\varepsilon_2,\{L\},(H_2,r\mapsto v'),v\rangle \tag{116}$$

Then we can apply [ASSIGN] again to get:

$$\langle \emptyset,\varepsilon_1\cup\varepsilon_2\cup\{L\},H,e_1:=e_2\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2\cup\{L\}} \langle \varepsilon_1\cup\varepsilon_2\cup\{L\},\emptyset,(H_2,r\mapsto v),v\rangle$$

**case** [IF-T] **:**

$$[\text{IF-T}]\frac{\begin{array}{c}\langle 1,1,H,e_1\rangle \longrightarrow_{\varepsilon_1} \langle 1,1,H_1,v_1\rangle \\ v_1=0 \\ \langle 1,1,H_1,e_2\rangle \longrightarrow_{\varepsilon_2} \langle 1,1,H_2,v\rangle\end{array}}{\langle 1,1,H,\mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_{\varepsilon_1\cup\varepsilon_2} \langle 1,1,H_2,v\rangle}$$

Assuming the lemma holds for the premises we have:

$$\langle \emptyset, \varepsilon_1, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \varepsilon_1, \emptyset, H_1, v_1 \rangle \tag{117}$$
$$\langle \emptyset, \varepsilon_2, H_1, e_2 \rangle \longrightarrow_{\varepsilon_2} \langle \varepsilon_2, \emptyset, H_2, v \rangle \tag{118}$$

We can transform these with Lemma A.3 to:

$$\langle \emptyset, \varepsilon_1 \cup \varepsilon_2, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \varepsilon_1, \varepsilon_2, H_1, v_1 \rangle \tag{119}$$
$$\langle \varepsilon_1, \varepsilon_2, H_1, e_2 \rangle \longrightarrow_{\varepsilon_2} \langle \varepsilon_1 \cup \varepsilon_2, \emptyset, H_2, v \rangle \tag{120}$$

then we apply [IF-T] again to get:

$$\langle \emptyset, \varepsilon_1 \cup \varepsilon_2, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle \varepsilon_1 \cup \varepsilon_2, \emptyset, H_2, v \rangle$$

**case** [IF-F] **:**
Similar to the above.
**case** [LET] **:**
Similar to [CALL].

□

**Lemma A.5** (Adequacy of Semantics)**.** *If*

$$\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', v \rangle$$

*then*

1. $\alpha' = \alpha \cup \varepsilon$
2. $\omega = \omega' \cup \varepsilon$

*Proof.* Proof by induction on the evaluation derivation.

**case** [ID] **:**

$$[\text{ID}]\frac{}{\langle \alpha, \omega, H, v \rangle \longrightarrow_{\emptyset} \langle \alpha, \omega, H, v \rangle}$$

Obviously, $\alpha = \alpha \cup \emptyset, \omega = \omega \cup \emptyset$.
**case** [CALL] **:**

$$[\text{CALL}]\frac{\begin{array}{c}\langle \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, \lambda x.e \rangle \\ \langle \alpha_1, \omega_1, H_1, e_2 \rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v_2 \rangle \\ \langle \alpha_2, \omega_2, H_2, e[x \mapsto v_2] \rangle \longrightarrow_{\varepsilon_3} \langle \alpha', \omega', H', v \rangle\end{array}}{\langle \alpha, \omega, H, e_1\ e_2 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle \alpha', \omega', H', v \rangle}$$

Assuming the lemma holds for the premises, we have:

$$\alpha_1 = \alpha \cup \varepsilon_1 \tag{121}$$
$$\alpha_2 = \alpha_1 \cup \varepsilon_2 \tag{122}$$
$$\alpha' = \alpha_2 \cup \varepsilon_3 \tag{123}$$
$$\omega = \omega_1 \cup \varepsilon_1 \tag{124}$$
$$\omega_1 = \omega_2 \cup \varepsilon_2 \tag{125}$$
$$\omega_2 = \omega' \cup \varepsilon_3 \tag{126}$$

From (123), (122), (121) we have $\alpha' = \alpha_2 \cup \varepsilon_3 = (\alpha_1 \cup \varepsilon_2) \cup \varepsilon_3 = \alpha \cup \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$.
Similarly, from (124), (125), (126) we get: $\omega = \omega' \cup \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$.
**case** [REF] **:**

$$[\text{REF}]\frac{\begin{array}{c}\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', v \rangle \\ r \notin \mathrm{dom}(H')\end{array}}{\langle \alpha, \omega, H, \mathsf{ref}^L\ e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', (H', r \mapsto v), r_L \rangle}$$

This case is trivially proven by induction hypothesis for the premise.
**case** [DEREF] **:**

$$[\text{DEREF}]\frac{\begin{array}{c}\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega' \cup \{L\}, H', r_L \rangle \\ r \in \mathrm{dom}(H')\end{array}}{\langle \alpha, \omega, H, !\, e \rangle \longrightarrow_{\varepsilon \cup \{L\}} \langle \alpha' \cup \{L\}, \omega', H', H'(r) \rangle}$$

From induction hypothesis we have:

$$\alpha' = \alpha \cup \varepsilon \tag{127}$$
$$\omega = (\omega' \cup \{L\}) \cup \varepsilon \tag{128}$$

By adding $\{L\}$ to both sides of the above equations and parenthesizing for readability, we get

$$\alpha' \cup \{L\} = \alpha \cup (\varepsilon \cup \{L\}) \tag{129}$$
$$\omega = \omega' \cup (\varepsilon \cup \{L\}) \tag{130}$$

which proves the case.

**case** [ASSIGN] **:**

$$[\textsc{Assign}]\frac{\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, r_L\rangle \quad \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2 \cup \{L\}, (H_2, r \mapsto v'), v\rangle}{\langle \alpha, \omega, H, e_1 := e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \{L\}} \langle \alpha_2 \cup \{L\}, \omega_2, (H_2, r \mapsto v), v\rangle}$$

From the induction hypothesis we have:

$$\alpha_1 = \alpha \cup \varepsilon_1 \tag{131}$$
$$\alpha_2 = \alpha_1 \cup \varepsilon_2 \tag{132}$$
$$\omega = \omega_1 \cup \varepsilon_1 \tag{133}$$
$$\omega_1 = (\omega_2 \cup \{L\}) \cup \varepsilon_2 \tag{134}$$

From the first two we have $\alpha_2 = \alpha \cup \varepsilon_1 \cup \varepsilon_2$ therefore $\alpha_2 \cup \{L\} = \alpha \cup \varepsilon_1 \cup \varepsilon_2 \cup \{L\}$.

From the second two we have $\omega = \omega_2 \cup \varepsilon_1 \cup \varepsilon_2 \cup \{L\}$.

**case** [IF-T] **:**

$$[\textsc{If-T}]\frac{\langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, v_1\rangle \quad v_1 = 0 \quad \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v\rangle}{\langle \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle \alpha_2, \omega_2, H_2, v\rangle}$$

From induction on the premises we have

$$\alpha_1 = \alpha \cup \varepsilon_1 \tag{135}$$
$$\alpha_2 = \alpha_1 \cup \varepsilon_2 \tag{136}$$
$$\omega = \omega_1 \cup \varepsilon_1 \tag{137}$$
$$\omega_1 = \omega_2 \cup \varepsilon_2 \tag{138}$$

The first two give $\alpha_2 = \alpha \cup \varepsilon_1 \cup \varepsilon_2$

The last two give $\omega = \omega_2 \cup \varepsilon_1 \cup \varepsilon_2$

**case** [IF-F] **:**

Similar to the above.

**case** [LET] **:**

Similar to [CALL].

□

**Lemma A.6** (Substitution)**.** *If*

$$\Phi; \Gamma, x : \tau \vdash e : \tau'$$
$$\Phi_\emptyset; \Gamma \vdash v : \tau$$

*then*

$$\Phi; \Gamma \vdash e[x \mapsto v] : \tau'$$

*Proof.* Proof is straightforward by induction on the typing derivation. □

**Lemma A.7** (Weakening of environment)**.** *If* $\Phi; \Gamma \vdash e : \tau$ *and* $\Gamma' \supseteq \Gamma$ *then* $\Phi; \Gamma' \vdash e : \tau$.

*Proof.* Proof is straightforward by induction on the typing derivation. □

**Definition A.8** (Canonical typing)**.** *We say that a typing derivation is canonical when*

1. *It ends with [TSub] and the rule above [TSub] is not [TSub] again*
2. *All sub-derivations of the rule above [TSub] are canonical.*

**Lemma A.9** (Construct canonical typing)**.** *If there exists a typing derivation proving the judgment* $\Phi; \Gamma \vdash e : \tau$ *then there exists a canonical typing derivation that proves it as well.*

*Proof.* Trivial. □

**Definition A.10** (Substitution)**.** *Given $T_e :: [\Phi_e; \Gamma, x : \tau \vdash e : \tau']$ and a canonical typing derivation $T_v :: [\Phi_\emptyset; \Gamma \vdash v : \tau]$, we define a substitution algorithm*

$$\textsc{Subst}\left([T_e], [T_v]\right)$$

*that constructs a canonical typing derivation for*

$$\Phi_e; \Gamma \vdash e[x \mapsto v] : \tau'$$

*based on the substitution lemma.*

$$\textsc{Subst}\left([\Phi; \Gamma, x : \tau \vdash n : \mathit{int}], [\Phi_\emptyset; \Gamma \vdash v : \tau]\right) = \left(\begin{array}{c} \textsc{[TInt]}\ \dfrac{}{\Phi_\emptyset; \Gamma \vdash n : \mathit{int}} \\ \Phi_\emptyset \leq \Phi \\ \mathit{int} \leq \mathit{int} \\ \textsc{[TSub]}\ \dfrac{}{\Phi; \Gamma \vdash n : \mathit{int}} \end{array}\right)$$

$$\textsc{Subst}\left([\Phi; \Gamma, x : \tau \vdash x : \tau], \left[\begin{array}{c} T'_v :: \Phi_\emptyset; \Gamma \vdash v : \tau' \\ \Phi_\emptyset \leq \Phi_\emptyset \\ \tau' \leq \tau \\ \textsc{[TSub]}\ \dfrac{}{T_v :: \Phi_\emptyset; \Gamma \vdash v : \tau} \end{array}\right]\right) = \left(\begin{array}{c} T'_v :: \Phi_\emptyset; \Gamma \vdash v : \tau' \\ \Phi_\emptyset \leq \Phi \\ \tau' \leq \tau \\ \textsc{[TSub]}\ \dfrac{}{\Phi; \Gamma \vdash v : \tau} \end{array}\right)$$

$$\textsc{Subst}\left(\left[\textsc{[TVar]}\ \frac{(\Gamma, x : \tau)(y) = \tau'}{\Phi_\emptyset; \Gamma, x : \tau \vdash y : \tau'}\right], [\Phi_\emptyset; \Gamma \vdash v : \tau]\right) \textit{ where } x \neq y = \left(\begin{array}{c} \textsc{[TVar]}\ \dfrac{\Gamma(y) = \tau'}{\Phi_\emptyset; \Gamma \vdash y : \tau'} \\ \Phi_\emptyset \leq \Phi_\emptyset \\ \tau' \leq \tau' \\ \textsc{[TSub]}\ \dfrac{}{\Phi_\emptyset; \Gamma \vdash y : \tau'} \end{array}\right)$$

$$\textsc{Subst}\left(\left[\begin{array}{c} T_1 :: \Phi_1; \Gamma, x : \tau \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ T_2 :: \Phi_2; \Gamma, x : \tau \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi \\ \textsc{[TApp]}\ \dfrac{}{\Phi; \Gamma, x : \tau \vdash e_1\ e_2 : \tau_2} \end{array}\right], [T_v :: \Phi_\emptyset; \Gamma \vdash v : \tau]\right) = \left(\begin{array}{c} \textsc{Subst}\left([T_1], [T_v]\right) \\ \textsc{Subst}\left([T_2], [T_v]\right) \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi \\ \textsc{[TApp]}\ \dfrac{}{\Phi; \Gamma \vdash e_1\ e_2[x \mapsto v] : \tau_2} \end{array}\right)$$

*The other cases are similar and follow the substitution lemma.*

**Definition A.11** (Weakening)**.** *Similarly to substitution, we define a weakening function, that constructs a weakened typing derivation using a given $\Gamma'$ (following the structure of the proof of the weakening lemma):*

$$\textsc{Weaken}\left([\Phi; \Gamma \vdash e : \tau], \Gamma'\right) = \left(\Phi; \Gamma' \vdash e : \tau\right)$$

*Given $\Gamma' \supseteq \Gamma$, we define:*

$$\textsc{Weaken}\left([\Phi; \Gamma \vdash n : \mathit{int}], \Gamma'\right) = \left(\textsc{[TInt]}\ \frac{}{\Phi; \Gamma' \vdash n : \mathit{int}}\right)$$

$$\textsc{Weaken}\left(\left[\textsc{[TVar]}\ \frac{\Gamma(x) = \tau}{\Phi; \Gamma \vdash x : \tau}\right], \Gamma'\right) = \left(\textsc{[TVar]}\ \frac{\Gamma'(x) = \tau}{\Phi; \Gamma' \vdash x : \tau}\right)$$

$$\textsc{Weaken}\left(\left[\begin{array}{c} T_1 :: \Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi \\ \textsc{[TApp]}\ \dfrac{}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2} \end{array}\right], \Gamma'\right) = \left(\begin{array}{c} \textsc{Weaken}\left([T_1], \Gamma'\right) \\ \textsc{Weaken}\left([T_2], \Gamma'\right) \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi \\ \textsc{[TApp]}\ \dfrac{}{\Phi; \Gamma' \vdash e_1\ e_2 : \tau_2} \end{array}\right)$$

*The other cases are similar.*

$$[\text{ID-A}]\frac{}{\langle T, \alpha, \omega, H, v\rangle \longrightarrow_\emptyset \langle T', \alpha, \omega, H, v\rangle}$$

where

$$[\text{TSUB}]\frac{\begin{array}{c}\Phi_\emptyset; \Gamma \vdash v : \tau' \\ \tau' \leq \tau \\ \Phi_\emptyset \leq \Phi\end{array}}{T :: \Phi; \Gamma \vdash v : \tau}$$

and

$$T' :: \Phi_\emptyset; \Gamma \vdash v : \tau$$

**Figure 13.** Typed operational semantics for values

***Soundness proof strategy*** To prove the soundness of the contextual effect system, we must show that the effect $\Phi$ of a term $e$ approximates the trace $\alpha$ and promise $\omega$ of its evaluation. Note, however, that as the program reduces to a value, individual subterms might change through substitutions. It is therefore not always obvious which $\Phi$ in the typing derivation for the original term corresponds to a subterm produced during evaluation. To make this connection explicit, we define a *typed operational semantics* that annotates each state in the evaluation with a typing derivation. Our semantics is "natural," in the sense that as subterms are modified by substitutions, our semantics "preserves" the $\Phi$ associated with them.

Note that since the terms might change during evaluation, the typing derivations that we use to annotate the evaluation need not be parts of the original typing—but the $\Phi$'s that show up in the new typings always are. By defining this new semantics, we can easily express soundness for contextual effects: the $\Phi$ assigned to an evaluated term by our semantics always overapproximates the $\alpha$ and $\omega$ for the term at runtime. To show the soundness property is not vacuous, we also need to show that we can always construct such a typed operational semantics derivation, given any ordinary evaluation derivation and typing derivation.

***Typed operational semantics*** Typed evaluations have the form:

$$\langle T, \alpha, \omega, H, e\rangle \longrightarrow_\emptyset \langle T', \alpha', \omega', H', v\rangle$$

where $T$ is a canonical typing derivation for the expression $e$ that is evaluated:

$$\Phi; \Gamma \vdash e : \tau$$

and $T'$ is a canonical typing derivation for the result of the evaluation:

$$\Phi_\emptyset; \Gamma' \vdash v : \tau$$

Since $T$ is a canonical derivation, it must end with an application of [TSUB] which follows the "normal" typing of the value $v$. Since $v$ is a value, $T'$ can always use $\Phi_\emptyset$ to type it, which simplifies the rules. The new environment $\Gamma'$ is not in general the same as $\Gamma$, because it might contain extra typings for pointers $r$ that are created during the evaluation of $e$, but we will show that it is always a superset of $\Gamma$. The type of the value is always the same as the type $\tau$ of $e$.

Fig. 13 presents the typed evaluation rule for values. As in the untyped operational semantics, a value $v$ evaluates to itself without changing the state of the heap, or the trace or promise sets. We have added a typing $T$ in the input state and a typing $T'$ in the output state. Note that we list the constraints on typing derivations $T$ and $T'$ after the rule, even though they are actually premises of the rule, to improve readability and reduce the complexity of the presentation. We follow this practice for the rest of the annotated evaluation rules, writing the constraints on typing derivations $T$ that annotate states after the rule as side conditions.

A more interesting case is the rule for typed semantics of the evaluation of function calls [CALL-A], shown in Fig. 14. As before, we annotate the first and last state of each evaluation (both in the conclusion and the premises of the rule) with a typing. For the conclusion, we require the typing $T$ of the application to be canonical (ending with [TSUB], followed by [TAPP]). We require the typing for the evaluation of $e_1$ in the first premise to be the same as in the premise of $T$, this way forcing $\Phi_1$ to be the same between the typing of $e_1$ in the premise and the typing of $e_1$ as a subterm of the conclusion. Note that this constraint is essentially the definition of which $\Phi$ in the typing of the superterm is the "correct" one to use for a subterm. In other words, this constraint specifies that $\Phi_1$ in the typing $T$ is the effect of the evaluation of $e_1$. This way, we assign static effects $\Phi$ from the static typing of a term to the evaluations of its sub-terms. We can then prove that this assignment is indeed sound, i.e. the $\Phi_1$ we selected for the evaluation of $e_1$ provides a sound approximation for the actual contextual effect of the evaluation of $e_1$.

We require the typing $T_1'$ of the result of the first premise to be canonical (ending with [TSUB]) followed by [TLAM] since the result is a lambda-term. The typing $T_1'$ uses environment $\Gamma'$ which will be a superset of $\Gamma$, possibly with extra bindings. We annotate the second premise of the typed evaluation with a typing $T_2$, constructed by weakening $T_1$ to use $\Gamma'$. The partial function $\text{WEAKEN}\,([]\,,\,)$ is only applicable when $\Gamma' \supseteq \Gamma$. We cannot directly use $T_2$ to annotate the initial state of the second premise, because we need to maintain the invariant that the environment in each state types all the locations of the heap at that state. For that reason we constrain $T_2'$ to be a weakened version of $T_2$, and later prove that $\Gamma'$ is always a superset of $\Gamma$ and therefore the weakening is well defined and $T_2'$ is a valid derivation. In any case $T_2$ and $T_2'$ share the same $\Phi_2$ from the original typing, which we later prove that correctly approximates the contextual effects of the evaluation in the second premise.

As before, we annotate the resulting state of the second premise with a canonical derivation $T_2''$, which types the resulting value $v_2$ under environment $\Gamma_2$ to produce the same type $\tau_1$ as $T_2$. As before, we do not need to constrain $\Gamma_2$, since we can prove that $\Gamma_2 \supseteq \Gamma_1$.

$$\text{[CALL-A]}\frac{\begin{array}{c}\langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_1', \alpha_1, \omega_1, H_1, \lambda x.e\rangle \\ \langle T_2', \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle T_2'', \alpha_2, \omega_2, H_2, v_2\rangle \\ \langle T_3, \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle\end{array}}{\langle T, \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle}$$

where

$$\text{[TSUB]}\frac{\text{[TAPP]}\ \dfrac{\begin{array}{c}T_1 :: \Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2} \quad \begin{array}{c}\tau_2 \leq \tau' \\ \Phi \leq \Phi'\end{array}}{T :: \Phi'; \Gamma \vdash e_1\ e_2 : \tau'}$$

and

$$\text{[TSUB]}\frac{\text{[TLAM]}\ \dfrac{T_f :: \Phi_f'; \Gamma_1, x : \tau_1' \vdash e : \tau_2'}{\Phi_\emptyset; \Gamma_1 \vdash \lambda x.e : \tau_1' \longrightarrow^{\Phi_f'} \tau_2'} \quad \begin{array}{c}\tau_1' \longrightarrow^{\Phi_f'} \tau_2' \leq \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ \Phi_\emptyset \leq \Phi_\emptyset\end{array}}{T_1' :: \Phi_\emptyset; \Gamma_1 \vdash \lambda x.e : \tau_1 \longrightarrow^{\Phi_f} \tau_2}$$

and

$$T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right)$$

and

$$\text{[TSUB]}\frac{\begin{array}{c}T_{v_2} :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1'' \\ \tau_1'' \leq \tau_1 \\ \Phi_\emptyset \leq \Phi_\emptyset\end{array}}{T_2'' :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1}$$

and

$$T_f' = \textsc{Weaken}\left([T_f], (\Gamma_2, x : \tau_1')\right)$$

and

$$T_3 \text{ is } \textsc{Subst}\left(\left[\text{[TSUB]}\frac{\begin{array}{c}T_f' :: \Phi_f'; \Gamma_2, x : \tau_1' \vdash e : \tau_2' \\ \tau_2' \leq \tau_2 \leq \tau' \\ \Phi_f' \leq \Phi_f\end{array}}{\Phi_f; \Gamma_2, x : \tau_1' \vdash e : \tau'}\right], \left[\text{[TSUB]}\frac{\begin{array}{c}T_{v_2} :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1'' \\ \tau_1'' \leq \tau_1 \leq \tau_1' \\ \Phi_\emptyset \leq \Phi_\emptyset\end{array}}{\Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1'}\right]\right)$$

and

$$T_v :: \Phi_\emptyset; \Gamma_3 \vdash v : \tau'$$

**Figure 14.** Typed operational semantics for function call

Interestingly, the third premise of the [CALL] evaluation does not reduce a term that exists in the superterm, but instead the term that results from substituting $x$ with $v$ in $e$, where $e$ is the body of the lambda term of the first premise. Therefore, there is no straightforward way to use an existing typing derivation from the sub-derivations of $T$ to annotate the third evaluation. Instead, we construct the correct typing derivation from the premises of $T_f$ (the typing of the lambda term) and $T_2''$ (the typing of $v_2$). We define a partial function $\textsc{Subst}\left([T], [T']\right)$ that constructs a typing for $e[v \mapsto x]$ given appropriate typings for $e$ and $v$, similarly to the weakening function. We later prove that it can be applied and will construct a typing for the term under $\Phi_f$, the effect used to type the body of the function in $T_1'$. Note that this is one of the few cases where we need the typing that annotates the result of a typed evaluation. Finally, we annotate the result of the third premise of the evaluation with $T_v$, a typing of $v$ under $\Phi_\emptyset$ and $\Gamma_3$, which we will show is a superset of $\Gamma_2$, to give the same type $\tau'$ as $T$.

**Definition A.12** (Consistent type state)**.** *A typed operational semantics state*

$$\langle T, \alpha, \omega, H, e\rangle$$

*where*

$$T :: \Phi; \Gamma \vdash e : \tau$$

*is consistent, written*

$$\vdash \langle T, \alpha, \omega, H, e\rangle$$

$$\text{[REF-A]}\ \frac{\langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle \quad r \notin \mathrm{dom}(H')}{\langle T, \alpha, \omega, H, \mathsf{ref}^L\ e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega', (H', r \mapsto v), r_L\rangle}$$

where

$$\text{[TREF]}\ \frac{T_1 :: \Phi; \Gamma \vdash e : \tau}{\Phi; \Gamma \vdash \mathsf{ref}^L\ e : \mathit{ref}^{\{L\}}\ \tau}$$

$$\text{[TSUB]}\ \frac{\mathit{ref}^{\{L\}}\ \tau \leq \mathit{ref}^{\varepsilon}\ \tau' \quad \Phi \leq \Phi'}{T :: \Phi'; \Gamma \vdash \mathsf{ref}^L\ e : \mathit{ref}^{\varepsilon}\ \tau'}$$

and

$$T_v :: \Phi_\emptyset; \Gamma' \vdash v : \tau$$

and

$$\text{[TSUB]}\ \frac{\Phi_\emptyset; (\Gamma_1, r \mapsto \tau) \vdash r_L : \mathit{ref}^{\{L\}}\ \tau \quad \Phi_\emptyset \leq \Phi_\emptyset \quad \mathit{ref}^{\{L\}}\ \tau \leq \mathit{ref}^{\varepsilon}\ \tau'}{T_r :: \Phi_\emptyset; (\Gamma_1, r \mapsto \tau) \vdash r_L : \mathit{ref}^{\{\varepsilon\}}\ \tau'}$$

**Figure 15.** Typed operational semantics for reference

$$\text{[DEREF-A]}\ \frac{\langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega' \cup \{L\}, H', r_L\rangle \quad r \in \mathrm{dom}(H')}{\langle T, \alpha, \omega, H, !\, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle}$$

where

$$\text{[TDEREF]}\ \frac{T_1 :: \Phi_1; \Gamma \vdash e : \mathit{ref}^{\varepsilon}\ \tau' \quad \Phi_2^{\varepsilon} = \varepsilon \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi'}{\Phi'; \Gamma \vdash !\, e : \tau'}$$

$$\text{[TSUB]}\ \frac{\tau' \leq \tau \quad \Phi' \leq \Phi}{T :: \Phi; \Gamma \vdash !\, e : \tau}$$

and

$$\text{[TLOC]}\ \frac{\Gamma'(r) = \tau}{\Phi_\emptyset; \Gamma' \vdash r_L : \mathit{ref}^{\{L\}}\ \tau}$$

$$\text{[TSUB]}\ \frac{\Phi_\emptyset \leq \Phi_\emptyset \quad \mathit{ref}^{\{L\}}\ \tau \leq \mathit{ref}^{\varepsilon}\ \tau'}{T_r :: \Phi_\emptyset; \Gamma' \vdash r_L : \mathit{ref}^{\varepsilon}\ \tau'}$$

and

$$T_v = \Phi_\emptyset; \Gamma' \vdash v : \tau$$

**Figure 16.** Typed operational semantics for dereference

$$\text{[IF-T-A]}\frac{\begin{array}{c}\langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_{v_1}, \alpha_1, \omega_1, H_1, v_1\rangle \\ v_1 = 0 \\ \langle T_2', \alpha_1, \omega_1, H, e_2\rangle \longrightarrow_{\varepsilon_1} \langle T_v, \alpha_2, \omega_2, H_2, v\rangle\end{array}}{\langle T, \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle T_v, \alpha_2, \omega_2, H_2, v\rangle}$$

where

$$\text{[TIF]}\frac{\begin{array}{c}T_1 :: \Phi_1; \Gamma \vdash e_1 : int \\ T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau \\ T_3 :: \Phi_2; \Gamma \vdash e_3 : \tau \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{T :: \Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}$$

and

$$T_{v_1} :: \Phi_\emptyset; \Gamma_1 \vdash v_1 : int$$

and

$$T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right)$$

and

$$T_v :: \Phi_\emptyset; \Gamma_2 \vdash v : \tau$$

$$\text{[IF-F-A]}\frac{\begin{array}{c}\langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_{v_1}, \alpha_1, \omega_1, H_1, v_1\rangle \\ v_1 \neq 0 \\ \langle T_2', \alpha_1, \omega_1, H, e_3\rangle \longrightarrow_{\varepsilon_2} \langle T_v, \alpha_2, \omega_2, H_2, v\rangle\end{array}}{\langle T, \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle T_v, \alpha_2, \omega_2, H_2, v\rangle}$$

where

$$\text{[TIF]}\frac{\begin{array}{c}T_1 :: \Phi_1; \Gamma \vdash e_1 : int \\ T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau \\ T_3 :: \Phi_2; \Gamma \vdash e_3 : \tau \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{T :: \Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}$$

and

$$T_{v_1} :: \Phi_\emptyset; \Gamma_1 \vdash v_1 : int$$

and

$$T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right)$$

and

$$T_v :: \Phi_\emptyset; \Gamma_2 \vdash v : \tau$$

**Figure 17.** Typed operational semantics for conditional

*if*

$$\Gamma \vdash H$$

**Lemma A.13** (Environment grows, types do not)**.** *If*

$$\langle T, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle$$

*and*

$$T :: \Phi; \Gamma \vdash e : \tau$$

*and*

$$T_v :: \Phi'; \Gamma' \vdash v : \tau'$$

*then*

$$\Gamma' \supseteq \Gamma$$

*and*

$$\tau = \tau'$$

*Proof.* Trivial. □

$$\text{[LET-A]}\frac{\langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_{v_1}, \alpha_1, \omega_1, H_1, v_1\rangle \quad \langle T_2', \alpha_1, \omega_1, H_1, e_2[x \mapsto v_1]\rangle \longrightarrow_{\varepsilon_2} \langle T_v, \alpha', \omega', H', v\rangle}{\langle T, \alpha, \omega, H, \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle T_v, \alpha', \omega', H', v\rangle}$$

where

$$\text{[TSUB]}\frac{\text{[TAPP]}\ \dfrac{\begin{array}{c} T_1 :: \Phi_1; \Gamma \vdash e_1 : \tau_1 \\ T_2 :: \Phi_2; \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi' \end{array}}{\Phi'; \Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2} \quad \begin{array}{c} \tau_2 \leq \tau \\ \Phi' \leq \Phi \end{array}}{T :: \Phi; \Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau}$$

and

$$T_{v_1} :: \Phi_\emptyset; \Gamma_1 \vdash v_1 : \tau_1$$

and

$$T_v :: \Phi_\emptyset; \Gamma_2 \vdash v : \tau_2$$

and

$$T_2' = \textsc{Weaken}\left([T_2], (\Gamma_1, x : \tau_1)\right)$$

and

$$T_2'' = \textsc{Subst}\left(\left[T_2'\right], [T_{v_1}]\right)$$

**Figure 18.** Typed operational semantics for let

**Lemma A.14** (Consistent typed states)**.** *If*

$$\langle T, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle$$

*and*

$$\vdash \langle T, \alpha, \omega, H, e\rangle$$

*then*

$$\vdash \langle T_v, \alpha', \omega', H', v\rangle$$

*Proof.* **case** [ID-A] **:**
Given

$$\langle T, \alpha, \omega, H, v\rangle \longrightarrow_\varepsilon \langle T, \alpha, \omega, H, v\rangle$$

and

$$\vdash \langle T, \alpha, \omega, H, v\rangle$$

then obviously

$$\vdash \langle T, \alpha, \omega, H, v\rangle$$

**case** [CALL-A] **:**
Given

$$\text{[CALL-A]}\frac{\langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_1', \alpha_1, \omega_1, H_1, \lambda x.e\rangle \quad \langle T_2', \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle T_2'', \alpha_2, \omega_2, H_2, v_2\rangle \quad \langle T_3, \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle}{\langle T, \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle}$$

where

$$\text{[TSUB]}\frac{\text{[TAPP]}\ \dfrac{\begin{array}{c} T_1 :: \Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi \end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2} \quad \begin{array}{c} \tau_2 \leq \tau' \\ \Phi \leq \Phi' \end{array}}{T :: \Phi'; \Gamma \vdash e_1\ e_2 : \tau'}$$

and

$$\vdash \langle T, \alpha, \omega, H, e_1\ e_2\rangle$$

we have

$$\Gamma \vdash H$$

From that and $T_1$ we have

$$\vdash \langle T_1, \alpha, \omega, H, e_1 \rangle$$

Therefore, by induction we get

$$\vdash \langle T_1', \alpha_1, \omega_1, H_1, \lambda x.e \rangle$$

where

$$T_1' :: \Phi_\emptyset; \Gamma_1 \vdash \lambda x.e : \tau_1 \longrightarrow^{\Phi_f} \tau_2$$

which gives

$$\Gamma_1 \vdash H_1$$

and

$$T_f :: \Phi_f'; \Gamma, x : \tau_1' \vdash e : \tau_2'$$

Also, lemma A.13 gives $\Gamma_1 \supseteq \Gamma$, therefore

$$T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right)$$

gives

$$\vdash \langle T_2', \alpha_1, \omega_1, H_1, e_2 \rangle$$

By induction we get

$$\vdash \langle T_2'', \alpha_2, \omega_2, H_2, v_2 \rangle$$

where

$$T_2'' :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1$$
$$\Gamma_2 \vdash H_2$$

and lemma A.13 gives $\Gamma_2 \supseteq \Gamma_1$, therefore

$$T_f' = \textsc{Weaken}\left(\left[T_f\right], \Gamma_2, x : \tau_1'\right)$$

is

$$T_f' :: \Phi_f'; \Gamma_2, x : \tau_1' \vdash e : \tau_2'$$

From $T_2''$ and $\tau_1 \leq \tau_1'$ we get

$$T_2''' :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1'$$

Then with

$$T_3 = \textsc{Subst}\left(\left[T_f'\right], \left[T_2'''\right]\right)$$

we have

$$\vdash \langle T_3, \alpha_2, \omega_2, H_2, e[x \mapsto v_2] \rangle$$

and by induction

$$\vdash \langle T_v, \alpha', \omega', H', v \rangle$$

**case** [REF-A] **:**
Given

$$[\textsc{Ref-A}]\frac{\langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle \quad r \notin \mathrm{dom}(H')}{\langle T, \alpha, \omega, H, \mathsf{ref}^L\ e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega', (H', r \mapsto v), r_L\rangle}$$

and

$$\vdash \langle T, \alpha, \omega, H, \mathsf{ref}^L\ e \rangle$$

Similarly to the above, we get

$$\vdash \langle T_1, \alpha, \omega, H, e \rangle$$

and by induction

$$\vdash \langle T_v, \alpha', \omega', H', v \rangle$$

where $T_v :: \Phi_\emptyset; \Gamma' \vdash v : \tau$. Then for $\Gamma'' = (\Gamma', r \mapsto \tau\ H'' = (H', r \mapsto v)$ we have $\Gamma'' \vdash H''$ Also $\Gamma'' \supseteq \Gamma'$ and from lemma A.13 we get $\Gamma' \supseteq \Gamma$, therefore

$$\vdash \langle T_r, \alpha', \omega', H'', r_L \rangle$$

**case** Other **:**
The remaining cases are similar.

□

**Lemma A.15** (A typed evaluation derivation exists)**.** *If*

$$T :: \Phi; \Gamma \vdash e : \tau$$

*and*

$$D :: \langle \alpha, \omega, H, e \rangle \longrightarrow_\varepsilon \langle \alpha', \omega', H', v \rangle$$

*and*

$$\vdash \langle \alpha, \omega, H, e \rangle$$

*then there exists $T_v$ such that*

$$\langle T, \alpha, \omega, H, e \rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v \rangle$$

*Proof.* **case** [ID] **:**

Given

$$T :: \Phi; \Gamma \vdash v : \tau$$

and

$$D :: \langle \alpha, \omega, H, v\rangle \longrightarrow_{\emptyset} \langle \alpha, \omega, H, v\rangle$$

From [ID-A] we get

$$\langle T, \alpha, \omega, H, v\rangle \longrightarrow_{\emptyset} \langle T, \alpha, \omega, H, v\rangle$$

**case** [CALL] **:**

The assumptions are:

$$\text{[TSUB]}\dfrac{\text{[TAPP]}\ \dfrac{\begin{array}{c} T_1 :: \Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi \end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2} \\ \begin{array}{c} \tau_2 \leq \tau' \\ \Phi \leq \Phi' \end{array}}{T :: \Phi'; \Gamma \vdash e_1\ e_2 : \tau'}$$

and

$$\text{[CALL]}\dfrac{\begin{array}{c} D_1 :: \langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, \lambda x.e\rangle \\ D_2 :: \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v_2\rangle \\ D_3 :: \langle \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle \alpha', \omega', H', v\rangle \end{array}}{D :: \langle \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle \alpha', \omega', H', v\rangle}$$

and $\Gamma \vdash H$. From $T_1$, $D_1$ and $\Gamma \vdash H$, by induction we have: $E_1 :: \langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_1', \alpha_1, \omega_1, H_1, \lambda x.e\rangle$ where

$$\text{[TSUB]}\dfrac{\text{[TLAM]}\ \dfrac{T_f :: \Phi_f'; \Gamma_1, x : \tau_1' \vdash e : \tau_2'}{\Phi_\emptyset; \Gamma_1 \vdash \lambda x.e : \tau_1' \longrightarrow^{\Phi_f'} \tau_2'} \\ \begin{array}{c} \tau_1' \longrightarrow^{\Phi_f'} \tau_2' \leq \tau_1 \longrightarrow^{\Phi_f} \tau_2 \\ \Phi_\emptyset \leq \Phi_\emptyset \end{array}}{T_1' :: \Phi_\emptyset; \Gamma_1 \vdash \lambda x.e : \tau_1 \longrightarrow^{\Phi_f} \tau_2}$$

is a canonical typing, $\Gamma_1 \supseteq \Gamma$ and $\Gamma_1 \vdash H_1$.
From $T_2$ and $\Gamma_1 \supseteq \Gamma$ we get $T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right)$.
From $T_2'$, $D_2$, and $\Gamma_1 \vdash H_1$, we get by induction: $E_2 :: \langle T_2', \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle T_{v_2}, \alpha_2, \omega_2, H_2, v_2\rangle$ where $T_{v_2} :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1$, $\Gamma_2 \supseteq \Gamma_1$ and $\Gamma_2 \vdash H_2$.
From $\tau_1' \longrightarrow^{\Phi_f'} \tau_2' \leq \tau_1 \longrightarrow^{\Phi_f} \tau_2$ we get $\tau_1 \leq \tau_1'$, $\tau_2' \leq \tau_2$ and $\Phi_f' \leq \Phi_f$, therefore

$$\text{[TSUB]}\dfrac{\begin{array}{c} T_{v_2} :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1 \\ \tau_1 \leq \tau_1' \\ \Phi_\emptyset \leq \Phi_\emptyset \end{array}}{T_2'' :: \Phi_\emptyset; \Gamma_2 \vdash v_2 : \tau_1'}$$

Also, from $\Gamma_2 \supseteq \Gamma_1$ we get a $T_f' = \textsc{Weaken}\left([T_f], \Gamma_2\right)$. Then we construct $T_3' = \textsc{Subst}\left(\left[T_f'\right], \left[T_2''\right]\right)$ such that $T_3' :: \Phi_f'; \Gamma_2 \vdash e[x \mapsto v_2] : \tau_2'$. Finally, from $\tau_2' \leq \tau_2$ we construct $T_3$:

$$\text{[TSUB]}\dfrac{\begin{array}{c} T_3' :: \Phi_f'; \Gamma_2 \vdash e[x \mapsto v_2] : \tau_2' \\ \tau_2' \leq \tau_2 \\ \Phi_f' \leq \Phi_f \end{array}}{T_3 :: \Phi_f; \Gamma_2 \vdash e[x \mapsto v_2] : \tau_2}$$

From the last and induction, we get $\langle T_3, \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle$ where $T_v :: \Phi_\emptyset; \Gamma_3 \vdash v : \tau_2$, $\Gamma_3 \supseteq \Gamma_2$ and $\Gamma_3 \vdash H'$.
So, we can apply [CALL-A] to get

$$\text{[CALL-A]}\dfrac{\begin{array}{c} \langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_1', \alpha_1, \omega_1, H_1, \lambda x.e\rangle \\ \langle T_2, \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle T_{v_2}, \alpha_2, \omega_2, H_2, v_2\rangle \\ \langle T_3, \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle \end{array}}{\langle T, \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle}$$

**case** [REF] **:**

Given

$$\text{[TSUB]}\dfrac{\text{[TREF]}\ \dfrac{T_1 :: \Phi'; \Gamma \vdash e : \tau'}{\Phi; \Gamma \vdash \mathsf{ref}^L\ e : \mathit{ref}^{\{L\}}\ \tau'} \\ \begin{array}{c} \Phi' \leq \Phi \\ \mathit{ref}^{\{L\}}\ \tau' \leq \mathit{ref}^{\varepsilon'}\ \tau \end{array}}{T :: \Phi; \Gamma \vdash \mathsf{ref}^L\ e : \mathit{ref}^{\varepsilon'}\ \tau}$$

and

$$[\textsc{Ref}]\frac{D_1 :: \langle \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle \alpha', \omega', H', v\rangle \quad r \notin \mathrm{dom}(H')}{D :: \langle \alpha, \omega, H, \mathsf{ref}^L\ e\rangle \longrightarrow_\varepsilon \langle \alpha', \omega', (H', r \mapsto v), r_L\rangle}$$

and $\Gamma \vdash H$

By induction, $T_1$ and $D_1$ we have

$$\langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle$$

where $T_v :: \Phi_\emptyset; \Gamma' \vdash e : \tau'$ is a canonial typing, $\Gamma' \supseteq \Gamma$ and $\Gamma' \vdash H'$

We extend $\Gamma'$ and $H'$ to define $\Gamma'' = (\Gamma', r \mapsto \tau')$ and $H'' = (H', r \mapsto v)$ respectively. Then clearly $\Gamma'' \supseteq \Gamma'$ and $\Gamma'' \vdash H''$.

Therefore we can apply [Ref-A], $\langle T, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega', (H', r \mapsto v), r_L\rangle$ where

$$[\textsc{TSub}]\frac{[\textsc{TLoc}]\ \dfrac{\Gamma''(r) = \tau'}{\Phi_\emptyset; \Gamma'' \vdash r_L : ref^{\{L\}}\ \tau'} \quad ref^{\{L\}}\ \tau' \leq ref^{\varepsilon'}\ \tau \quad \Phi_\emptyset \leq \Phi_\emptyset}{T_r :: \Phi_\emptyset; \Gamma'' \vdash r_L : ref^{\{\varepsilon\}}\ \tau}$$

$\Gamma' \supseteq \Gamma$ and $\Gamma'' \vdash H''$

**case** [Deref] **:**

Given

$$[\textsc{TSub}]\frac{[\textsc{TDeref}]\ \dfrac{\Phi_1; \Gamma \vdash e : ref^{\varepsilon_1}\ \tau' \quad \Phi_2^\varepsilon = \varepsilon_1 \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi'}{\Phi'; \Gamma \vdash !\, e : \tau'} \quad \tau' \leq \tau \quad \Phi' \leq \Phi}{\Phi; \Gamma \vdash !\, e : \tau}$$

and

$$[\textsc{Deref}]\frac{D_1 :: \langle \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle \alpha', \omega' \cup \{L\}, H', r_L\rangle \qquad r \in \mathrm{dom}(H')}{D :: \langle \alpha, \omega, H, !\, e\rangle \longrightarrow_{\varepsilon \cup \{L\}} \langle \alpha' \cup \{L\}, \omega', H', H'(r)\rangle}$$

and $\Gamma \vdash H$.

By induction, $T_1$ and $D_1$ we have

$$\langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega' \cup \{L\}, H', r_L\rangle$$

where

$$[\textsc{TSub}]\frac{[\textsc{TLoc}]\ \dfrac{\Gamma'(r) = \tau'}{\Phi_\emptyset; \Gamma' \vdash r_L : ref^{\varepsilon'}\ \tau'} \quad \Phi_\emptyset \leq \Phi_\emptyset \quad ref^{\varepsilon'}\ \tau' \leq ref^{\varepsilon}\ \tau}{T_r :: \Phi_\emptyset; \Gamma' \vdash r_L : ref^{\varepsilon}\ \tau}$$

and $\Gamma' \vdash H'$.

From $\Gamma' \vdash H'$ we have $r \in \mathrm{dom}(H') = \mathrm{dom}(\Gamma')$ and

$$T_v :: \Phi_\emptyset; \Gamma' \vdash H'(r) : \Gamma'(r)$$

Now we can apply [Deref-A]:

$$[\textsc{Deref-A}]\frac{\langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega' \cup \{L\}, H', r_L\rangle \quad r \in \mathrm{dom}(H')}{\langle T, \alpha, \omega, H, !\, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle}$$

**case** [Assign] **:**

Similar to [Deref].

**case** [If-T] **:**

Given

$$[\textsc{If-T}]\frac{D_1 :: \langle \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle \alpha_1, \omega_1, H_1, v_1\rangle \quad v_1 = 0 \quad D_2 :: \langle \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle \alpha_2, \omega_2, H_2, v\rangle}{D :: \langle \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle \alpha_2, \omega_2, H_2, v\rangle}$$

and

$$[\textsc{TIf}]\frac{T_1 :: \Phi_1; \Gamma \vdash e_1 : int \quad T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau \quad T_3 :: \Phi_2; \Gamma \vdash e_3 : \tau \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{T :: \Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}$$

and $\Gamma \vdash H$.
By induction, $D_1$ and $T_1$ we have

$$\langle T_1, \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle T_{v_1}, \alpha_1, \omega_1, H_1, v_1 \rangle$$

where

$$[\text{TSUB}]\dfrac{[\text{TINT}]\ \dfrac{}{\Phi_\emptyset; \Gamma_1 \vdash v_1 : int} \quad \Phi_\emptyset \leq \Phi_\emptyset \quad int \leq int}{T_{v_1} :: \Phi_\emptyset; \Gamma_1 \vdash v_1 : int}$$

and $\Gamma_1 \supseteq \Gamma$ and $\Gamma_1 \vdash H_1$.
We have $\Gamma_1 \supseteq \Gamma$, so we get $T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right) :: \Phi_2; \Gamma_1 \vdash e_2 : \tau$.
By induction, $T_2'$ and $D_2$ we get

$$\langle T_2', \alpha_1, \omega_1, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle T_v, \alpha_2, \omega_2, H_2, v \rangle$$

where

$$T_v :: \Phi_\emptyset; \Gamma_2 \vdash v : \tau$$

and $\Gamma_2 \supseteq \Gamma_1$ and $\Gamma_2 \vdash H_2$.
Then we can apply [IF-T-A] to get

$$[\text{IF-T-A}]\dfrac{\langle T_1, \alpha, \omega, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle T_{v_1}, \alpha_1, \omega_1, H_1, v_1 \rangle \quad v_1 = 0 \quad \langle T_2', \alpha_1, \omega_1, H, e_1 \rangle \longrightarrow_{\varepsilon_1} \langle T_v, \alpha_2, \omega_2, H_2, v \rangle}{\langle T, \alpha, \omega, H, \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2} \langle T_v, \alpha_2, \omega_2, H_2, v \rangle}$$

**case** [IF-F] **:**
Similar to [IF-T].
**case** [LET] **:**
Similar to [CALL].

□

**Lemma A.16** (The typed evaluation derivation is complete w.r.t. the evaluation)**.** *If*

$$E :: \langle T, \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle T_v, \alpha', \omega', H', v \rangle$$

*Then*

$$\langle \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle \alpha', \omega', H', v \rangle$$

*Proof.* Trivial. Sketch: all typed evaluation rules have a corresponding untyped evaluation rule, we can convert each typed evaluation rule to its corresponding untyped by just removing the annotation $T$. □

**Theorem A.17** (Prior and Future Effect Soundness)**.** *If*

$$E :: \langle T, \alpha, \omega, H, e \rangle \longrightarrow_{\varepsilon} \langle T_v, \alpha', \omega', H', v \rangle$$

*where*

$$T :: \Phi; \Gamma \vdash e : \tau$$

*and*

$$\alpha \subseteq \Phi^\alpha$$

*and*

$$\omega' \subseteq \Phi^\omega$$

*then for all sub-derivations $E_i$ of $E$,*

$$E_i :: \langle T_i, \alpha_i, \omega_i, H_i, e_i \rangle \longrightarrow_{\varepsilon} \langle T_{v_i}, \alpha_i', \omega_i', H_i', v_i \rangle$$

*where*

$$T_i :: \Phi_i; \Gamma_i \vdash e_i : \tau_i$$

*it will hold that*

$$\alpha_i \subseteq \Phi_i^\alpha$$

*and*

$$\omega_i' \subseteq \Phi_i^\omega$$

*Proof.* **case** [ID-A] **:**
Given

$$[\text{ID-A}]\dfrac{E :: \langle T, \alpha, \omega, H, v \rangle \longrightarrow_{\emptyset} \langle T, \alpha, \omega, H, v \rangle \quad T :: \Phi; \Gamma \vdash e : \tau \quad \alpha \subseteq \Phi^\alpha \quad \omega' \subseteq \Phi^\omega}{}$$

There are no sub-derivations, therefore the lemma holds vacuously.

**case** [CALL-A] **:**

Given

$$\text{[CALL-A]}\frac{\begin{array}{c}E_1 :: \langle T_1, \alpha, \omega, H, e_1\rangle \longrightarrow_{\varepsilon_1} \langle T_1', \alpha_1, \omega_1, H_1, \lambda x.e\rangle \\ E_2 :: \langle T_2', \alpha_1, \omega_1, H_1, e_2\rangle \longrightarrow_{\varepsilon_2} \langle T_2'', \alpha_2, \omega_2, H_2, v_2\rangle \\ E_3 :: \langle T_3, \alpha_2, \omega_2, H_2, e[x \mapsto v_2]\rangle \longrightarrow_{\varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle\end{array}}{E :: \langle T, \alpha, \omega, H, e_1\ e_2\rangle \longrightarrow_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3} \langle T_v, \alpha', \omega', H', v\rangle}$$

and

$$T :: \Phi; \Gamma \vdash e_1\ e_2 : \tau'$$
$$\alpha \subseteq \Phi^\alpha$$
$$\omega' \subseteq \Phi^\omega$$

From [CALL-A] we have:

$$T_1 :: \Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2$$
$$T_2 :: \Phi_2; \Gamma \vdash e_2 : \tau_1$$
$$T_2' = \textsc{Weaken}\left([T_2], \Gamma_1\right)$$
$$T_3 :: \Phi_f; \Gamma_2 \vdash e[x \mapsto v_2] : \tau'$$

Using Destruction of the typed evaluation (Lemma A.16), we get the corresponding untyped evaluation to $E$. We can then use Weakening of evaluations (Lemma A.3) to relax $\alpha$ and $\omega$ to 1, we can apply Standard Effect soundness (Theorem A.2) to get

$$\varepsilon_1 \subseteq \Phi_1^\varepsilon$$
$$\varepsilon_2 \subseteq \Phi_2^\varepsilon$$
$$\varepsilon_3 \subseteq \Phi_f^\varepsilon$$

From

$$\Phi_1 \rhd \Phi_2 \rhd \Phi_f \hookrightarrow \Phi$$

we get

$$\Phi_1^\alpha = \Phi^\alpha$$
$$\Phi_2^\alpha = \Phi^\alpha \cup \Phi_1^\varepsilon$$
$$\Phi_f^\alpha = \Phi^\alpha \cup \Phi_1^\varepsilon \cup \Phi_2^\varepsilon$$

and

$$\Phi_1^\omega = \Phi^\omega \cup \Phi_2^\varepsilon \cup \Phi_f^\varepsilon$$
$$\Phi_2^\omega = \Phi^\omega \cup \Phi_f^\varepsilon$$
$$\Phi_f^\omega = \Phi^\omega$$

From Traces and Promises (Lemma A.5) we get

$$\alpha_1 = \alpha \cup \varepsilon_1$$
$$\alpha_2 = \alpha \cup \varepsilon_1 \cup \varepsilon_2$$

and

$$\omega_1 = \omega \cup \varepsilon_2 \cup \varepsilon_3$$
$$\omega_2 = \omega \cup \varepsilon_3$$

Therefore, for $E_1$

$$\alpha \subseteq \Phi^\alpha = \Phi_1^\alpha$$
$$\omega_1 = \omega \cup \varepsilon_2 \cup \varepsilon_3 \subseteq \Phi^\omega \cup \Phi_2^\varepsilon \cup \Phi_f^\varepsilon = \Phi_1^\omega$$

We can now apply the lemma inductively on $E_1$ to get that then for all sub-derivations $E_i$ of $E_1$,

$$E_i :: \langle T_i, \alpha_i, \omega_i, H_i, e_i\rangle \longrightarrow_\varepsilon \langle T_{v_i}, \alpha_i', \omega_i', H_i', v_i\rangle$$

where

$$T_i :: \Phi_i; \Gamma_i \vdash e_i : \tau_i$$

it will hold that

$$\alpha_i \subseteq \Phi_i^\alpha$$

and

$$\omega_i' \subseteq \Phi_i^\omega$$

For $E_2$

$$\alpha_1 = \alpha \cup \varepsilon_1 \subseteq \Phi^\alpha \cup \Phi_1^\varepsilon = \Phi_2^\alpha$$
$$\omega_2 = \omega \cup \varepsilon_3 \subseteq \Phi^\omega \cup \Phi_f^\varepsilon = \Phi_2^\omega$$

Similarly to $E_1$, we can now apply induction to get the wanted property for all sub-derivations of $E_2$.
For $E_3$

$$\alpha_2 = \alpha \cup \varepsilon_1 \cup \varepsilon_2 \subseteq \Phi^\alpha \cup \Phi_1^\varepsilon \cup \Phi_2^\varepsilon = \Phi_f^\alpha$$
$$\omega' = \omega \subseteq \Phi^\omega = \Phi_f^\omega$$

As before, we can now apply induction to get the wanted property for all sub-derivations of $E_3$.

**case** [REF-A] **:**

From [REF-A] we have

$$\text{[REF-A]}\frac{\begin{array}{c}E_1 :: \langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle \\ r \notin \mathrm{dom}(H')\end{array}}{E :: \langle T, \alpha, \omega, H, \mathsf{ref}^{\,L}\ e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega', (H', r \mapsto v), r_L\rangle}$$

$$T :: \Phi; \Gamma \vdash \mathsf{ref}^{\,L}\ e : \mathit{ref}^{\,\varepsilon}\ \tau$$

$$\alpha \subseteq \Phi^\alpha$$

$$\omega' \subseteq \Phi^\omega$$

From the premises of [REF-A]

$$T_1 :: \Phi'; \Gamma \vdash e : \tau'$$

$$\Phi' \leq \Phi$$

Clearly, for $E_1$ we have from the last

$$\alpha \subseteq \Phi^\alpha \subseteq \Phi'^\alpha$$

$$\omega' \subseteq \Phi^\omega \subseteq \Phi'^\omega$$

Similarly to the previous case, we get the wanted property for all sub-derivations by induction.

**case** [DEREF-A] **:**

$$\text{[DEREF-A]}\frac{\begin{array}{c}E_1 :: \langle T_1, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_r, \alpha', \omega' \cup \{L\}, H', r_L\rangle \\ r \in \mathrm{dom}(H')\end{array}}{E :: \langle T, \alpha, \omega, H, !\, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle}$$

$$T :: \Phi; \Gamma \vdash !\, e : \tau$$

$$\alpha \subseteq \Phi^\alpha$$

$$\omega' \subseteq \Phi^\omega$$

From the premises

$$\Phi_2^\varepsilon = \varepsilon$$

$$\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi'$$

$$\Phi' \leq \Phi$$

$$\mathit{ref}^{\,\{L\}}\ \tau \leq \mathit{ref}^{\,\varepsilon}\ \tau'$$

Therefore

$$\Phi_1^\alpha = \Phi'^\alpha$$

$$\Phi_1^\omega = \Phi'^\omega \cup \Phi_2^\varepsilon$$

For $E_1$ we have

$$\alpha \subseteq \Phi^\alpha \subseteq \Phi'^\alpha = \Phi_1^\alpha$$

$$\omega' \cup \{L\} \subseteq \Phi^\omega \cup \varepsilon \subseteq \Phi'^\omega \cup \Phi_2^\varepsilon = \Phi_1^\omega$$

As before, we get the wanted property for all sub-derivations by induction.

**case** Other **:**

The other cases are similar.

□

**Theorem A.18** (Contextual Effect Soundness)**.** *Given a program $e_p$ with no free variables, its typing $\mathcal{T}$ and its canonical evaluation $\mathcal{D}$, we can construct a typed evaluation $\mathcal{E}$ such that for every sub-derivation*

$$E :: \langle T, \alpha, \omega, H, e\rangle \longrightarrow_\varepsilon \langle T_v, \alpha', \omega', H', v\rangle$$

*in $\mathcal{E}$, where $T :: \Phi; \Gamma \vdash e : \tau$, it is always the case that $\alpha \subseteq \Phi^\alpha$ $\varepsilon \subseteq \Phi^\varepsilon$ and $\omega \subseteq \Phi^{\omega'}$.*

*Proof.* Sketch: Follows as a corollary from Lemmas A.17 and A.15, with initial $\Gamma_p = \emptyset$ and $H_p = \emptyset$. Since $\mathcal{D}$ is canonical, it is $\alpha_p = \omega'_p = \emptyset$ for the whole program (base case) and by induction we get the theorem for all sub-derivations. □

## B. Proteus-tx Proofs

**Lemma B.1** (Weakening)**.** *If* $\Phi; \Gamma \vdash e : \tau$ *and* $\Gamma' \supseteq \Gamma$ *then* $\Phi; \Gamma' \vdash e : \tau$.

*Proof.* By induction on the typing derivation of $\Phi; \Gamma \vdash e : \tau$. □

**Lemma B.2** (Flow subtyping)**.** *If* $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$ *then* $\Phi_1 \leq \Phi$ *and* $\Phi_2 \leq \Phi$.

*Proof.* Follows directly from the definitions. □

**Lemma B.3** (Subtyping reflexivity)**.** $\tau \leq \tau$ *for all* $\tau$.

*Proof.* Straightforward, from the definition of subtyping in Figure 2. □

**Lemma B.4** (Subtyping transitivity)**.** *For all* $\tau, \tau', \tau''$*, if* $\tau \leq \tau'$ *and* $\tau' \leq \tau''$ *then* $\tau \leq \tau''$.

*Proof.* By simultaneous induction on $\tau \leq \tau'$ and $\tau' \leq \tau''$. Notice that subtyping is syntax-directed, and this forces the final rule of each derivation to be the same:

**case** (SINT,SINT) **:**
From the definition of (SINT), we have $int \leq int$, hence $\tau \leq \tau''$ follows directly.

**case** (SREF,SREF) **:**
We have:

$$\text{SRef}\frac{\tau \leq \tau' \quad \tau' \leq \tau \quad \varepsilon \subseteq \varepsilon'}{ref^{\varepsilon}\ \tau \leq ref^{\varepsilon'}\ \tau'} \qquad \text{SRef}\frac{\tau' \leq \tau'' \quad \tau'' \leq \tau' \quad \varepsilon' \subseteq \varepsilon''}{ref^{\varepsilon'}\ \tau' \leq ref^{\varepsilon''}\ \tau''}$$

We know that $\varepsilon \subseteq \varepsilon' \wedge \varepsilon' \subseteq \varepsilon'' \Rightarrow \varepsilon \subseteq \varepsilon''$, and by induction we have that $\tau \leq \tau' \wedge \tau' \leq \tau'' \Rightarrow \tau \leq \tau''$ and $\tau' \leq \tau \wedge \tau'' \leq \tau' \Rightarrow \tau'' \leq \tau$, respectively.
We can now apply (SREF):

$$\text{SRef}\frac{\tau \leq \tau'' \quad \tau'' \leq \tau \quad \varepsilon \subseteq \varepsilon''}{ref^{\varepsilon}\ \tau \leq ref^{\varepsilon''}\ \tau''}$$

**case** (SFUN,SFUN) **:**
We have:

$$\text{SFun}\frac{\tau_1' \leq \tau_1 \quad \tau_2 \leq \tau_2' \quad \Phi \leq \Phi'}{\tau_1 \longrightarrow^{\Phi} \tau_2 \leq \tau_1' \longrightarrow^{\Phi'} \tau_2'} \qquad \text{SFun}\frac{\tau_1'' \leq \tau_1' \quad \tau_2' \leq \tau_2'' \quad \Phi' \leq \Phi''}{\tau_1' \longrightarrow^{\Phi'} \tau_2' \leq \tau_1'' \longrightarrow^{\Phi''} \tau_2''}$$

We know that $\Phi \leq \Phi' \wedge \Phi' \leq \Phi'' \Rightarrow \Phi \leq \Phi''$, and by induction (see (SREF,SREF) above) we have $\tau_1'' \leq \tau_1$ and $\tau_2 \leq \tau_2''$.
We can now apply (SFUN):

$$\text{SFun}\frac{\tau_1'' \leq \tau_1 \quad \tau_2 \leq \tau_2'' \quad \Phi \leq \Phi''}{\tau_1 \longrightarrow^{\Phi} \tau_2 \leq \tau_1'' \longrightarrow^{\Phi''} \tau_2''}$$

□

**Lemma B.5** (Value typing)**.** *If* $\Phi; \Gamma \vdash v : \tau$ *then* $\Phi'; \Gamma \vdash v : \tau$ *for all* $\Phi'$.

*Proof.* By induction on the typing derivation of $\Phi; \Gamma \vdash v : \tau$.

**case** (TINT) **:**
Thus $v \equiv n$ and we prove the result as follows:

$$\text{TSub}\frac{\text{TInt}\ \dfrac{}{\Phi_\emptyset; \Gamma \vdash n : int} \qquad int \leq int \qquad \text{SCtxt}\ \dfrac{\Phi_\emptyset \equiv \Phi_\emptyset \quad \Phi' \equiv [\alpha; \varepsilon'; \omega] \quad \emptyset \subseteq \varepsilon'}{\Phi_\emptyset \leq \Phi'}}{\Phi'; \Gamma \vdash n : int}$$

**case** (TGVAR) **:**
We have

$$\text{TGvar}\frac{\Gamma(\mathsf{f}) = \tau}{\Phi_\emptyset; \Gamma \vdash \mathsf{f} : \tau}$$

We prove the result as follows:

$$\text{TSub}\frac{\text{TGvar}\ \dfrac{\Gamma(\mathsf{f}) = \tau}{\Phi_\emptyset; \Gamma \vdash \mathsf{f} : \tau} \qquad \tau \leq \tau \qquad \text{SCtxt}\ \dfrac{\Phi_\emptyset \equiv \Phi_\emptyset \quad \Phi' \equiv [\alpha; \varepsilon'; \omega] \quad \emptyset \subseteq \varepsilon'}{\Phi_\emptyset \leq \Phi'}}{\Phi'; \Gamma \vdash \mathsf{f} : \tau}$$

**case** (TLOC) **:**
Similar to (TGVAR).

**case** (TSUB) **:**
We have

$$\text{TSub}\frac{\Phi'';\Gamma\vdash v:\tau' \quad \tau'\le\tau \qquad \text{SCtxt}\ \dfrac{\Phi''\equiv[\alpha;\varepsilon'';\omega] \quad \Phi\equiv[\alpha;\varepsilon;\omega] \quad \varepsilon''\subseteq\varepsilon}{\Phi''\le\Phi}}{\Phi;\Gamma\vdash v:\tau}$$

The result follows by induction on $\Phi'';\Gamma\vdash v:\tau'$ and by applying [TSUB].

□

**Lemma B.6** (Subtyping Derivations)**.** *If $\Phi;\Gamma\vdash e:\tau$ then we can construct a proof derivation of this judgment that ends in one use of* (TSUB) *whose premise uses a rule other than* (TSUB).

*Proof.* By induction on $\Phi;\Gamma\vdash e:\tau$.

**case** (TSUB) **:**
We have

$$\text{TSub}\frac{\Phi';\Gamma\vdash e:\tau' \quad \tau'\le\tau \qquad \text{SCtxt}\ \dfrac{\Phi'^{\varepsilon}\subseteq\Phi^{\varepsilon} \quad \Phi^{\alpha}\subseteq\Phi'^{\alpha} \quad \Phi^{\omega}\subseteq\Phi'^{\omega}}{\Phi'\le\Phi}}{\Phi;\Gamma\vdash e:\tau}$$

By induction, we have

$$\text{TSub}\frac{\Phi'';\Gamma\vdash e:\tau'' \quad \tau''\le\tau' \qquad \text{SCtxt}\ \dfrac{\Phi''^{\varepsilon}\subseteq\Phi'^{\varepsilon} \quad \Phi'^{\alpha}\subseteq\Phi''^{\alpha} \quad \Phi'^{\omega}\subseteq\Phi''^{\omega}}{\Phi''\le\Phi'}}{\Phi';\Gamma\vdash e:\tau'}$$

where the derivation $\Phi'';\Gamma\vdash e:\tau''$ does not conclude with (TSUB). By the transitivity of subtyping (Lemma B.4), we have $\tau''\le\tau$; we also have $\varepsilon''\subseteq\varepsilon$ and finally we get the desired result by (TSUB):

$$\text{TSub}\frac{\Phi'';\Gamma\vdash e:\tau'' \quad \tau''\le\tau \qquad \text{SCtxt}\ \dfrac{\Phi''^{\varepsilon}\subseteq\Phi^{\varepsilon} \quad \Phi^{\alpha}\subseteq\Phi''^{\alpha} \quad \Phi^{\omega}\subseteq\Phi''^{\omega}}{\Phi''\le\Phi}}{\Phi;\Gamma\vdash e:\tau}$$

**case** all others **:**
Since we have that the last rule in $\Phi;\Gamma\vdash e:\tau$ is not (TSUB), we have the desired result by applying (TSUB) (where $\tau\le\tau$ follows from the reflexivity of subtyping, Lemma B.3):

$$\text{TSub}\frac{\Phi;\Gamma\vdash e:\tau \quad \tau\le\tau \quad \Phi\le\Phi}{\Phi;\Gamma\vdash e:\tau}$$

□

**Lemma B.7** (Flow effect weakening)**.** *If $\Phi;\Gamma\vdash e:\tau$ where $\Phi\equiv[\alpha;\varepsilon;\omega]$, then $\Phi';\Gamma\vdash e:\tau$ where $\Phi'\equiv[\alpha';\varepsilon;\omega']$, $\alpha'\subseteq\alpha$, and $\omega'\subseteq\omega$, and all uses of [*TSUB*] applying $\Phi'\le\Phi$ require $\Phi'^{\omega}=\Phi^{\omega}$ and $\Phi'^{\alpha}=\Phi^{\alpha}$.*

*Proof.* By induction on $\Phi;\Gamma\vdash e:\tau$.

**case** (TGVAR),(TINT),(TVAR) **:**
Trivial.
**case** (TUPDATE) **:**
We have

$$\text{TUpdate}\frac{\alpha\subseteq\alpha'' \quad \omega\subseteq\omega''}{(\Phi_{\emptyset});\Gamma\vdash \mathsf{update}^{\alpha'',\omega''}:\mathit{int}}$$

Since $\alpha'\subseteq\alpha$ and $\omega'\subseteq\omega$ we can apply (TUPDATE):

$$\text{TUpdate}\frac{\alpha'\subseteq\alpha'' \quad \omega'\subseteq\omega''}{([\alpha';\varepsilon;\omega']);\Gamma\vdash \mathsf{update}^{\alpha',\omega'}:\mathit{int}}$$

**case** (TTRANSACT) **:**
We have

$$\text{TTransact}\frac{\Phi'';\Gamma\vdash e:\tau \quad \Phi^{\alpha}\subseteq\Phi''^{\alpha} \quad \Phi^{\omega}\subseteq\Phi''^{\omega}}{\Phi;\Gamma\vdash \mathsf{tx}\ e:\tau}$$

Let $\Phi'=[\alpha';\varepsilon;\omega']$. Since $\Phi'^{\alpha}\subseteq\Phi^{\alpha}$ and $\Phi'^{\omega}\subseteq\Phi^{\omega}$ we can apply (TTRANSACT):

$$\text{TTransact}\frac{\Phi'';\Gamma\vdash e:\tau \quad \Phi'^{\alpha}\subseteq\Phi''^{\alpha} \quad \Phi'^{\omega}\subseteq\Phi''^{\omega}}{\Phi';\Gamma\vdash \mathsf{tx}\ e:\tau}$$

**case** (TINTRANS) **:**
Similar to (TTRANSACT).

**case** (TSUB) **:**
We have

$$\text{TSub}\frac{\Phi';\Gamma\vdash e:\tau' \qquad \tau'\leq\tau \qquad \dfrac{\Phi'^{\varepsilon}\subseteq\Phi^{\varepsilon} \qquad \Phi^{\omega}\subseteq\Phi'^{\omega} \qquad \Phi^{\alpha}\subseteq\Phi'^{\alpha}}{\Phi'\leq\Phi}}{\Phi;\Gamma\vdash e:\tau}$$

Let $\Phi'' = [\Phi^{\alpha};\Phi'^{\varepsilon};\Phi^{\omega}]$. Thus we have

$$\text{TSub}\frac{\Phi'';\Gamma\vdash e:\tau' \qquad \tau'\leq\tau \qquad \dfrac{\Phi''^{\varepsilon}\subseteq\Phi^{\varepsilon} \qquad \Phi^{\omega}=\Phi''^{\omega} \qquad \Phi^{\alpha}=\Phi''^{\alpha}}{\Phi''\leq\Phi}}{\Phi;\Gamma\vdash e:\tau}$$

where the first premise follows by induction (which we can apply because $\Phi''^{\omega}\subseteq\Phi'^{\omega}$ and $\Phi''^{\alpha}\subseteq\Phi'^{\alpha}$ by assumption); the first premise of $\Phi''\leq\Phi$ is by assumption, and the latter two premises are by definition of $\Phi''$.

**case** (TREF) **:**
We know that

$$\text{TRef}\frac{\Phi;\Gamma\vdash e:\tau}{\Phi;\Gamma\vdash \mathsf{ref}\ \ e : \mathit{ref}^{\varepsilon}\ \tau}$$

and have $\Phi';\Gamma\vdash e:\tau$ by induction, hence we get the result by (TREF).

**case** (TDEREF) **:**
We know that

$$\text{TDeref}\frac{\Phi_1;\Gamma\vdash e:\mathit{ref}^{\varepsilon}\ \tau \qquad \Phi_2^{\varepsilon}=\varepsilon \qquad \Phi_1\rhd\Phi_2\hookrightarrow\Phi}{\Phi;\Gamma\vdash !\,e:\tau}$$

We have $\Phi'\equiv[\alpha';\Phi_1^{\varepsilon}\cup\Phi_2^{\varepsilon};\omega']$ where $\alpha'\subseteq\Phi^{\alpha}$ and $\omega'\subseteq\Phi^{\omega}$. Choose $\Phi_1'\equiv[\alpha';\Phi_1^{\varepsilon};\Phi_2^{\varepsilon}\cup\omega']$ and $\Phi_2'\equiv[\alpha'\cup\Phi_1^{\varepsilon};\Phi_2^{\varepsilon};\omega']$, hence $\Phi_1'\longrightarrow\Phi_2'$, $\Phi_2'^{\varepsilon}=\Phi_2^{\varepsilon}=\varepsilon$, and $\Phi'\equiv\Phi_1'\rhd\Phi_2'$. We want to prove that $\Phi';\Gamma\vdash !\,e:\tau$. Since $\alpha'\subseteq\alpha$ and $\Phi_2^{\varepsilon}\cup\omega'\subseteq\Phi_2^{\varepsilon}\cup\omega$ we can apply induction to get $\Phi_1';\Gamma\vdash e:\mathit{ref}^{\varepsilon}\ \tau$ and we get the result by applying (TDEREF):

$$\text{TDeref}\frac{\Phi_1';\Gamma\vdash e:\mathit{ref}^{\varepsilon}\ \tau \qquad \Phi_2'^{\varepsilon}=\varepsilon \qquad \Phi_1'\rhd\Phi_2'\hookrightarrow\Phi'}{\Phi';\Gamma\vdash !\,e:\tau}$$

**case** (TRET) **:**
Similar to [TDEREF].

**case** (TAPP) **:**
We know that

$$\text{TApp}\frac{\Phi_1;\Gamma\vdash e_1:\tau_1\longrightarrow^{\Phi_f}\tau_2 \qquad \Phi_2;\Gamma\vdash e_2:\tau_1 \qquad \Phi_1\rhd\Phi_2\rhd\Phi_3\hookrightarrow\Phi \qquad \Phi_3^{\varepsilon}=\Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha}\subseteq\Phi_f^{\alpha} \qquad \Phi_3^{\omega}\subseteq\Phi_f^{\omega}}{\Phi;\Gamma\vdash e_1\ e_2:\tau_2}$$

We have $\Phi'\equiv[\alpha';\Phi_1^{\varepsilon}\cup\Phi_2^{\varepsilon}\cup\Phi_3^{\varepsilon};\omega']$ where $\alpha'\subseteq\Phi^{\alpha}$ and $\omega'\subseteq\Phi^{\omega}$. Choose $\Phi_1'\equiv[\alpha';\Phi_1^{\varepsilon};\Phi_2^{\varepsilon}\cup\Phi_3^{\varepsilon}\cup\omega']$, $\Phi_2'\equiv[\alpha'\cup\Phi_1^{\varepsilon};\Phi_2^{\varepsilon};\Phi_3^{\varepsilon}\cup\omega']$, $\Phi_3'\equiv[\alpha'\cup\Phi_1^{\varepsilon}\cup\Phi_2^{\varepsilon};\Phi_3^{\varepsilon};\omega']$, hence $\Phi_3'^{\varepsilon}=\Phi_3^{\varepsilon}=\varepsilon_f$, and $\Phi_1'\rhd\Phi_2'\rhd\Phi_3'\hookrightarrow\Phi'$. We want to prove that $\Phi';\Gamma\vdash e_1\ e_2:\tau_2$. Since $\alpha'\subseteq\alpha$ and $\Phi_2^{\varepsilon}\cup\Phi_3^{\varepsilon}\cup\omega'\subseteq\Phi_2^{\varepsilon}\cup\Phi_3^{\varepsilon}\cup\omega'$ we can apply induction to get $\Phi_1';\Gamma\vdash e_1:\tau_1\longrightarrow^{\Phi_f}\tau_2$. Similarly, since $\alpha'\cup\Phi_1^{\varepsilon}\subseteq\alpha\cup\Phi_1^{\varepsilon}$ and $\Phi_3^{\varepsilon}\cup\omega'\subseteq\Phi_3^{\varepsilon}\cup\omega$, we can apply induction to get $\Phi_2';\Gamma\vdash e_2:\tau_1$. We get the get the result by applying (TAPP):

$$\text{TApp}\frac{\Phi_1';\Gamma\vdash e_1:\tau_1\longrightarrow^{\Phi_f'}\tau_2 \qquad \Phi_2';\Gamma\vdash e_2:\tau_1 \qquad \Phi_1'\rhd\Phi_2'\rhd\Phi_3'\hookrightarrow\Phi' \qquad \Phi_3'^{\varepsilon}=\Phi_f^{\varepsilon} \qquad \Phi_3'^{\alpha}\subseteq\Phi_f^{\alpha} \qquad \Phi_3'^{\omega}\subseteq\Phi_f^{\omega}}{\Phi';\Gamma\vdash e_1\ e_2:\tau_2}$$

**case** (TASSIGN), (TIF), (TLET) **:**
Similar to (TAPP).

□

**Definition B.8.** *If* $\Phi;\Gamma\vdash e:\tau$, $[\![\Phi;\Gamma\vdash e:\tau]\!]=\mathcal{R}$, *and* $\Phi;\Gamma\vdash e:\tau\rightsquigarrow\mathcal{R}'$ *then* $\mathcal{R}\equiv\mathcal{R}'$, *where* $[\![\Phi;\Gamma\vdash e:\tau]\!]$ *is defined in Figure 19.*

**Lemma B.9** (Left subexpression version consistency)**.** *If* $\Phi,\mathcal{R};H\vdash\Sigma$ *and* $\Phi_1\rhd\Phi_2\hookrightarrow\Phi$ *then* $\Phi_1,\mathcal{R};H\vdash\Sigma$.

*Proof.* We know that $\Phi_1\rhd\Phi_2\equiv[\alpha_1;\varepsilon_1\cup\varepsilon_2;\omega_2]$. We have two cases:

$\mathcal{R}\equiv\cdot$**:** Thus $\Sigma\equiv(n',\sigma)$ and by assumption we have:

$$\text{TC1}\frac{\mathsf{f}\in\sigma\Rightarrow\mathsf{f}\in\alpha_1 \qquad \mathsf{f}\in\varepsilon_1\cup\varepsilon_2\Rightarrow n'\in ver(H,\mathsf{f})}{[\alpha_1;\varepsilon_1\cup\varepsilon_2;\omega_2],\cdot;H\vdash(n',\sigma)}$$

$$\left[\!\left[(\textsc{TInt})\frac{}{\Phi_\emptyset;\Gamma\vdash n:\mathit{int}}\right]\!\right]=\cdot \qquad \left[\!\left[(\textsc{TVar})\frac{\Gamma(x)=\tau}{\Phi_\emptyset;\Gamma\vdash x:\tau}\right]\!\right]=\cdot \qquad \left[\!\left[(\textsc{TGVar})\frac{\Gamma(\mathsf{f})=\tau}{\Phi_\emptyset;\Gamma\vdash \mathsf{f}:\tau}\right]\!\right]=\cdot$$

$$\left[\!\left[(\textsc{TSub})\frac{\begin{array}{c}D::\Phi';\Gamma\vdash e:\tau'\\ \tau'\le\tau \qquad \Phi'\le\Phi\end{array}}{\Phi;\Gamma\vdash e:\tau}\right]\!\right]=\mathcal{R},\ \text{where}\ [\![D]\!]=\mathcal{R}$$

$$\left[\!\left[(\textsc{TUpdate})\frac{\Phi^\alpha\subseteq\alpha' \qquad \Phi^\omega\subseteq\omega'}{\Phi;\Gamma\vdash \mathsf{update}^{\alpha',\omega'}:\mathit{int}}\right]\!\right]=\cdot$$

$$\left[\!\left[(\textsc{TRef})\frac{D::\Phi;\Gamma\vdash e:\tau}{\Phi;\Gamma\vdash \mathsf{ref}\ \ e:\mathit{ref}^{\,\varepsilon}\ \tau}\right]\!\right]=\mathcal{R},\ \text{where}\ [\![D]\!]=\mathcal{R}$$

$$\left[\!\left[(\textsc{TDeref})\frac{\begin{array}{c}D_1::\Phi_1;\Gamma\vdash e:\mathit{ref}^{\,\varepsilon}\ \tau\\ \Phi_2^\varepsilon=\varepsilon \qquad \Phi_1\rhd\Phi_2\hookrightarrow\Phi\end{array}}{\Phi;\Gamma\vdash !\,e:\tau}\right]\!\right]=\mathcal{R}_1,\ \text{where}\ [\![D_1]\!]=\mathcal{R}_1$$

$$\left[\!\left[(\textsc{TAssign})\frac{\begin{array}{c}D_1::\Phi_1;\Gamma\vdash e_1:\mathit{ref}^{\,\varepsilon}\ \tau \qquad D_2::\Phi_2;\Gamma\vdash e_2:\tau\\ \Phi_3^\varepsilon=\varepsilon \qquad\qquad \Phi_1\rhd\Phi_2\rhd\Phi_3\hookrightarrow\Phi\end{array}}{\Phi;\Gamma\vdash e_1:=e_2:\tau}\right]\!\right]=\mathcal{R}_1\bowtie\mathcal{R}_2,\ \text{where}\ \begin{array}{l}[\![D_1]\!]=\mathcal{R}_1\\ [\![D_2]\!]=\mathcal{R}_2\\ e_1\not\equiv v\Rightarrow\mathcal{R}_2=\cdot\end{array}$$

$$\left[\!\left[(\textsc{TIf})\frac{\begin{array}{c}D_1::\Phi_1;\Gamma\vdash e_1:\mathit{int}\\ D_2::\Phi_2;\Gamma\vdash e_2:\tau \qquad D_3::\Phi_2;\Gamma\vdash e_3:\tau\\ \Phi_1\rhd\Phi_2\hookrightarrow\Phi\end{array}}{\Phi;\Gamma\vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3:\tau}\right]\!\right]=\mathcal{R}_1,\ \text{where}\ \begin{array}{l}[\![D_1]\!]=\mathcal{R}_1\\ [\![D_2]\!]=\cdot\\ [\![D_3]\!]=\cdot\end{array}$$

$$\left[\!\left[(\textsc{TLet})\frac{\begin{array}{c}D_1::\Phi_1;\Gamma\vdash e_1:\tau_1 \qquad D_2::\Phi_2;\Gamma,x:\tau_1\vdash e_2:\tau_2\\ \Phi_1\rhd\Phi_2\hookrightarrow\Phi\end{array}}{\Phi;\Gamma\vdash \mathsf{let}\ x:\tau_1=e_1\ \mathsf{in}\ e_2:\tau_2}\right]\!\right]=\mathcal{R}_1,\ \text{where}\ \begin{array}{l}[\![D_1]\!]=\mathcal{R}_1\\ [\![D_2]\!]=\cdot\end{array}$$

$$\left[\!\left[(\textsc{TApp})\frac{\begin{array}{c}D_1::\Phi_1;\Gamma\vdash e_1:\tau_1\longrightarrow^{\Phi_f}\tau_2 \qquad D_2::\Phi_2;\Gamma\vdash e_2:\tau_1\\ \Phi_1\rhd\Phi_2\rhd\Phi_3\hookrightarrow\Phi\\ \Phi_3^\varepsilon=\Phi_f^\varepsilon \qquad \Phi_3^\alpha\subseteq\Phi_f^\alpha \qquad \Phi_3^\omega\subseteq\Phi_f^\omega\end{array}}{\Phi;\Gamma\vdash e_1\ e_2:\tau_2}\right]\!\right]=\mathcal{R}_1\bowtie\mathcal{R}_2,\ \text{where}\ \begin{array}{l}[\![D_1]\!]=\mathcal{R}_1\\ [\![D_2]\!]=\mathcal{R}_2\\ e_1\not\equiv v\Rightarrow\mathcal{R}_2=\cdot\end{array}$$

$$\left[\!\left[(\textsc{TTransact})\frac{\Phi_1;\Gamma\vdash e:\tau \qquad \Phi^\alpha\subseteq\Phi_1^\alpha \qquad \Phi^\omega\subseteq\Phi_1^\omega}{\Phi;\Gamma\vdash \mathsf{tx}\ e:\tau}\right]\!\right]=\cdot$$

$$\left[\!\left[(\textsc{TIntrans})\frac{D_1::\Phi_1;\Gamma\vdash e:\tau \qquad \Phi^\alpha\subseteq\Phi_1^\alpha \qquad \Phi^\omega\subseteq\Phi_1^\omega}{\Phi;\Gamma\vdash \mathsf{intx}\ e:\tau}\right]\!\right]=\Phi,\mathcal{R}_1\ \text{where}\ [\![D_1]\!]=\mathcal{R}_1$$

$$\cdot\bowtie\mathcal{R}=\mathcal{R}\ \wedge\ \mathcal{R}\bowtie\cdot=\mathcal{R}$$

**Figure 19.** Transaction effect extraction

The result follows from [TC1]:

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f}\in\sigma\Rightarrow\mathsf{f}\in\alpha_1\\ \mathsf{f}\in\varepsilon_1\Rightarrow n'\in\mathit{ver}(H,\mathsf{f})\end{array}}{[\alpha_1;\varepsilon_1;\omega_1],\cdot;H\vdash(n',\sigma)}$$

$\mathcal{R} \equiv \Phi', \mathcal{R}'$**:** Thus we must have

$$\text{TC2}\frac{\begin{array}{c}\Phi', \mathcal{R}'; H \vdash \Sigma' \\ \Phi \equiv [\alpha_1; \varepsilon_1 \cup \varepsilon_2; \omega_2] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \cup \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi, \Phi', \mathcal{R}'; H \vdash ((n', \sigma), \Sigma')}$$

where $\Sigma \equiv ((n', \sigma), \Sigma')$. The result follows by [TC2]:

$$\text{TC2}\frac{\begin{array}{c}\Phi', \mathcal{R}'; H \vdash \Sigma' \\ \Phi_1 \equiv [\alpha_1; \varepsilon_1; \omega_1] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi_1, \Phi', \mathcal{R}'; H \vdash ((n', \sigma), \Sigma')}$$

where the first premise follows by assumption.

□

**Lemma B.10** (Subexpression version consistency)**.** *If* $\Phi, \mathcal{R}_1 \bowtie \mathcal{R}_2; H \vdash \Sigma$ *and* $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$ *then*

*(i)* $\mathcal{R}_2 \equiv \cdot$ *implies* $\Phi_1, \mathcal{R}_1; H \vdash \Sigma$
*(ii)* $\mathcal{R}_1 \equiv \cdot$ *and* $\Phi_1^\varepsilon \equiv \emptyset$ *implies* $\Phi_2, \mathcal{R}_2; H \vdash \Sigma$

*Proof.* (i) Since $\mathcal{R}_2 = \cdot$ by assumption, we have $\mathcal{R}_1 = \mathcal{R}_1 \bowtie \mathcal{R}_2$. We have two cases:

$\mathcal{R}_1 \equiv \cdot$**:** Thus we must have

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \cup \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{[\alpha_1; \varepsilon_1 \cup \varepsilon_2; \omega_2], \cdot; H \vdash (n', \sigma)}$$

where $\Sigma \equiv (n', \sigma)$. The result follows from [TC1] :

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{[\alpha_1; \varepsilon_1; \omega_1], \cdot; H \vdash (n', \sigma)}$$

$\mathcal{R}_1 \equiv \Phi', \mathcal{R}'$**:** Thus we must have

$$\text{TC2}\frac{\begin{array}{c}\Phi', \mathcal{R}'; H \vdash \Sigma' \\ \Phi \equiv [\alpha_1; \varepsilon_1 \cup \varepsilon_2; \omega_2] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \cup \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi, \Phi', \mathcal{R}'; H \vdash ((n', \sigma), \Sigma')}$$

where $\Sigma \equiv ((n', \sigma), \Sigma')$. The result follows by [TC2]:

$$\text{TC2}\frac{\begin{array}{c}\Phi', \mathcal{R}'; H \vdash \Sigma' \\ \Phi_1 \equiv [\alpha_1; \varepsilon_1; \omega_1] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi_1, \Phi', \mathcal{R}'; H \vdash ((n', \sigma), \Sigma')}$$

where the first premise follows by assumption.

(ii) Since $\mathcal{R}_1 = \cdot$ by assumption, we have $\mathcal{R}_2 = \mathcal{R}_1 \bowtie \mathcal{R}_2$. We have two cases:

$\mathcal{R}_2 \equiv \cdot$**:** Thus we must have

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \cup \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{[\alpha_1; \varepsilon_1 \cup \varepsilon_2; \omega_2], \cdot; H \vdash (n', \sigma)}$$

where $\Sigma \equiv (n', \sigma)$. Since $\Phi_1^\varepsilon \equiv \emptyset$ and $\Phi_2^\alpha = \Phi_1^\alpha \cup \Phi_1^\varepsilon$ we have $\Phi_2^\alpha = \Phi_1^\alpha$ and the result follows from [TC1]:

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_2 \\ \mathsf{f} \in \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{[\alpha_2; \varepsilon_2; \omega_2], \cdot; H \vdash (n', \sigma)}$$

$\mathcal{R}_2 \equiv \Phi', \mathcal{R}'$**:** Thus we must have

$$\text{TC2}\frac{\begin{array}{c}\Phi', \mathcal{R}'; H \vdash \Sigma' \\ \Phi \equiv [\alpha_1; \varepsilon_1 \cup \varepsilon_2; \omega_2] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_1 \\ \mathsf{f} \in \varepsilon_1 \cup \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi, \Phi', \mathcal{R}'; H \vdash ((n', \sigma), \Sigma')}$$

where $\Sigma \equiv ((n', \sigma), \Sigma')$. The result follows by [TC2] and $\Phi_2^\alpha = \Phi_1^\alpha$ (because $\Phi_1^\varepsilon \equiv \emptyset$ and $\Phi_2^\alpha = \Phi_1^\alpha \cup \Phi_1^\varepsilon$):

$$\text{TC2}\frac{\begin{array}{c}\Phi', \mathcal{R}'; H \vdash \Sigma' \\ \Phi_2 \equiv [\alpha_2; \varepsilon_2; \omega_2] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha_2 \\ \mathsf{f} \in \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi_2, \Phi', \mathcal{R}'; H \vdash ((n', \sigma), \Sigma')}$$

where the first premise follows by assumption.

□

**Lemma B.11** (Stack Shapes)**.** *If* $\langle n; \Sigma; H; e\rangle \longrightarrow_{\varepsilon_0} \langle n; \Sigma'; H'; e'\rangle$ *then* $top(\Sigma) = (n', \sigma)$ *and* $top(\Sigma') = (n'', \sigma')$ *where* $n' = n''$ *and* $\sigma \subseteq \sigma'$.

*Proof.* By induction on $\langle n; \Sigma; H; e\rangle \longrightarrow_{\varepsilon_0} \langle n; \Sigma'; H'; e'\rangle$. □

**Lemma B.12** (Update preserves heap safety)**.** *If* $n; \Gamma \vdash H$ *and* $updateOK(upd, H, (\alpha, \omega), dir)$ *then* $n+1; \mathcal{U}[\Gamma]^{upd} \vdash \mathcal{U}[H]^{upd}_{n+1}$.

*Proof.* Let $n' = n+1$ and $\Gamma' \equiv \mathcal{U}[\Gamma]^{upd}$ and $H' \equiv \mathcal{U}[H]^{upd}_{n'}$. From the definition of heap typing (Figure 9), to prove $n'; \Gamma' \vdash H'$, we need to show:

1. $\mathrm{dom}(\Gamma') = \mathrm{dom}(H')$
2. $\forall \mathsf{z} \mapsto (\tau, v, \nu) \in H'. \quad \Phi_\emptyset; \Gamma' \vdash v : \tau \ \wedge\ \Gamma'(\mathsf{z}) = \mathit{ref}^{\,\varepsilon}\ \tau \ \wedge\ \mathsf{z} \in \varepsilon$
3. $\forall \mathsf{z} \mapsto (\tau \longrightarrow^{\Phi} \tau', \lambda(x).e, \nu) \in H'. \quad \Phi; \Gamma', x : \tau \vdash e : \tau' \ \wedge\ \Gamma'(\mathsf{z}) = \tau \longrightarrow^{\Phi} \tau' \ \wedge\ \mathsf{z} \in \Phi^\alpha \ \wedge\ \mathsf{z} \in \Phi^\varepsilon$
4. $\forall r \mapsto (\cdot, v, \nu) \in H'. \quad \Phi_\emptyset; \Gamma' \vdash v : \tau \ \wedge\ \Gamma'(r) = \mathit{ref}^{\,\varepsilon}\ \tau$
5. $\forall \mathsf{z} \mapsto (\tau, b, \nu) \in H'. \quad n' \in \nu$

Proof by induction on $H$.

**case** $H \equiv \emptyset$ **:**

We have $\mathcal{U}[\emptyset]^{upd}_{n'} = upd^{add}$ (modified to have version set $\{n+1\}$), and thus $\mathrm{dom}(H') = \mathrm{dom}(upd^{add})$. Our assumption $\mathrm{dom}(H) = \mathrm{dom}(\Gamma)$ implies that $\Gamma = \emptyset$, and thus $\Gamma' = \mathcal{U}[\emptyset]^{upd} = \mathit{types}(upd^{add})$.

1. $\mathrm{dom}(\Gamma') = \mathrm{dom}(\mathit{types}(upd^{add})) = \mathrm{dom}(upd^{add}) = \mathrm{dom}(H')$.
2. Since $H' = upd^{add}$, this follows directly from $updateOK(upd, H, (\alpha, \omega), dir)$ given the definition of $\Gamma' = \mathcal{U}[\emptyset]^{upd} = \mathit{types}(upd^{add})$.
3. Similar to 2.
4. Vacuously true, since $r \notin \mathrm{dom}(H') = \mathrm{dom}(upd^{add})$ for all $r$.
5. Holds by the definition of $\mathcal{U}[\emptyset]^{upd}_{n'}$.

**case** $H \equiv (r \mapsto (\cdot, b, \emptyset), H'')$ **:**

We have $H' \equiv \mathcal{U}[(r \mapsto (\cdot, b, \emptyset), H'')]^{upd}_{n'} = (r \mapsto (\cdot, b, \emptyset)), \mathcal{U}[H'']^{upd}_{n'}$. Our assumption $\mathrm{dom}(H) = \mathrm{dom}(\Gamma)$ implies $\Gamma \equiv (r : \tau, \Gamma'')$ for some $\Gamma''$, where $\mathrm{dom}(H'') = \mathrm{dom}(\Gamma'')$ and $\Gamma' \equiv \mathcal{U}[r : \tau, \Gamma'']^{upd} = r : \tau, \mathcal{U}[\Gamma'']^{upd}$.

1. By induction we know $\mathrm{dom}(\mathcal{U}[\Gamma'']^{upd}) = \mathrm{dom}(\mathcal{U}[H'']^{upd}_{n'})$. But $\mathrm{dom}(H') = \mathrm{dom}(r = (\cdot, b, \emptyset)), \mathcal{U}[H'']^{upd}_{n'}) = \{r\} \cup \mathrm{dom}(\mathcal{U}[H'']^{upd}_{n'})$, and $\mathrm{dom}(\Gamma') = \mathrm{dom}(r : \tau, \mathcal{U}[\Gamma'']^{upd}) = \{r\} \cup \mathrm{dom}(\mathcal{U}[\Gamma'']^{upd})$.
2. Follows by induction, since $r \neq \mathsf{z}$ for all $\mathsf{z}$.
3. Same as above.
4. For $r$, this follows by assumption, since it is clear that $H(r) = \mathcal{U}[H]^{upd}_{n'}(r)$ and $\Gamma(r) = \mathcal{U}[\Gamma]^{upd}(r)$, and for the rest of the heap the property follows by induction.
5. Follows by induction, since $r \neq \mathsf{z}$ for all $\mathsf{z}$.

**case** $H \equiv (\mathsf{z} = (\tau, b, \nu), H'')$ **:**

We have $H' \equiv \mathcal{U}[(\mathsf{z} \mapsto (\tau, b, \nu), H'')]^{upd}_{n'} = (\mathsf{z} \mapsto (\tau, b', \nu')), \mathcal{U}[H'']^{upd}_{n'}$. Our assumption $\mathrm{dom}(H) = \mathrm{dom}(\Gamma)$ implies $\Gamma \equiv (\mathsf{z} : \mathit{heapType}(\mathsf{z}, \tau), \Gamma'')$ for some $\Gamma''$, where $\mathrm{dom}(H'') = \mathrm{dom}(\Gamma'')$ and $\Gamma' \equiv \mathcal{U}[\mathsf{z} : \tau, \Gamma'']^{upd} = \mathsf{z} : \mathit{heapType}(\mathsf{z}, \tau), \mathcal{U}[\Gamma'']^{upd}$.

1. Similar to the argument for the $H \equiv (r \mapsto (...), H'')$ case.
4. This follows by induction, since $\mathsf{z} \neq r$.

Now consider the remaining cases according to $\mathsf{z}$ with respect to $upd^{chg}$:

**case** $\mathsf{z} \notin \mathrm{dom}(upd^{chg})$ **:**

2. For $\mathsf{z}$, this follows by assumption, since it is clear that $H(\mathsf{z}) = \mathcal{U}[H]^{upd}_{n'}(\mathsf{z})$ and $\Gamma(\mathsf{z}) = \mathcal{U}[\Gamma]^{upd}(\mathsf{z})$. The rest of the heap follows by induction.
3. Same as above.
5. We have $\mathcal{U}[(\mathsf{z} \mapsto (\tau, b, \nu), H'')]^{upd}_{n'} = (\mathsf{z} \mapsto (\tau, b, \nu \cup \{n'\}), \mathcal{U}[H'']^{upd}_{n'})$ where $n' \in (\nu \cup \{n'\})$ for $\mathsf{z}$, and the rest follows by induction.

**case** $\mathsf{z} \in \mathrm{dom}(upd^{chg})$ **:**

2. From the definition of $updateOK(upd, H, (\alpha, \omega), dir)$ we know that (i) $\Phi_\emptyset; \mathcal{U}[\Gamma]^{upd} \vdash v' : \tau$. Considering $\mathsf{z}$, from the definition of $heapType(\tau, \mathsf{z})$ we have (ii) $heapType(\tau, \mathsf{z}) = ref^{\varepsilon}\ \tau$ where $\mathsf{z} \in \varepsilon$. Combining (i) and (ii) yields

$$\Phi_\emptyset; \Gamma' \vdash v : \tau \ \wedge\ \Gamma'(\mathsf{z}) = ref^{\varepsilon}\ \tau \ \wedge \mathsf{z} \in \varepsilon$$

   The property holds for the rest of the heap by induction.
3. Similar to the previous.
5. We have $\mathcal{U}[(\mathsf{z} \mapsto (\tau, b, \nu), H'')]_{n'}^{upd} = (\mathsf{z} \mapsto (\tau, b', \{n'\}), \mathcal{U}[H'']_{n'}^{upd})$ and obviously $n' \in \{n'\}$ for $\mathsf{z}$, and the rest by induction.

□

The following lemma states that if we start with a well-typed program and a version-consistent trace and we take an update step, then afterward we will still have a well-typed program whose trace is version-consistent.

**Lemma B.13** (Update preservation)**.**
*Suppose we have the following:*

1. $n \vdash H, e : \tau$ *(such that* $\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}$ *and* $n; \Gamma \vdash H$ *for some* $\Gamma, \Phi$*)*
2. $\Phi, \mathcal{R}; H \vdash \Sigma$
3. $traceOK(\Sigma)$
4. $\langle n; \Sigma; H; e\rangle \longrightarrow_{\mu} \langle n+1; \Sigma'; H';\ e\rangle$

*where* $H' \equiv \mathcal{U}[H]_{n+1}^{upd}$, $\Gamma' \equiv \mathcal{U}[\Gamma]^{upd}$, $\mu = (upd, dir)$, $\Sigma' \equiv \mathcal{U}[\Sigma]_{n}^{upd, dir}$, *and* $top(\Sigma') = (n'', \sigma')$. *Then for some* $\Phi'$ *such that* $\Phi'^{\alpha} = \Phi^{\alpha}$, $\Phi'^{\omega} = \Phi^{\omega}$, *and* $\Phi'^{\varepsilon} \subseteq \Phi^{\varepsilon}$ *and some* $\Gamma' \supseteq \Gamma$ *we have that:*

1. $n+1 \vdash H', e : \tau$ *where* $\Phi'; \Gamma' \vdash e : \tau \rightsquigarrow \mathcal{R}$ *and* $n+1; \Gamma' \vdash H'$
2. $\Phi', \mathcal{R}; H' \vdash \Sigma'$
3. $traceOK(\Sigma')$
4. $(dir = bck) \Rightarrow n'' \equiv n+1 \ \wedge\ (dir = fwd) \Rightarrow (\mathsf{f} \in \omega \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f}))$

*Proof.* Since $\mathcal{U}[\Gamma]^{upd} \supseteq \Gamma$, $\Phi; \mathcal{U}[\Gamma]^{upd} \vdash e : \tau \rightsquigarrow \mathcal{R}$ follows by weakening (Lemma B.1). Proceed by simultaneous induction on the typing derivation of $e$ ($n \vdash H, e : \tau$) and on the evaluation derivation $\langle n; \Sigma; H; e\rangle \longrightarrow_{\mu} \langle n+1; \Sigma'; H';\ e\rangle$. Consider the last rule used in the evaluation derivation:

**case** [GVAR-DEREF], [GVAR-ASSIGN], [CALL], [LET], [TX-START], [TX-END], [REF], [DEREF], [ASSIGN], [IF-T], [IF-F], [NO-UPDATE] **:**
Not possible, as these transitions cannot cause an update to occur.

**case** [UPDATE] **:**
This implies that $e \equiv \mathsf{update}^{\alpha,\omega}$ and thus

$$\langle n; (n', \sigma); H; \mathsf{update}^{\alpha,\omega}\rangle \longrightarrow_{\mu} \langle n+1; \mathcal{U}\big[(n', \sigma)\big]_{n+1}^{upd, dir}; \mathcal{U}[H]_{n+1}^{upd}; 1\rangle$$

where $\mu \equiv (upd, dir)$ and $updateOK(upd, H, (\alpha, \omega), dir)$. By subtyping derivations (Lemma B.6) we have

$$\text{TSub}\dfrac{\text{TUpdate}\dfrac{\alpha \subseteq \alpha'' \quad \omega \subseteq \omega'' \quad \Phi_u \equiv [\alpha; \emptyset; \omega]}{\Phi_u; \Gamma \vdash \mathsf{update}^{\alpha'', \omega''} : int \rightsquigarrow \cdot} \quad int \leq int \quad \Phi_u \leq \Phi \quad \Phi \equiv [\alpha; \varepsilon; \omega]}{\Phi; \Gamma \vdash \mathsf{update}^{\alpha,\omega} : int \rightsquigarrow \cdot}$$

and by flow effect weakening (Lemma B.7) we know that $\alpha$ and $\omega$ are unchanged in the use of (TSUB). Let $\Phi' = \Phi_u$ (hence $\Phi'^{\alpha} = \Phi^{\alpha}$, $\Phi'^{\omega} = \Phi^{\omega}$, and $\emptyset \subseteq \Phi^{\varepsilon}$ as required) and $(n'', \sigma') \equiv \mathcal{U}[(n', \sigma)]_{n+1}^{upd, dir}$.
To prove 1., we get $n+1; \Gamma' \vdash H'$ by Lemma B.12 and $\Phi_u; \Gamma' \vdash 1 : int \rightsquigarrow \cdot$ by [TINT].
To prove 2., we must show $\Phi_u, \cdot; H' \vdash (n'', \sigma')$. By assumption, we have

$$\text{TC1}\dfrac{\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \quad \mathsf{f} \in \varepsilon \Rightarrow n' \in ver(H, \mathsf{f})}{[\alpha; \varepsilon; \omega], \cdot; H \vdash (n', \sigma)}$$

We need to prove

$$\text{TC1}\dfrac{\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \quad \mathsf{f} \in \emptyset \Rightarrow n'' \in ver(H', \mathsf{f})}{[\alpha; \emptyset; \omega], \cdot; H' \vdash (n'', \sigma')}$$

We have the first premise by assumption (since $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$ from the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd, dir}$). The second premise holds vacuously.
To prove 3., we must show $traceOK(n'', \sigma')$. Consider each possible update type:
**case** $dir = bck$ **:**
From the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd, bck}$, we know that $n'' = n+1$. Consider $(\mathsf{f}, \nu) \in \sigma$; it must be the case that $\mathsf{f} \notin \mathrm{dom}(upd^{chg})$. This is because $dir = bck$ implies $\alpha \cap \mathrm{dom}(upd^{chg}) = \emptyset$ and by assumption (from the first premise of [TC1] above) $\mathsf{f} \in \alpha$. Therefore, since $\mathsf{f} \notin \mathrm{dom}(upd^{chg})$, its $\sigma'$ entry is $(\mathsf{f}, \nu \cup \{n''\})$, which is the required result.

**case** $dir = fwd$ **:**

Since $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,fwd} = (n', \sigma)$, the result is true by assumption.

To prove 4., we must show $n'' \equiv n + 1 \vee (\mathsf{f} \in \omega \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f}))$. Consider each possible update type:

**case** $dir = bck$ **:**

From the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,bck}$, we know that $n'' = n + 1$ so we are done.

**case** $dir = fwd$ **:**

We have $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,fwd} = (n', \sigma)$, and from $updateOK(upd, H, (\alpha, \omega), dir)$ we know that $\mathsf{f} \in \omega \Rightarrow \mathsf{f} \notin \mathrm{dom}(upd^{chg})$. From the definition of $\mathcal{U}[H]_n^{upd}$ we know that $\mathcal{U}[(\mathsf{f} \mapsto (\tau, b, \nu), H)]_{n+1}^{upd} = \mathsf{f} \mapsto (\tau, b, \nu \cup \{n+1\})$ if $\mathsf{f} \notin \mathrm{dom}(upd^{chg})$. This implies that for $\mathsf{f} \in \omega$, $ver(H, \mathsf{f}) = \nu$ and $ver(H', \mathsf{f}) = \nu \cup \{n+1\}$, and therefore $ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f})$.

**case** [TX-CONG-1] **:**

We have that $\langle n; ((n', \sigma), \Sigma); H; \mathsf{intx}\ \ e\rangle \longrightarrow_\mu \langle n+1; (\mathcal{U}[(n', \sigma)]_{n+1}^{upd,dir}, \Sigma'); H'; \mathsf{intx}\ \ e'\rangle$ follows from $\langle n; \Sigma; H; e\rangle \longrightarrow_\mu \langle n+1; \Sigma'; H'; e'\rangle$ by [TX-CONG-1], where $\mu \equiv (upd, dir)$. Let $(n'', \sigma') \equiv \mathcal{U}[(n', \sigma)]_{n+1}^{upd,dir}$. By assumption and subtyping derivations (Lemma B.6) we have

$$\textsc{TSub}\ \dfrac{\textsc{TIntrans}\ \dfrac{\Phi_e; \Gamma \vdash e : \tau' \rightsquigarrow \mathcal{R} \quad \alpha \subseteq \Phi_e^\alpha \quad \omega \subseteq \Phi_e^\omega}{[\alpha; \emptyset; \omega]; \Gamma \vdash \mathsf{intx}\ e : \tau' \rightsquigarrow \Phi_e, \mathcal{R}} \qquad \tau' \leq \tau \qquad [\alpha; \emptyset; \omega] \leq [\alpha; \varepsilon; \omega]}{[\alpha; \varepsilon; \omega]; \Gamma \vdash \mathsf{intx}\ e : \tau \rightsquigarrow \Phi_e, \mathcal{R}}$$

and by flow effect weakening (Lemma B.7) we know that $\alpha$ and $\omega$ are unchanged in the use of (TSUB). We have $\Phi_e \equiv [\alpha_e; \varepsilon_e; \omega_e]$, so that $\omega_e \supseteq \omega$ and $\alpha_e \supseteq \alpha$. To apply induction, we must show that $\Phi_e, \mathcal{R}; H \vdash \Sigma$ (which follows by inversion on $\Phi, \Phi_e, \mathcal{R}; H \vdash ((n', \sigma), \Sigma))$; $\Phi_e; \Gamma \vdash e : \tau' \rightsquigarrow \mathcal{R}$ (which follows by assumption); and $n; \Gamma \vdash H$ (by assumption).

By induction we have:

(i) $\Phi'_e; \Gamma' \vdash e' : \tau' \rightsquigarrow \mathcal{R}$ and

(ii) $n+1; \Gamma' \vdash H'$

(iii) $\Phi'_e, \mathcal{R}; H' \vdash \Sigma'$

(iv) $traceOK(\Sigma')$

(v) $(dir = bck) \Rightarrow n'' \equiv n + 1 \ \wedge\ (dir = fwd) \Rightarrow (\mathsf{f} \in \omega_e \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f}))$

where $\Phi'_e \equiv [\alpha_e; \varepsilon'_e; \omega_e]$, $\varepsilon'_e \subseteq \varepsilon_e$.

Let $\Phi' = [\alpha; \emptyset; \omega]$ (hence $\Phi'^\alpha = \Phi^\alpha$, $\Phi'^\omega = \Phi^\omega$, and $\emptyset \subseteq \Phi^\varepsilon$ as required). To prove 1., we can show

$$\textsc{TSub}\ \dfrac{\textsc{TIntrans}\ \dfrac{\Phi'_e; \Gamma' \vdash e' : \tau \rightsquigarrow \mathcal{R} \quad \alpha \subseteq \Phi'^\alpha_e \quad \omega \subseteq \Phi'^\omega_e}{\Phi'; \Gamma \vdash \mathsf{intx}\ e' : \tau \rightsquigarrow \Phi'_e, \mathcal{R}} \qquad \tau' \leq \tau \qquad \Phi' \leq \Phi'}{\Phi'; \Gamma \vdash \mathsf{intx}\ e' : \tau \rightsquigarrow \Phi'_e, \mathcal{R}}$$

The first premise of [TINTRANS] follows by (i), and the second since $\alpha_e \supseteq \alpha$ and $\omega_e \supseteq \omega$.

To prove 2., we need to show that

$$\textsc{TC2}\ \dfrac{\Phi'_e, \mathcal{R}; H' \vdash \Sigma' \quad \mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \quad \mathsf{f} \in \emptyset \Rightarrow n'' \in ver(H', \mathsf{f})}{[\alpha; \emptyset; \omega], \Phi'_e, \mathcal{R}; H' \vdash ((n'', \sigma'), \Sigma')}$$

We have the first premise by (iii), the second by assumption (since $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$ from the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,dir}$), and the last holds vacuously.

To prove 3., we must show $traceOK((n'', \sigma'), \Sigma')$, which reduces to proving $traceOK(n'', \sigma')$ since we have $traceOK(\Sigma')$ from (iv). We have $traceOK(n', \sigma)$ by assumption. Consider each possible update type:

**case** $dir = bck$ **:**

From the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,bck}$, we know that $n'' = n + 1$. Consider $(\mathsf{f}, \nu) \in \sigma$; it must be the case that $\mathsf{f} \notin \mathrm{dom}(upd^{chg})$. This is because $dir = bck$ implies $\alpha_e \cap \mathrm{dom}(upd^{chg}) = \emptyset$ and by assumption we have $\alpha \subseteq \alpha_e$ (from (TINTRANS)) and $\mathsf{f} \in \alpha$ (from the first premise of [TC1] above). Therefore, since $\mathsf{f} \notin \mathrm{dom}(upd^{chg})$, its $\sigma'$ entry is $(\mathsf{f}, \nu \cup \{n''\})$, which is the required result.

**case** $dir = fwd$ **:**

Since $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,fwd} = (n', \sigma)$, the result is true by assumption.

Part 4. follows directly from (v) and the fact that $\omega_e \supseteq \omega$.

**case** [CONG] **:**

We have that $\langle n; \Sigma; H; \mathbb{E}[e]\rangle \longrightarrow_\mu \langle n+1; \Sigma'; H'; \mathbb{E}[e]\rangle$ follows from $\langle n; \Sigma; H; e\rangle \longrightarrow_\mu \langle n+1; \Sigma'; H'; e\rangle$ by [CONG], where $\mu \equiv (upd, dir)$. Consider the shape of $\mathbb{E}$:

**case** $\_$ **:**

The result follows directly by induction.

**case** $\mathbb{E}\ e_2$ **:**

By assumption, we have $\Phi; \Gamma \vdash (\mathbb{E}\ e_2)[e_1] : \tau \rightsquigarrow \mathcal{R}$. By subtyping derivations (Lemma B.6) we know we can construct a proof derivation of this ending in (TSUB):

$$\text{TSub}\ \frac{\dfrac{\begin{array}{c}\Phi_1; \Gamma \vdash \mathbb{E}[e_1] : \tau_1 \longrightarrow^{\Phi_f} \tau_2' \rightsquigarrow \mathcal{R}_1 \qquad \Phi_2; \Gamma \vdash e_2 : \tau_1 \rightsquigarrow \cdot \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi_s \\ \Phi_3^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3^{\omega} \subseteq \Phi_f^{\omega} \\ \mathbb{E}\,[e_1] \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi_s; \Gamma \vdash (\mathbb{E}\ e_2)[e_1] : \tau_2' \rightsquigarrow \mathcal{R}_1}\ \text{TApp} \qquad \dfrac{\begin{array}{c}\Phi \equiv [\alpha; \varepsilon; \omega] \\ \Phi_s \equiv [\alpha; \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f; \omega] \\ (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f) \subseteq \varepsilon\end{array}}{\Phi_s \leq \Phi}\ \text{SCtxt}}{\Phi; \Gamma \vdash (\mathbb{E}\ e_2)[e_1] : \tau_2 \rightsquigarrow \mathcal{R}_1}\ \text{TSub} \quad \tau_2' \leq \tau_2$$

and by flow effect weakening (Lemma B.7) we know that $\alpha$ and $\omega$ are unchanged in the use of (TSUB).
By inversion on $\langle n; \Sigma; H; (\mathbb{E}\ e_2)[e_1]\rangle \longrightarrow_{\mu} \langle n+1; \Sigma'; H'; (\mathbb{E}\ e_2)[e_1]\rangle$ we have $\langle n; \Sigma; H; e_1\rangle \longrightarrow_{\mu} \langle n+1; \Sigma'; H'; e_1'\rangle$, and then applying [CONG] we have $\langle n; \Sigma; H; \mathbb{E}[e_1]\rangle \longrightarrow_{\mu} \langle n+1; \Sigma'; H'; \mathbb{E}[e_1']\rangle$. From $\Phi, \mathcal{R}_1; H \vdash \Sigma$ we know that:

$$\begin{array}{l}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon \Rightarrow n' \in ver(H, \mathsf{f})\end{array}$$

where $(n', \sigma)$ is the top of $\Sigma$. Since $\Phi \equiv [\alpha; \varepsilon; \omega]$ and $\Phi_s \equiv [\alpha; \varepsilon_s; \omega]$ and $\varepsilon_s = \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$ (where $\varepsilon_3 = \varepsilon_f$), we have

$$\begin{array}{l}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon_1 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}$$

but since $\Phi_1 \equiv [\alpha; \varepsilon_1; \omega_1]$, we have $\Phi_1, \mathcal{R}_1; H \vdash \Sigma$. Hence we can apply induction on $\Phi_1; \Gamma \vdash \mathbb{E}[e_1] : \tau_1 \longrightarrow^{\Phi_f} \tau_2' \rightsquigarrow \mathcal{R}_1$, yielding:
(i) $\Phi_1'; \Gamma' \vdash \mathbb{E}[e_1'] : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \mathcal{R}_1$ and
(ii) $n+1; \Gamma' \vdash H'$
(iii) $\Phi_1', \mathcal{R}_1; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$
(v) $(dir = bck) \Rightarrow n'' \equiv n+1 \ \wedge\ (dir = fwd) \Rightarrow (\mathsf{f} \in \omega_1 \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f}))$
where $\Phi_1' \equiv [\alpha_s; \varepsilon_1'; \omega_1]$ and $\varepsilon_1' \subseteq \varepsilon_1$. Choose $\Phi_2' = [\alpha_1 \cup \varepsilon_1'; \varepsilon_2; \omega_2]$ and $\Phi_3' = [\alpha_1 \cup \varepsilon_1' \cup \varepsilon_2; \varepsilon_f; \omega_s]$ and thus $\Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi_s'$ and $\Phi_3'^{\varepsilon} = \Phi_f^{\varepsilon}$. Let $\Phi' = [\alpha; \varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_f; \omega]$, where $\varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_f \subseteq \varepsilon$, as required.
To prove 1., we have $n+1; \Gamma' \vdash H'$ by (ii), and apply (TAPP):

$$\text{TApp}\ \frac{\begin{array}{c}\Phi_1'; \Gamma' \vdash \mathbb{E}[e_1'] : \tau_1 \longrightarrow^{\Phi_f} \tau_2' \rightsquigarrow \mathcal{R}_1 \qquad \Phi_2'; \Gamma' \vdash e_2 : \tau_1 \rightsquigarrow \cdot \\ \Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi_s' \\ \Phi_3'^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3'^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3'^{\omega} \subseteq \Phi_f^{\omega} \\ \mathbb{E}[e_1'] \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi_s'; \Gamma' \vdash (\mathbb{E}\ e_2)[e_1'] : \tau_2' \rightsquigarrow \mathcal{R}_1}$$

The first premise follows by (i), the second because we have $\Phi_2; \Gamma' \vdash e_2 : \tau_1$ by weakening (since $\Gamma' \supseteq \Gamma$) and then $\Phi_2'; \Gamma' \vdash e_2 : \tau_1$ by flow effect weakening (Lemma B.7) (which we can apply because $\Phi_2'^{\omega} = \Phi_2^{\omega}$, $\Phi_2'^{\varepsilon} = \Phi_2^{\varepsilon}$, $\Phi_2'^{\alpha} = \alpha_1 \cup \varepsilon_1'$ $\Phi_2^{\alpha} = \alpha_1 \cup \varepsilon_1$ hence $\Phi_2'^{\alpha} \subseteq \Phi_2^{\alpha}$) the third—sixth by choice of $\Phi_2'$, $\Phi_3'$ and $\Phi_s'$, and the last as $\mathcal{R}_2 \equiv \cdot$ by assumption. We can now apply (TSUB):

$$\text{TSub}\ \frac{\begin{array}{c}\Phi'; \Gamma \vdash (\mathbb{E}\ e_2)[e_1'] : \tau_2' \rightsquigarrow \mathcal{R}_1 \\ \tau_2' \leq \tau_2 \qquad \Phi' \leq \Phi'\end{array}}{\Phi'; \Gamma \vdash (\mathbb{E}\ e_2)[e_1'] : \tau_2 \rightsquigarrow \mathcal{R}_1'}$$

To prove part 2., we must show that $\Phi', \mathcal{R}_1; H' \vdash \Sigma'$.
By inversion on $\Phi, \mathcal{R}_1; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$ or $\Sigma \equiv (n', \sigma), \Sigma''$. We have two cases:
$\Sigma \equiv (n', \sigma)$**:** By (iii) we must have $\mathcal{R}_1 \equiv \cdot$ such that

$$\text{TC1}\ \frac{\begin{array}{c}\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon_1' \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{[\alpha; \varepsilon_1'; \omega_1], \cdot; H' \vdash (n'', \sigma')}$$

To achieve the desired result we need to prove:

$$\text{TC1}\ \frac{\begin{array}{c}\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_f \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{[\alpha; \varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_f; \omega], \cdot; H' \vdash (n'', \sigma')}$$

The first premise is by assumption (since $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$ from the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd, dir}$). For the second premise, we need to show that for all $\mathsf{f} \in (\varepsilon_2 \cup \varepsilon_f) \Rightarrow n'' \in ver(H', \mathsf{f})$ (for those $\mathsf{f} \in \varepsilon_1'$ the result is by assumption).
Consider each possible update type:

**case** $dir = bck$ **:**

From the definition of $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,bck}$, we know that $n'' = n + 1$; from the definition of $\mathcal{U}[H]_n^{upd}$ we know that $n + 1 \in ver(H', \mathsf{f})$ for all $\mathsf{f}$, hence $n'' \in ver(H', \mathsf{f})$ for all $\mathsf{f}$.

**case** $dir = fwd$ **:**

From (v) we have that $\mathsf{f} \in \omega_1 \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f})$. Since $(\varepsilon_2 \cup \varepsilon_f) \subseteq \omega_1$ (by $\Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi'$), we have $\mathsf{f} \in (\varepsilon_2 \cup \varepsilon_f) \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f})$. By inversion on $\Phi, \mathcal{R}_1; H \vdash \Sigma$ we have $\mathsf{f} \in (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f) \Rightarrow n' \in ver(H, \mathsf{f})$, and thus $\mathsf{f} \in (\varepsilon_2 \cup \varepsilon_f) \Rightarrow n' \in ver(H', \mathsf{f})$. We have $\mathcal{U}[(n', \sigma)]_{n+1}^{upd,fwd} = (n', \sigma)$ hence $n'' = n'$, so finally we have $\mathsf{f} \in (\varepsilon_2 \cup \varepsilon_f) \Rightarrow n'' \in ver(H', \mathsf{f})$.

$\Sigma \equiv (n', \sigma), \Sigma''$ By (iii), we must have $\mathcal{R}_1 \equiv \Phi'', \mathcal{R}''$ such that

$$\text{TC2}\frac{\begin{array}{c}\Phi'', \mathcal{R}''; H' \vdash \Sigma'' \\ \Phi_1' \equiv [\alpha; \varepsilon_1'; \omega_1] \\ \mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon_1' \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{\Phi_1', \Phi'', \mathcal{R}''; H' \vdash ((n'', \sigma'), \Sigma'')}$$

We wish to show that

$$\text{TC2}\frac{\begin{array}{c}\Phi'', \mathcal{R}''; H' \vdash \Sigma'' \\ \Phi' \equiv [\alpha; \varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_f; \omega] \\ \mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in (\varepsilon_1' \cup \varepsilon_2 \cup \varepsilon_f) \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{\Phi', \Phi'', \mathcal{R}''; H' \vdash ((n'', \sigma'), \Sigma'')}$$

$\Phi'', \mathcal{R}''; H' \vdash \Sigma$ follows by assumption while the third and fourth premises follow by the same argument as in the $\Sigma \equiv (n', \sigma)$ case, above.

Part 3. follows directly from (iv).

Part 4. follows directly from (v) and the fact that $\omega_1 \supseteq \omega$ (because $\omega_1 \equiv \varepsilon_2 \cup \varepsilon_f \cup \omega$).

**case** $v\ \mathbb{E}$ **:**

By assumption, we have $\Phi; \Gamma \vdash (v\ \mathbb{E})[e_2] : \tau \rightsquigarrow \mathcal{R}$. By subtyping derivations (Lemma B.6) we have:

$$\text{TSub}\frac{\tau_2' \leq \tau_2 \qquad \text{TApp}\dfrac{\begin{array}{c}\Phi_1; \Gamma \vdash v : \tau_1 \longrightarrow^{\Phi_f} \tau_2' \rightsquigarrow \cdot \qquad \Phi_2; \Gamma \vdash \mathbb{E}[e_2] : \tau_1 \rightsquigarrow \mathcal{R}_2 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi_s \\ \Phi_3^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3^{\omega} \subseteq \Phi_f^{\omega} \\ v \not\equiv v' \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi_s; \Gamma \vdash (v\ \mathbb{E})[e_2] : \tau_2' \rightsquigarrow \mathcal{R}_2} \qquad \text{SCtxt}\dfrac{\begin{array}{c}\Phi \equiv [\alpha; \varepsilon; \omega] \\ \Phi_s \equiv [\alpha; \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f; \omega] \\ (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f) \subseteq \varepsilon\end{array}}{\Phi_s \leq \Phi}}{\Phi; \Gamma \vdash (v\ \mathbb{E})[e_2] : \tau_2 \rightsquigarrow \mathcal{R}_2}$$

and by flow effect weakening (Lemma B.7) we know that $\alpha$ and $\omega$ are unchanged in the use of (TSUB).

By inversion on $\langle n; \Sigma; H; (v\ \mathbb{E})[e_2] \rangle \longrightarrow_\mu \langle n+1; \Sigma'; H'; (v\ \mathbb{E})[e_2] \rangle$ we have $\langle n; \Sigma; H; e_2 \rangle \longrightarrow_\mu \langle n+1; \Sigma'; H'; e_2' \rangle$, and then applying [CONG] we have $\langle n; \Sigma; H; \mathbb{E}[e_2] \rangle \longrightarrow_\mu \langle n+1; \Sigma'; H'; \mathbb{E}[e_2] \rangle$. From $\Phi, \mathcal{R}_2; H \vdash \Sigma$ we know that:

$$\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon \Rightarrow n' \in ver(H, \mathsf{f})\end{array}$$

where $(n', \sigma)$ is the top of $\Sigma$. We have $\Phi \equiv [\alpha; \varepsilon; \omega]$, $\Phi_s \equiv [\alpha_s; \varepsilon_s; \omega_s]$, $\varepsilon_s \subseteq \varepsilon$, $\varepsilon_s = \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3$ (where $\varepsilon_3 = \varepsilon_f$), $\Phi_2 \equiv [\alpha_2; \varepsilon_2; \omega_2]$, $\alpha_2 \equiv \alpha_1 \cup \varepsilon_1 = \alpha$ (since $\varepsilon_1 = \emptyset$; if it's not $\emptyset$ we can construct a derivation for $v$ that has $\varepsilon_1 = \emptyset$ as argued in preservation (Lemma B.15), (TAPP)-[CONG], case $v\ \mathbb{E}$). We have

$$\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon_2 \Rightarrow n' \in ver(H, \mathsf{f})\end{array}$$

hence $\Phi_2, \mathcal{R}_2; H \vdash \Sigma$ and we can apply induction on $\Phi_2; \Gamma \vdash \mathbb{E}[e_2] : \tau_1 \longrightarrow^{\Phi_f} \tau_2' \rightsquigarrow \mathcal{R}_2$, yielding:

(i) $\Phi_2'; \Gamma' \vdash \mathbb{E}[e_2] : \tau_1 \rightsquigarrow \mathcal{R}_2$ and

(ii) $n+1; \Gamma' \vdash H'$

(iii) $\Phi_2', \mathcal{R}_2; H' \vdash \Sigma'$

(iv) $traceOK(\Sigma')$

(v) $(dir = bck) \Rightarrow n'' \equiv n+1 \ \wedge \ (dir = fwd) \Rightarrow (\mathsf{f} \in \omega_2 \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f}))$

where $\Phi_2' \equiv [\alpha_2; \varepsilon_2'; \omega_2]$ and $\varepsilon_2' \subseteq \varepsilon_2$. Choose $\Phi_1' = [\alpha; \emptyset; \omega_2 \cup \varepsilon_2']$ and $\Phi_3' = [\alpha \cup \varepsilon_2'; \varepsilon_f; \omega]$ and thus $\Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi'$ and $\Phi_3'^{\varepsilon} = \Phi_f^{\varepsilon}$.

Let $\Phi' \equiv [\alpha; \varepsilon_2' \cup \varepsilon_f; \omega]$ and thus $\varepsilon_2' \cup \varepsilon_f \subseteq \varepsilon$ as required.

To prove 1., we have $n+1; \Gamma' \vdash H'$ by (ii), and apply (TAPP):

$$\text{TApp}\frac{\begin{array}{c}\Phi'_1; \Gamma' \vdash v : \tau_1 \longrightarrow^{\Phi_f} \tau'_2 \rightsquigarrow \cdot \qquad \Phi'_2; \Gamma' \vdash \mathbb{E}[e_2] : \tau_1 \rightsquigarrow \mathcal{R}_2 \\ \Phi'_1 \rhd \Phi'_2 \rhd \Phi'_3 \hookrightarrow \Phi' \\ \Phi'^{\varepsilon}_3 = \Phi^{\varepsilon}_f \qquad \Phi'^{\alpha}_3 \subseteq \Phi^{\alpha}_f \qquad \Phi'^{\omega}_3 \subseteq \Phi^{\omega}_f \\ v \not\equiv v' \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi'; \Gamma' \vdash (v\ \mathbb{E})[e_2] : \tau'_2 \rightsquigarrow \mathcal{R}_2}$$

The first premise follows by value typing, the second by (i), the third—sixth by choice of $\Phi'_1$ and $\Phi'_3$, and the last holds vacuously. We can now apply (TSUB):

$$\text{TSub}\frac{\begin{array}{c}\Phi'; \Gamma \vdash (v\ \mathbb{E})[e_2] : \tau'_2 \rightsquigarrow \mathcal{R}_2 \\ \tau'_2 \leq \tau_2 \qquad \Phi' \leq \Phi'\end{array}}{\Phi'; \Gamma \vdash (v\ \mathbb{E})[e_2] : \tau_2 \rightsquigarrow \mathcal{R}_2}$$

To prove part 2., we must show that $\Phi', \mathcal{R}_2; H' \vdash \Sigma'$.
By inversion on $\Phi, \mathcal{R}_2; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$ or $\Sigma \equiv (n', \sigma), \Sigma''$. We have two cases:
$\Sigma \equiv (n', \sigma)$**:** By (iii) we must have $\mathcal{R}_2 \equiv \cdot$ such that

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon'_2 \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{[\alpha; \varepsilon'_2; \omega_2], \cdot; H' \vdash (n'', \sigma')}$$

To achieve the desired result we need to prove:

$$\text{TC1}\frac{\begin{array}{c}\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon'_2 \cup \varepsilon_f \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{[\alpha; \varepsilon'_2 \cup \varepsilon_f; \omega], \cdot; H' \vdash (n'', \sigma')}$$

The first premise follows by assumption (since $\mathrm{dom}(\sigma) = \mathrm{dom}(\sigma')$ from the definition of $\mathcal{U}[(n', \sigma)]^{upd,dir}_{n+1}$). For the second premise, we need to show that for all $\mathsf{f} \in \varepsilon_f \Rightarrow n'' \in ver(H', \mathsf{f})$ (for those $\mathsf{f} \in \varepsilon'_2$ the result is by assumption).
Consider each possible update type:
**case** $dir = bck$ **:**
From the definition of $\mathcal{U}[(n', \sigma)]^{upd,bck}_{n+1}$, we know that $n'' = n+1$; from the definition of $\mathcal{U}[H]^{upd}_n$ we know that $n+1 \in ver(H', \mathsf{f})$ for all $\mathsf{f}$, hence $n'' \in ver(H', \mathsf{f})$ for all $\mathsf{f}$.
**case** $dir = fwd$ **:**
From (v) we have that $\mathsf{f} \in \omega_2 \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f})$. Thus $\varepsilon_f \subseteq \omega_2$ (by $\Phi'_1 \rhd \Phi'_2 \rhd \Phi'_3 \hookrightarrow \Phi'$) implies $\mathsf{f} \in \varepsilon_f \Rightarrow ver(H, \mathsf{f}) \subseteq ver(H', \mathsf{f})$. By inversion on $\Phi, \mathcal{R}_2; H \vdash \Sigma$ we have $\mathsf{f} \in (\varepsilon_2 \cup \varepsilon_f) \Rightarrow n' \in ver(H, \mathsf{f})$, and thus $\mathsf{f} \in \varepsilon_f \Rightarrow n' \in ver(H', \mathsf{f})$. We have $\mathcal{U}[(n', \sigma)]^{upd,fwd}_{n+1} = (n', \sigma)$ hence $n'' = n'$, so finally we have $\mathsf{f} \in \varepsilon_f \Rightarrow n'' \in ver(H', \mathsf{f})$.
$\Sigma \equiv (n', \sigma), \Sigma''$ By (iii), we must have $\mathcal{R}_2 \equiv \Phi'', \mathcal{R}''$ such that

$$\text{TC2}\frac{\begin{array}{c}\Phi'', \mathcal{R}''; H' \vdash \Sigma'' \\ \Phi'_2 \equiv [\alpha; \varepsilon'_2; \omega_2] \\ \mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon'_2 \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{\Phi'_2, \Phi'', \mathcal{R}''; H' \vdash ((n'', \sigma'), \Sigma'')}$$

We wish to show that

$$\text{TC2}\frac{\begin{array}{c}\Phi'', \mathcal{R}''; H' \vdash \Sigma'' \\ \Phi' \equiv [\alpha; \varepsilon'_2 \cup \varepsilon_f; \omega] \\ \mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in (\varepsilon'_2 \cup \varepsilon_f) \Rightarrow n'' \in ver(H', \mathsf{f})\end{array}}{\Phi', \Phi'', \mathcal{R}''; H' \vdash ((n'', \sigma'), \Sigma'')}$$

$\Phi'', \mathcal{R}''; H' \vdash \Sigma$ follows by assumption while the third and fourth premises follow by the same argument as in the $\Sigma \equiv (n', \sigma)$ case, above.
Part 3. follows directly from (iv).
Part 4. follows directly from (v) and the fact that $\omega_2 \supseteq \omega$.
**case** all others **:**
Similar to cases above.

□

This lemma says that if take an evaluation step that is not an update, the version set of any $\mathsf{z}$ remains unchanged.

**Lemma B.14** (Non-update step version preservation)**.** *If* $\langle n; \Sigma; H; e\rangle \longrightarrow_{\varepsilon} \langle n; \Sigma'; H';\ e'\rangle$ *then for all* $\mathsf{z} \in \mathrm{dom}(H')$, $ver(H', \mathsf{z}) = ver(H, \mathsf{z})$.

*Proof.* By inspection of the evaluation rules. □

The following lemma states that if we start with a well-typed program and a version-consistent trace and we can take an evaluation step, then afterward we will still have a well-typed program whose trace is version-consistent.

**Lemma B.15** (Preservation)**.**
*Suppose we have the following:*

1. $n \vdash H, e : \tau$ *(such that* $\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}$ *and* $n; \Gamma \vdash H$ *for some* $\Gamma$ *and* $\Phi$*)*
2. $\Phi, \mathcal{R}; H \vdash \Sigma$
3. $traceOK(\Sigma)$
4. $\langle n; \Sigma; H; e\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H';\ e'\rangle$

*Then for some* $\Gamma' \supseteq \Gamma$ *and* $\Phi' \equiv [\Phi^\alpha \cup \varepsilon_0; \varepsilon'; \Phi^\omega]$ *such that* $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$*, we have:*

1. $n \vdash H', e' : \tau$ *where* $\Phi'; \Gamma' \vdash e' : \tau \rightsquigarrow \mathcal{R}'$ *and* $n; \Gamma' \vdash H'$
2. $\Phi', \mathcal{R}'; H' \vdash \Sigma'$
3. $traceOK(\Sigma')$

*Proof.* Induction on the typing derivation $n \vdash H, e : \tau$. By inversion, we have that $\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}$; consider each possible rule for the conclusion of this judgment:

**case** (TINT-TVAR-TGVAR-TLOC) **:**
These expressions do not reduce, so the result is vacuously true.

**case** (TREF) **:**
We have that:

$$(\text{TREF})\frac{\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}}{\Phi; \Gamma \vdash \mathsf{ref}\ e : ref^{\,\varepsilon}\ \tau \rightsquigarrow \mathcal{R}}$$

There are two possible reductions:

**case** [REF] **:**
We have that $e \equiv v$, $\mathcal{R} = \cdot$, and $\langle n; (n', \sigma); H; \mathsf{ref}\ v\rangle \longrightarrow_\emptyset \langle n; (n', \sigma); H'; r\rangle$ where $r \notin \mathrm{dom}(H)$ and $H' = H, r \mapsto (\cdot, v, \emptyset)$.
Let $\Gamma' = \Gamma, r : ref^{\,\varepsilon}\ \tau$ and $\Phi' = \Phi$ (which is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \emptyset$, $\varepsilon' \cup \emptyset \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$), and $\mathcal{R}' = \cdot$. We have part 1. as follows:

$$(\text{TSUB})\frac{(\text{TLOC})\ \dfrac{\Gamma'(r) = ref^{\,\varepsilon}\ \tau}{\Phi_\emptyset; \Gamma' \vdash r : ref^{\,\varepsilon}\ \tau \rightsquigarrow \cdot} \qquad ref^{\,\varepsilon}\ \tau \leq ref^{\,\varepsilon}\ \tau \qquad \Phi_\emptyset \leq \Phi}{\Phi; \Gamma' \vdash r : ref^{\,\varepsilon}\ \tau \rightsquigarrow \cdot}$$

Heap well-formedness $n; \Gamma' \vdash H, r \mapsto (\cdot, v, \emptyset)$ holds since $\Phi_\emptyset; \Gamma' \vdash v : \tau$ follows by value typing (Lemma B.5) from $\Phi; \Gamma' \vdash v : \tau$, which we have by assumption and weakening; we have $n; \Gamma' \vdash H$ by weakening.
To prove 2., we must show $\Phi, \cdot; H' \vdash (n', \sigma)$. This follows by assumption since $H'$ only contains an additional location (i.e., not a global variable) and nothing else has changed. Part 3. follows by assumption since $\Sigma' = \Sigma$.

**case** [CONG] **:**
We have that $\langle n; \Sigma; H; \mathsf{ref}\ \mathbb{E}[e'']\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; \mathsf{ref}\ \mathbb{E}[e''']\rangle$ from $\langle n; \Sigma; H; e''\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e'''\rangle$. By [CONG], we have $\langle n; \Sigma; H; e\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e'\rangle$ where $e \equiv \mathbb{E}[e'']$ and $e' \equiv \mathbb{E}[e''']$.
By induction we have:
(i) $\Phi'; \Gamma' \vdash e' : \tau \rightsquigarrow \mathcal{R}'$ and
(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi', \mathcal{R}'; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$
where $\Phi'^\alpha = \Phi^\alpha \cup \varepsilon_0$, $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$. We prove 1. using (ii), and applying [TREF] using (i):

$$(\text{TREF})\frac{\Phi'; \Gamma' \vdash e' : \tau \rightsquigarrow \mathcal{R}'}{\Phi'; \Gamma' \vdash \mathsf{ref}\ e' : ref^{\,\varepsilon}\ \tau \rightsquigarrow \mathcal{R}'}$$

Part 2. follows directly from (iii), and part 3. follows directly from (iv).

**case** (TDEREF) **:**
We know that

$$(\text{TDEREF})\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e : ref^{\,\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R} \\ \Phi_2^\varepsilon = \varepsilon_r \qquad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash !\, e : \tau \rightsquigarrow \mathcal{R}}$$

We can reduce using either [GVAR-DEREF], [DEREF], or [CONG].

**case** [GVAR-DEREF] **:**
Thus we have $e \equiv \mathsf{z}$ such that

$$\langle n; (n', \sigma); (H'', \mathsf{z} \mapsto (\tau', v, \nu)); !\,\mathsf{z}\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); (H'', \mathsf{z} \mapsto (\tau', v, \nu)); v\rangle$$

(where $H \equiv (H'', \mathsf{z} \mapsto (\tau', v, \nu))$), by subtyping derivations (Lemma B.6) we have

$$\text{(TSUB)}\frac{\text{(TGVAR)}\ \dfrac{\Gamma(\mathsf{z}) = \mathit{ref}^{\varepsilon'_r}\ \tau'}{\Phi_\emptyset; \Gamma \vdash \mathsf{z} : \mathit{ref}^{\varepsilon'_r}\ \tau' \rightsquigarrow \cdot} \qquad \dfrac{\tau' \leq \tau \quad \tau \leq \tau' \quad \varepsilon'_r \subseteq \varepsilon_r}{\mathit{ref}^{\varepsilon'_r}\ \tau' \leq \mathit{ref}^{\varepsilon_r}\ \tau} \qquad \Phi_\emptyset \leq \Phi_1}{\Phi_1; \Gamma \vdash \mathsf{z} : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \cdot}$$

and

$$\text{(TDEREF)}\frac{\Phi_1; \Gamma \vdash \mathsf{z} : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \cdot \quad \Phi_2^\varepsilon = \varepsilon_r \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash !\,\mathsf{z} : \tau \rightsquigarrow \cdot}$$

(where $\mathcal{R} = \cdot$) and $\Phi \equiv [\Phi_1^\alpha; \Phi_1^\varepsilon \cup \varepsilon_r; \Phi_2^\omega]$. Let $\Gamma' = \Gamma$, $\Phi' = [\Phi_1^\alpha \cup \{\mathsf{z}\}; \emptyset; \Phi_2^\omega]$ and $\mathcal{R}' = \mathcal{R} = \cdot$. Since $\mathsf{z} \in \varepsilon_r$ (by $n; \Gamma \vdash H$) we have $\emptyset \cup \{\mathsf{z}\} \subseteq (\Phi_1^\varepsilon \cup \varepsilon_r)$ hence $\varepsilon' \cup \{\mathsf{z}\} \subseteq \Phi^\varepsilon$. The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \{\mathsf{z}\}$, $\varepsilon' \cup \{\mathsf{z}\} \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$.

To prove 1., we need to show that $\Phi'; \Gamma \vdash v : \tau \rightsquigarrow \cdot$ (the rest of the premises follow by assumption of $n \vdash H, !\,\mathsf{z} : \tau$). $H(\mathsf{z}) = (\tau', v, \nu)$ and $\Gamma(\mathsf{z}) = \mathit{ref}^{\varepsilon'_r}\ \tau'$ implies $\Phi'; \Gamma \vdash v : \tau' \rightsquigarrow \cdot$ by $n; \Gamma \vdash H$. The result follows by (TSUB):

$$\text{(TSUB)}\frac{\Phi'; \Gamma \vdash v : \tau' \rightsquigarrow \cdot \quad \tau' \leq \tau \quad \Phi' \leq \Phi'}{\Phi'; \Gamma \vdash v : \tau \rightsquigarrow \cdot}$$

For part 2., we know $\Phi, \cdot; H \vdash (n', \sigma)$:

$$\text{(TC1)}\frac{\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \qquad \mathsf{f} \in (\Phi_1^\varepsilon \cup \varepsilon_r) \Rightarrow n' \in \mathit{ver}(H, \mathsf{f})}{[\Phi_1^\alpha; \Phi_1^\varepsilon \cup \varepsilon_r; \Phi_2^\omega], \cdot; H \vdash (n', \sigma)}$$

and need to prove $\Phi', \cdot; H \vdash (n', \sigma \cup (\mathsf{z}, \nu))$, hence:

$$\text{(TC1)}\frac{\mathsf{f} \in (\sigma \cup (\mathsf{z}, \nu)) \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \{\mathsf{z}\} \qquad \mathsf{f} \in \emptyset \Rightarrow n' \in \mathit{ver}(H, \mathsf{f})}{[\Phi_1^\alpha \cup \{\mathsf{z}\}; \emptyset; \Phi_2^\omega], \cdot; H \vdash (n', \sigma \cup (\mathsf{z}, \nu))}$$

The first premise is true by assumption for all $\mathsf{f} \in \sigma$, and for $(\mathsf{z}, \nu)$ since $\mathsf{z} \in \Phi_1^\alpha \cup \{\mathsf{z}\}$. The second premise is vacuously true.

For part 3., we need to prove $\mathit{traceOK}(n', \sigma \cup (\mathsf{z}, \nu))$; we have $\mathit{traceOK}(n', \sigma)$ by assumption, hence need to prove that $n' \in \nu$. Since by assumption of version consistency we have that $\mathsf{f} \in \Phi_1^\varepsilon \cup \varepsilon_r \Rightarrow n' \in \mathit{ver}(H, \mathsf{f})$ and $\mathit{ver}(H, \mathsf{f}) = \mathit{ver}(H', \mathsf{f}) = \nu$ (by Lemma B.14), and $\{\mathsf{z}\} \subseteq \varepsilon_r$ (by $n; \Gamma \vdash H$), we have $n' \in \nu$.

**case** [DEREF] **:**

Follows the same argument as the [GVAR-DEREF] case above for part 1.; parts 2 and 3 follow by assumption since the trace has not changed.

**case** [CONG] **:**

Here $\langle n; \Sigma; H; !\,e\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; !\,e'\rangle$ follows from $\langle n; \Sigma; H, e\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H', e'\rangle$. To apply induction, we must have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ which follows by Lemma B.9 since $\Phi, \mathcal{R}; H \vdash \Sigma$ and $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$.

Hence we have:

(i) $\Phi_1'; \Gamma' \vdash e' : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R}'$

(ii) $n; \Gamma' \vdash H'$

(iii) $\Phi_1', \mathcal{R}'; H' \vdash \Sigma'$

(iv) $\mathit{traceOK}(\Sigma')$

for some $\Gamma' \supseteq \Gamma$ and some $\Phi_1' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1'; \Phi_1^\omega]$ where $\varepsilon_1' \cup \varepsilon_0 \subseteq \Phi_1^\varepsilon$. Let $\Phi_2' = [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_r; \Phi_2^\omega]$ hence $\Phi_2'^\varepsilon = \varepsilon_r$ and $\Phi_1' \rhd \Phi_2' \hookrightarrow \Phi'$, where $\Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1' \cup \varepsilon_r; \Phi_2^\omega]$ and $(\varepsilon_1' \cup \varepsilon_r) \cup \varepsilon_0 \subseteq (\varepsilon_1 \cup \varepsilon_r)$ hence $\Phi'^\alpha = \Phi^\alpha \cup \varepsilon_0$, $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$ as required.

We prove 1. by (ii) and by applying [TDEREF]:

$$\text{(TDEREF)}\frac{\Phi_1'; \Gamma' \vdash e' : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R}' \quad \Phi_2'^\varepsilon = \varepsilon_r \quad \Phi_1' \rhd \Phi_2' \hookrightarrow \Phi'}{\Phi'; \Gamma' \vdash !\,e' : \tau \rightsquigarrow \mathcal{R}'}$$

The first premise follows from (i) and the second and third premises follows by definition of $\Phi'$ and $\Phi_2'$.

To prove part 2., we must show that $\Phi', \mathcal{R}'; H' \vdash \Sigma'$. We have two cases:

$\Sigma' \equiv (n', \sigma)$**:** By (iii) we must have $\mathcal{R}' \equiv \cdot$ such that

$$\text{(TC1)}\frac{\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \qquad \mathsf{f} \in \varepsilon_1' \Rightarrow n' \in \mathit{ver}(H', \mathsf{f})}{[\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1'; \Phi_1^\omega], \cdot; H' \vdash (n', \sigma)}$$

To achieve the desired result we need to prove:

$$\text{(TC1)}\frac{\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \qquad \mathsf{f} \in (\varepsilon_1' \cup \varepsilon_r) \Rightarrow n' \in \mathit{ver}(H', \mathsf{f})}{[\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1' \cup \varepsilon_r; \Phi_1^\omega], \cdot; H' \vdash (n', \sigma)}$$

The first premise follows directly from (iii). To prove the second premise, we observe that by Lemma B.11, $top(\Sigma) = (n', \sigma')$ where $\sigma' \subseteq \sigma$, and by inversion on $\Phi; \mathcal{R}; H \vdash \Sigma$ we know (a) $\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \Phi_1^\alpha$, and (b) $\mathsf{f} \in \varepsilon_1 \cup \varepsilon_r \Rightarrow n' \in ver(H, \mathsf{f})$. The second premise follows from (iii) and the fact that $\mathsf{f} \in \varepsilon_r \Rightarrow n' \in ver(H, \mathsf{f})$ by (b), and for all $\mathsf{f}$, $ver(H, \mathsf{f}) = ver(H', \mathsf{f})$ by Lemma B.14.

$\Sigma' \equiv (n', \sigma), \Sigma''$**:** By (iii), we must have $\mathcal{R}' \equiv \Phi''', \mathcal{R}'''$ such that

$$\text{(TC2)}\ \frac{\begin{array}{c}\Phi''', \mathcal{R}'''; H' \vdash \Sigma'' \\ \Phi_1' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1'; \Phi_1^\omega] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \\ \mathsf{f} \in \varepsilon_1' \Rightarrow n' \in ver(H', \mathsf{f})\end{array}}{\Phi_1', \Phi''', \mathcal{R}'''; H' \vdash (n', \sigma), \Sigma''}$$

We wish to show that

$$\text{(TC2)}\ \frac{\begin{array}{c}\Phi''', \mathcal{R}'''; H' \vdash \Sigma'' \\ \Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1' \cup \varepsilon_r; \Phi_2^\omega] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \\ \mathsf{f} \in \varepsilon_1' \cup \varepsilon_r \Rightarrow n' \in ver(H', \mathsf{f})\end{array}}{\Phi', \Phi''', \mathcal{R}'''; H' \vdash (n', \sigma), \Sigma''}$$

The first and third premises follow from (iii), while the fourth premise follows by the same argument as in the $\Sigma' \equiv (n', \sigma)$ case, above.

Part 3. follows directly from (iv).

**case** (TASSIGN) **:**

We know that:

$$\text{(TASSIGN)}\ \frac{\begin{array}{cc}\Phi_1; \Gamma \vdash e_1 : ref^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R}_1 & \Phi_2; \Gamma \vdash e_2 : \tau \rightsquigarrow \mathcal{R}_2 \\ \Phi_3^\varepsilon = \varepsilon_r & \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \multicolumn{2}{c}{e_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot}\end{array}}{\Phi; \Gamma \vdash e_1 := e_2 : \tau \rightsquigarrow \mathcal{R}_1 \bowtie \mathcal{R}_2}$$

From $\mathcal{R}_1 \bowtie \mathcal{R}_2$ it follows that either $\mathcal{R}_1 \equiv \cdot$ or $\mathcal{R}_2 \equiv \cdot$.

We can reduce using [GVAR-ASSIGN], [ASSIGN], or [CONG].

**case** [GVAR-ASSIGN] **:**

This implies that $e \equiv \mathsf{z} := v$ with

$$\langle n; (n', \sigma); (H'', \mathsf{z} \mapsto (\tau, v', \nu)); \mathsf{z} := v \rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); (H'', \mathsf{z} \mapsto (\tau, v, \nu)); v \rangle$$

where $H \equiv (H'', \mathsf{z} \mapsto (\tau, v', \nu))$. $\mathcal{R}_1 \equiv \cdot$ and $\mathcal{R}_2 \equiv \cdot$ (thus $\mathcal{R}_1 \bowtie \mathcal{R}_2 \equiv \cdot$).

Let $\Gamma' = \Gamma, \mathcal{R}' = \cdot$, and $\Phi' = [\Phi^\alpha \cup \{\mathsf{z}\}; \emptyset; \Phi^\omega]$. Since $\mathsf{z} \in \varepsilon_r$ (by $n; \Gamma \vdash H$) we have $\emptyset \subseteq (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_r)$, hence $\emptyset \cup \{\mathsf{z}\} \subseteq (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_r)$ which means $\varepsilon' \cup \{\mathsf{z}\} \subseteq \Phi^\varepsilon$. The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \{\mathsf{z}\}$, $\varepsilon' \cup \{\mathsf{z}\} \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$. We prove 1. as follows. Since $\Phi_2; \Gamma \vdash v : \tau \rightsquigarrow \cdot$, by value typing (Lemma B.5) we have $\Phi'; \Gamma \vdash v : \tau \rightsquigarrow \cdot$. $n; \Gamma \vdash H'$ follows from $n; \Gamma \vdash H$ and $\Phi'; \Gamma \vdash v : \tau \rightsquigarrow \cdot$ (since $\Phi^\varepsilon = \emptyset$).

Parts 2. and 3. are similar to the (TDEREF) case.

**case** [ASSIGN] **:**

Part 1. is similar to (GVAR-ASSIGN); we have parts 2. and 3. by assumption.

**case** [CONG] **:**

Consider the shape of $\mathbb{E}$:

**case** $\mathbb{E} := e$ **:**

$\langle n; \Sigma; H; e_1 := e_2 \rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e_1' := e_2 \rangle$ follows from $\langle n; \Sigma; H; e_1 \rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e_1' \rangle$.

Since $e_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot$ by assumption, by Lemma B.10 we have $\Phi_1, \mathcal{R}_1; H \vdash \Sigma$, hence we can apply induction:

(i) $\Phi_1'; \Gamma' \vdash e_1' : ref^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R}_1'$ and

(ii) $n; \Gamma' \vdash H'$

(iii) $\Phi_1', \mathcal{R}_1'; H' \vdash \Sigma'$

(iv) $traceOK(\Sigma')$

for some $\Gamma' \supseteq \Gamma$ and some $\Phi_1' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1'; \Phi_1^\omega]$ where $\varepsilon_1' \cup \varepsilon_0 \subseteq \varepsilon_1$ and $\Phi_1^\omega \equiv \Phi_2^\varepsilon \cup \varepsilon_r \cup \Phi_3^\omega$.

Let $\begin{array}{lcl}\Phi_2' & \equiv & [\Phi_1^\alpha \cup \varepsilon_1' \cup \varepsilon_0; \Phi_2^\varepsilon; \varepsilon_r \cup \Phi_3^\omega] \\ \Phi_3' & \equiv & [\Phi_1^\alpha \cup \varepsilon_1' \cup \varepsilon_0 \cup \Phi_2^\varepsilon; \varepsilon_r; \Phi_3^\omega]\end{array}$

Thus $\Phi_3'^\varepsilon = \varepsilon_r$ and $\Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi'$ such that $\Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1' \cup \Phi_2^\varepsilon \cup \varepsilon_r; \Phi_3^\omega]$ The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \varepsilon_0$, $(\varepsilon_1' \cup \varepsilon_r \cup \varepsilon_2) \cup \varepsilon_0 \subseteq (\varepsilon_1 \cup \varepsilon_r \cup \varepsilon_2)$ i.e., $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$ and $\Phi'^\omega = \Phi^\omega$ as required).

To prove 1., we have $n; \Gamma' \vdash H'$ by (ii), and apply (TASSIGN):

$$\textsc{(TASSIGN)}\ \frac{\textsc{(TSUB)}\ \dfrac{\Phi_2; \Gamma' \vdash e_2 : \tau \rightsquigarrow \mathcal{R}_2 \quad \tau \leq \tau \qquad \dfrac{\Phi_1'; \Gamma' \vdash e_1' : ref^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R}_1' \qquad \begin{array}{c}\Phi_1^\alpha \cup \varepsilon_1' \cup \varepsilon_0 \subseteq \Phi_1^\alpha \cup \Phi_1^\varepsilon \\ \Phi_2^\varepsilon \subseteq \Phi_2^\varepsilon \\ \varepsilon_r \cup \Phi_3^\omega \subseteq \varepsilon_r \cup \Phi_3^\omega\end{array}}{\Phi_2 \leq \Phi_2'}}{\Phi_2'; \Gamma' \vdash e_2 : \tau \rightsquigarrow \mathcal{R}_2} \qquad \begin{array}{c}\Phi_3'^{\varepsilon} = \varepsilon_r \qquad \Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi' \\ e_1' \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi'; \Gamma' \vdash e_1' := e_2 : \tau \rightsquigarrow \mathcal{R}_1' \bowtie \mathcal{R}_2}$$

Note that $\Phi_2; \Gamma' \vdash e_2 : \tau$ follows from $\Phi_2; \Gamma \vdash e_2 : \tau$ by weakening (Lemma B.1).
To prove part 2., we must show that $\Phi', \mathcal{R}_1'; H' \vdash \Sigma'$ (since $\mathcal{R}_1' \bowtie \mathcal{R}_2 = \mathcal{R}_1'$). By inversion on $\Phi, \mathcal{R}; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$ or $\Sigma \equiv (n', \sigma), \Sigma''$. We have two cases:
$\Sigma' \equiv (n', \sigma)$**:** By (iii) we must have $\mathcal{R}_1' \equiv \cdot$ such that

$$\textsc{(TC1)}\ \frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \\ \mathsf{f} \in \varepsilon_1' \Rightarrow n' \in ver(H', \mathsf{f})\end{array}}{[\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1'; \Phi_1^\omega], \cdot; H' \vdash (n', \sigma)}$$

To achieve the desired result we need to prove:

$$\textsc{(TC1)}\ \frac{\begin{array}{c}\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \\ \mathsf{f} \in (\varepsilon_1' \cup \Phi_2^\varepsilon \cup \varepsilon_r) \Rightarrow n' \in ver(H', \mathsf{f})\end{array}}{[\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1' \cup \Phi_2^\varepsilon \cup \varepsilon_r; \Phi_3^\omega], \cdot; H' \vdash (n', \sigma)}$$

The first premise follows directly from (iii). To prove the second premise, we observe that by Lemma B.11, $top(\Sigma) = (n', \sigma')$ where $\sigma' \subseteq \sigma$, and by inversion on $\Phi; \mathcal{R}; H \vdash \Sigma$ we know (a) $\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \Phi_1^\alpha$, and (b) $\mathsf{f} \in \Phi_1^\varepsilon \cup \Phi_2^\varepsilon \cup \varepsilon_r \Rightarrow n' \in ver(H, \mathsf{f})$. The second premise follows from (iii) and the fact that $\mathsf{f} \in \varepsilon_r \Rightarrow n' \in ver(H, \mathsf{f})$ by (b), and for all $\mathsf{f}$, $ver(H, \mathsf{f}) = ver(H', \mathsf{f})$ by Lemma B.14.
$\Sigma' \equiv (n', \sigma), \Sigma''$**:** By (iii), we must have $\mathcal{R}_1' \equiv \Phi''', \mathcal{R}'''$ such that

$$\textsc{(TC2)}\ \frac{\begin{array}{c}\Phi''', \mathcal{R}'''; H' \vdash \Sigma'' \\ \Phi_1' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1'; \Phi_1^\omega] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \\ \mathsf{f} \in \varepsilon_1' \Rightarrow n' \in ver(H', \mathsf{f})\end{array}}{\Phi_1', \Phi''', \mathcal{R}'''; H' \vdash (n', \sigma), \Sigma''}$$

We wish to show that

$$\textsc{(TC2)}\ \frac{\begin{array}{c}\Phi''', \mathcal{R}'''; H' \vdash \Sigma'' \\ \Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_1' \cup \Phi_2^\varepsilon \cup \varepsilon_r; \Phi_3^\omega] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \\ \mathsf{f} \in (\varepsilon_1' \cup \Phi_2^\varepsilon \cup \varepsilon_r) \Rightarrow n' \in ver(H', \mathsf{f})\end{array}}{\Phi', \Phi''', \mathcal{R}'''; H' \vdash (n', \sigma), \Sigma''}$$

The first and third premises follow from (iii), while the fourth premise follows by the same argument as in the $\Sigma' \equiv (n', \sigma)$ case, above.

Part 3. follows directly from (iv).

**case** $r := \mathbb{E}$ **:**

$\langle n; \Sigma; H; r := e_2 \rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; r := e_2' \rangle$ follows from $\langle n; \Sigma; H; e_2 \rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e_2' \rangle$.
Since $e_1 \equiv r$, by inversion $\mathcal{R}_1 \equiv \cdot$. By Lemma B.10 (which we can apply because $\Phi_1^\varepsilon \equiv \emptyset$; if $\Phi_1^\varepsilon \not\equiv \emptyset$ we can rewrite the derivation using value typing to make it so) we have $\Phi_2, \mathcal{R}_2; H \vdash \Sigma$, hence we can apply induction to get:
(i) $\Phi_2'; \Gamma' \vdash e_2' : \tau \rightsquigarrow \mathcal{R}_2'$
(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi_2', \mathcal{R}_2'; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$
for some $\Gamma' \supseteq \Gamma$ and some $\Phi_2' \equiv [\Phi_2^\alpha \cup \varepsilon_0; \varepsilon_2'; \Phi_2^\omega]$ where $(\varepsilon_2' \cup \varepsilon_0) \subseteq \Phi_2^\varepsilon$; note $\Phi_2^\alpha \equiv \Phi_1^\alpha$ (since $\Phi_1^\varepsilon \equiv \emptyset$) and $\Phi_2^\omega \equiv \varepsilon_3 \cup \Phi_3^\omega$.
Let $\begin{array}{lcl}\Phi_1' & \equiv & [\Phi_1^\alpha \cup \varepsilon_0; \emptyset; \varepsilon_2' \cup \varepsilon_r \cup \Phi_3^\omega] \\ \Phi_3' & \equiv & [\Phi_1^\alpha \cup \varepsilon_0 \cup \varepsilon_2'; \varepsilon_r; \Phi_3^\omega]\end{array}$
Thus $\Phi_3'^{\varepsilon} = \varepsilon_r$ and $\Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi'$ such that $\Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_2' \cup \varepsilon_r; \Phi_3^\omega]$ and $(\varepsilon_2' \cup \varepsilon_r) \cup \varepsilon_0 \subseteq (\Phi_2^\varepsilon \cup \varepsilon_r)$. The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \varepsilon_0$, $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$ and $\Phi'^\omega = \Phi^\omega$ as required).
To prove 1., we have $n; \Gamma' \vdash H'$ by (ii), and we can apply [TASSIGN]:

$$\textsc{(TASSIGN)}\ \frac{\begin{array}{c}\Phi_1'; \Gamma' \vdash r : ref^{\varepsilon_r}\ \tau \rightsquigarrow \cdot \qquad \Phi_2'; \Gamma' \vdash e_2' : \tau \rightsquigarrow \mathcal{R}_2' \\ \Phi_3'^{\varepsilon_r} = \varepsilon_r \qquad \Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi' \\ r \not\equiv v \Rightarrow \mathcal{R}_2' = \cdot\end{array}}{\Phi'; \Gamma' \vdash r := e_2' : \tau \rightsquigarrow \cdot \bowtie \mathcal{R}_2'}$$

Note that we have $\Phi_1'; \Gamma' \vdash r : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \cdot$ from $\Phi_1; \Gamma \vdash r : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \cdot$ by value typing and weakening
To prove part 2., we must show that $\Phi', \mathcal{R}_2'; H' \vdash \Sigma'$ (since $\mathcal{R}_1 \bowtie \mathcal{R}_2 = \mathcal{R}_2'$). By inversion on $\Phi, \mathcal{R}; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$ or $\Sigma \equiv (n', \sigma), \Sigma''$. We have two cases:
$\Sigma' \equiv (n', \sigma)$**:** By (iii) we must have $\mathcal{R}_2' \equiv \cdot$ such that

$$\text{(TC1)}\frac{\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_2^\alpha \cup \varepsilon_0 \quad \mathsf{f} \in \varepsilon_2' \Rightarrow n' \in \mathit{ver}(H', \mathsf{f})}{[\Phi_2^\alpha \cup \varepsilon_0; \varepsilon_2'; \Phi_2^\omega], \cdot; H' \vdash (n', \sigma)}$$

To achieve the desired result we need to prove:

$$\text{(TC1)}\frac{\mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_1^\alpha \cup \varepsilon_0 \quad \mathsf{f} \in (\varepsilon_r \cup \varepsilon_2') \Rightarrow n' \in \mathit{ver}(H', \mathsf{f})}{[\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_2' \cup \varepsilon_r; \Phi_3^\omega], \cdot; H' \vdash (n', \sigma)}$$

The first premise follows from (iii) since $\Phi_1^\alpha = \Phi_2^\alpha$.
To prove the second premise, we observe that by Lemma B.11, $\mathit{top}(\Sigma) = (n', \sigma')$ where $\sigma' \subseteq \sigma$, and by inversion on $\Phi; \mathcal{R}; H \vdash \Sigma$ we know (a) $\mathsf{f} \in \sigma' \Rightarrow \mathsf{f} \in \Phi_1^\alpha$, and (b) $\mathsf{f} \in \varepsilon_r \cup \Phi_2^\varepsilon \Rightarrow n' \in \mathit{ver}(H, \mathsf{f})$. The second premise follows from (iii) and the fact that $\mathsf{f} \in \varepsilon_r \Rightarrow n' \in \mathit{ver}(H, \mathsf{f})$ by (b), and for all $\mathsf{f}$, $\mathit{ver}(H, \mathsf{f}) = \mathit{ver}(H', \mathsf{f})$ by Lemma B.14.
$\Sigma' \equiv (n', \sigma), \Sigma''$**:** By (iii), we must have $\mathcal{R}_2' \equiv \Phi''', \mathcal{R}'''$ such that:

$$\text{(TC2)}\frac{\Phi''', \mathcal{R}'''; H' \vdash \Sigma'' \quad \Phi_2' \equiv [\Phi_2^\alpha \cup \varepsilon_0; \varepsilon_2'; \Phi_2^\omega] \quad \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi_2^\alpha \cup \varepsilon_0 \quad \mathsf{f} \in \varepsilon_2' \Rightarrow n' \in \mathit{ver}(H', \mathsf{f})}{\Phi_2', \Phi''', \mathcal{R}'''; H' \vdash (n', \sigma), \Sigma''}$$

We wish to show that

$$\text{(TC2)}\frac{\Phi''', \mathcal{R}'''; H' \vdash \Sigma'' \quad \Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_2' \cup \varepsilon_r; \Phi_3^\omega] \quad \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \cup \varepsilon_0 \quad \mathsf{f} \in \varepsilon_2' \cup \varepsilon_r \Rightarrow n' \in \mathit{ver}(H', \mathsf{f})}{\Phi', \Phi''', \mathcal{R}'''; H' \vdash (n', \sigma), \Sigma''}$$

The first and third premises follow from (iii), while the fourth premise follows by the same argument as in the $\Sigma' \equiv (n', \sigma)$ case, above.

Part 3. follows directly from (iv).

**case** (TUPDATE) **:**

**case** [NO-UPDATE] **:**
Thus we must have

$$\langle n; (n', \sigma); H; \mathsf{update}^{\alpha'', \omega''} \rangle \longrightarrow \langle n; (n', \sigma); H; 0 \rangle$$

Let $\Gamma' = \Gamma$ and $\Phi' = \Phi$ (and thus $\varepsilon \cup \emptyset \subseteq \Phi^\varepsilon$, $\Phi'^\alpha = \Phi^\alpha \cup \emptyset$, and $\Phi'^\omega = \Phi^\omega$) as required. For 1., $\Phi; \Gamma \vdash 0 : \mathit{int} \rightsquigarrow \cdot$ follows from (TINT) and value typing and $n; \Gamma \vdash H$ is true by assumption. Parts 2. and 3. follow by assumption.

**case** (TIF) **:**
We know that:

$$\text{(TIF)}\frac{\Phi_1; \Gamma \vdash e_1 : \mathit{int} \rightsquigarrow \mathcal{R} \quad \Phi_2; \Gamma \vdash e_2 : \tau \rightsquigarrow \cdot \quad \Phi_2; \Gamma \vdash e_3 : \tau \rightsquigarrow \cdot \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau \rightsquigarrow \mathcal{R}}$$

We can reduce using [IF-T], [IF-F] or [CONG].

**case** [IF-T] **:**
This implies that $e_1 \equiv v$ hence $\mathcal{R} = \cdot$. We have

$$\langle n; (n', \sigma); H; \mathsf{if0}\ v\ \mathsf{then}\ e2\ \mathsf{else}\ e3 \rangle \longrightarrow \langle n; (n', \sigma); H; e2 \rangle$$

Let $\Gamma' = \Gamma$ and $\Phi' = \Phi$ (and thus $\varepsilon \cup \emptyset \subseteq \Phi^\varepsilon$, $\Phi'^\alpha = \Phi^\alpha \cup \emptyset$, and $\Phi'^\omega = \Phi^\omega$) as required. To prove 1., we have $n; \Gamma \vdash H$ by assumption, and we have

$$\text{(TSUB)}\frac{\Phi_2; \Gamma \vdash e_2 : \tau \rightsquigarrow \cdot \quad \tau \leq \tau \quad \Phi_2 \leq \Phi}{\Phi; \Gamma \vdash e_2 : \tau \rightsquigarrow \cdot}$$

The first premise holds by assumption, the second by reflexivity of subtyping, and the third by Lemma B.2.

**case** [IF-F] **:**
This is similar to [IF-T].

**case** [CONG] **:**
$\langle n; \Sigma; H; \mathsf{if0}\ e_1\ \mathsf{then}\ e2\ \mathsf{else}\ e3 \rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; \mathsf{if0}\ e_1'\ \mathsf{then}\ e2\ \mathsf{else}\ e3 \rangle$ follows from $\langle n; \Sigma; H; e_1 \rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e_1' \rangle$. To apply induction, we must have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ which follows by Lemma B.9 since $\Phi, \mathcal{R}; H \vdash \Sigma$ and $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$. Hence we have:
(i) $\Phi_1'; \Gamma' \vdash e_1' : \mathit{int} \rightsquigarrow \mathcal{R}'$ and

(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi'_1, \mathcal{R}'; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$
for some $\Gamma' \supseteq \Gamma$ and some $\Phi'_1 \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon'_1; \Phi_1^\omega]$ where $\varepsilon'_1 \cup \varepsilon_0 \subseteq \Phi_1^\varepsilon$. (Note that $\Phi_1^\omega \equiv \Phi_2^\varepsilon \cup \Phi_2^\omega$.)
Let $\Phi'_2 \equiv [\Phi_1^\alpha \cup \varepsilon'_1 \cup \varepsilon_0; \Phi_2^\varepsilon; \Phi_2^\omega]$. Thus $\Phi'_1 \rhd \Phi'_2 \hookrightarrow \Phi'$ so that $\Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon'_1 \cup \Phi_2^\varepsilon; \Phi_2^\omega]$ where $\varepsilon'_1 \cup \varepsilon_0 \cup \Phi_2^\varepsilon \subseteq \Phi_1^\varepsilon \cup \Phi_2^\varepsilon$ and $\Phi'^\omega = \Phi^\omega$ as required.
To prove 1., we have $n; \Gamma' \vdash H'$ by (ii), and can apply (TIF): We prove 1. by (ii) and as follows:

$$\text{(TIF)}\ \frac{\text{(TSUB)}\ \dfrac{\Phi_2; \Gamma' \vdash e_2 : \tau \rightsquigarrow \cdot \quad \tau \leq \tau}{\Phi_2 \leq \Phi'_2} \qquad \text{(TSUB)}\ \dfrac{\Phi_2; \Gamma' \vdash e_2 : \tau \rightsquigarrow \cdot \quad \tau \leq \tau}{\Phi_2 \leq \Phi'_2}}{\dfrac{\Phi'_2; \Gamma' \vdash e_2 : \tau \rightsquigarrow \cdot \qquad \Phi'_1; \Gamma' \vdash e'_1 : int \rightsquigarrow \mathcal{R}'_1 \qquad \Phi'_1 \rhd \Phi'_2 \hookrightarrow \Phi'}{\Phi'; \Gamma' \vdash \mathsf{if0}\ e'_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau \rightsquigarrow \mathcal{R}'}}$$

Note that $\Phi_2; \Gamma' \vdash e_2 : \tau \rightsquigarrow \mathcal{R}$ follows from $\Phi_2; \Gamma \vdash e_2 : \tau \rightsquigarrow \mathcal{R}$ by weakening (Lemma B.1) and likewise for $\Phi_2; \Gamma' \vdash e_3 : \tau \rightsquigarrow \mathcal{R}$ .
Parts 2. and 3. follow by an argument similar to (TDEREF)-[CONG] and (TASSIGN)-[CONG].

**case** (TTRANSACT) **:**
We know that:

$$\text{(TTRANSACT)}\ \frac{\Phi''; \Gamma \vdash e : \tau \rightsquigarrow \cdot \quad \Phi^\alpha \subseteq \Phi''^\alpha \quad \Phi^\omega \subseteq \Phi''^\omega}{\Phi; \Gamma \vdash \mathsf{tx}\ e : \tau \rightsquigarrow \cdot}$$

We can reduce using [TX-START]:

$$\langle n; (n', \sigma); H; \mathsf{tx}\ e \rangle \longrightarrow \langle n; (n, \emptyset), (n', \sigma); H; \mathsf{intx}\ e \rangle$$

Let $\Gamma' = \Gamma$ and $\Phi' \equiv [\Phi^\alpha; \emptyset; \Phi^\omega]$ (and thus $\emptyset \cup \emptyset \subseteq \Phi^\varepsilon$, $\Phi'^\alpha = \Phi^\alpha \cup \emptyset$, and $\Phi'^\omega = \Phi^\omega$, as required). To prove 1., we have $n; \Gamma \vdash H$ by assumption, and the rest follows by (TINTRANS):

$$\text{(TINTRANS)}\ \frac{\Phi''; \Gamma \vdash e : \tau \rightsquigarrow \cdot \quad \Phi'^\alpha \subseteq \Phi''^\alpha \quad \Phi'^\omega \subseteq \Phi''^\omega}{\Phi'; \Gamma \vdash \mathsf{intx}\ e : \tau \rightsquigarrow \Phi'', \cdot}$$

The first premise is true by assumption, and the second by choice of $\Phi'$.
We prove 2. as follows:

$$\text{(TC2)}\ \frac{\text{(TC1)}\ \dfrac{\mathsf{f} \in \emptyset \Rightarrow \mathsf{f} \in \Phi''^\alpha \quad \mathsf{f} \in \Phi''^\varepsilon \Rightarrow n \in ver(H, \mathsf{f})}{\Phi'', \cdot; H \vdash (n, \emptyset)} \qquad \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \Phi^\alpha \quad \mathsf{f} \in \emptyset \Rightarrow n' \in ver(H, \mathsf{f})}{[\Phi^\alpha; \emptyset; \Phi^\omega], \Phi'', \cdot; H \vdash (n, \emptyset), (n', \sigma)}$$

First premise of [TC1] is true vacuously, and the second is true by $n; \Gamma \vdash H$, which we have by assumption. For [TC2], the first premise holds by inversion of $\Phi, \cdot; H \vdash (n', \sigma)$, which we have by assumption, and the second holds vacuously.
Part 3. follows easily: we have $traceOK((n', \sigma))$ by assumption, $traceOK((n, \emptyset))$ is vacuously true, hence $traceOK((n, \emptyset), (n', \sigma))$ is true.

**case** (TINTRANS) **:**
We know that:

$$\text{(TINTRANS)}\ \frac{\Phi''; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R} \quad \Phi^\alpha \subseteq \Phi''^\alpha \quad \Phi^\omega \subseteq \Phi''^\omega}{\Phi; \Gamma \vdash \mathsf{intx}\ e : \tau \rightsquigarrow \Phi'', \mathcal{R}}$$

There are two possible reductions:

**case** [TX-END] **:**
We have that $e \equiv v$ and thus $\mathcal{R} \equiv \cdot$; we reduce as follows:

$$\frac{traceOK(n'', \sigma')}{\langle n; ((n', \sigma'), (n'', \sigma'')); H; \mathsf{intx}\ v \rangle \longrightarrow \langle n; (n', \sigma); H; v \rangle}$$

Let $\Phi' = \Phi$ and $\Gamma' = \Gamma$ (and thus $\Phi'^\alpha = \Phi^\alpha \cup \emptyset$, $\varepsilon' \cup \emptyset \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$ as required). To prove 1., we know that $n; \Gamma \vdash H$ follows by assumption and $\Phi; \Gamma \vdash v : \tau \rightsquigarrow \cdot$ by value typing. To prove 2., we must show that $\Phi, \cdot; H \vdash (n', \sigma)$, but this is true by inversion on $\Phi, \Phi'', \cdot; H \vdash ((n', \sigma'), (n'', \sigma''))$.
For 3., $traceOK((n', \sigma))$ follows from $traceOK(((n', \sigma'), (n'', \sigma'')))$ (which is true by assumption).

**case** [TX-CONG-2] **:**
We know that

$$\frac{\langle n; \Sigma; H; e \rangle \longrightarrow_\varepsilon \langle n'; \Sigma'; H'; e' \rangle}{\langle n; \Sigma; H; \mathsf{intx}\ e \rangle \longrightarrow_\emptyset \langle n'; \Sigma'; H'; \mathsf{intx}\ e' \rangle}$$

follows from $\langle n; \Sigma; H; e\rangle \longrightarrow_\eta \langle n; \Sigma'; H'; e'\rangle$ (because the reduction does not perform an update, hence $\eta \equiv \varepsilon_0$ and we apply [TX-CONG-2]).
We have $\Phi'', \mathcal{R}; H \vdash \Sigma$ by inversion on $\Phi, \Phi'', \mathcal{R}; H \vdash ((n', \sigma), \Sigma)$, hence by induction:
(i) $\Phi'''; \Gamma' \vdash e' : \tau \rightsquigarrow \mathcal{R}'$ and
(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi''', \mathcal{R}'; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$
for some $\Gamma' \supseteq \Gamma$ and some $\Phi'''$ such that $\Phi'''^{\alpha} = \Phi''^{\alpha} \cup \varepsilon_0$, $\varepsilon''' \cup \varepsilon_0 \subseteq \Phi''^{\varepsilon}$, and $\Phi'''^{\omega} = \Phi''^{\omega}$.
Let $\Phi' = \Phi$ (hence $\Phi'^{\alpha} = \Phi'^{\alpha} \cup \emptyset$ , $\varepsilon' \cup \emptyset \subseteq \Phi^{\varepsilon}$, and $\Phi'^{\omega} = \Phi^{\omega}$ as required) and $\Gamma' = \Gamma$.
To prove 1., we have $n; \Gamma' \vdash H'$ by (ii), and we can apply [TINTRANS]:

$$(\text{TINTRANS})\ \frac{\Phi'''; \Gamma' \vdash e' : \tau \rightsquigarrow \mathcal{R}' \quad \Phi'^{\alpha} \subseteq \Phi'''^{\alpha} \quad \Phi'^{\omega} \subseteq \Phi'''^{\omega}}{\Phi'; \Gamma' \vdash \mathsf{intx}\ e' : \tau \rightsquigarrow \Phi''', \mathcal{R}'}$$

The first premise follows from (i), and the second holds because $\Phi^{\alpha} \subseteq \Phi''^{\alpha}$ and $\Phi^{\omega} \subseteq \Phi''^{\omega}$ by assumption and we picked $\Phi' = \Phi$ (hence $\Phi'^{\alpha} \subseteq \Phi'''^{\alpha}$ $\Phi'^{\omega} \subseteq \Phi'''^{\omega}$).
Part 2. follows directly from (iii). Part 3. follows directly from (iv).

**case** (TLET) **:**
We know that:

$$(\text{TLET})\ \frac{\Phi_1; \Gamma \vdash e_1 : \tau_1 \rightsquigarrow \mathcal{R} \quad \Phi_2; \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \rightsquigarrow \cdot \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash \mathsf{let}\ x : \tau_1 = e_1\ \mathsf{in}\ e_2 : \tau_2 \rightsquigarrow \mathcal{R}}$$

We can reduce using either [LET] or [CONG].

**case** [LET] **:**
This implies that $e_1 \equiv v$ hence $\mathcal{R} \equiv \cdot$. We have:

$$\langle n; (n', \sigma); H; \mathsf{let}\ x : \tau = v\ \mathsf{in}\ e\rangle \longrightarrow \langle n; (n', \sigma); H; e[x \mapsto v]\rangle$$

To prove 1., we have $n; \Gamma \vdash H$ by assumption; let $\Gamma' = \Gamma$ and $\Phi' = \Phi$; since $\varepsilon_2 \subseteq (\varepsilon_1 \cup \varepsilon_2)$, we can apply [TSUB]:

$$(\text{TSUB})\ \frac{\Phi_2; \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \rightsquigarrow \cdot \quad \tau_2 \leq \tau_2 \quad \Phi_2 \leq \Phi}{\Phi; \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \rightsquigarrow \cdot}$$

The first premise holds by assumption, the second by reflexivity of subtyping, and the third by Lemma B.2. By value typing we have $\Phi; \Gamma \vdash v : \tau_1 \rightsquigarrow \cdot$, so by substitution (Lemma B.17) we have $\Phi; \Gamma \vdash e_2[x \mapsto v] : \tau_2 \rightsquigarrow \cdot$.
Parts 2. and 3. hold by assumption.

**case** [CONG] **:**
Similar to (TIF)-[CONG].

**case** (TAPP) **:**
We know that:

$$(\text{TAPP})\ \frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \mathcal{R}_1 \quad \Phi_2; \Gamma \vdash e_2 : \tau_1 \rightsquigarrow \mathcal{R}_2 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \Phi_3^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3^{\omega} \subseteq \Phi_f^{\omega} \\ e_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2 \rightsquigarrow \mathcal{R}_1 \bowtie \mathcal{R}_2}$$

We can reduce using either [CALL] or [CONG].

**case** [CALL] **:**
We have that

$$\langle n; (n', \sigma); (H'', \mathsf{z} \mapsto (\tau, \lambda(x).e, \nu)); \mathsf{z}\ v\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); (H'', \mathsf{z} \mapsto (\tau, \lambda(x).e, \nu)); e[x \mapsto v]\rangle$$

(where $H \equiv (H'', \mathsf{z} \mapsto (\tau, \lambda(x).e, \nu))$), and

$$(\text{TAPP})\ \frac{\begin{array}{c}\Phi_1; \Gamma \vdash \mathsf{z} : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \cdot \quad \Phi_2; \Gamma \vdash v : \tau_1 \rightsquigarrow \cdot \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \Phi_3^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3^{\omega} \subseteq \Phi_f^{\omega} \\ \mathsf{z} \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi; \Gamma \vdash \mathsf{z}\ v : \tau_2 \rightsquigarrow \cdot}$$

where by subtyping derivations (Lemma B.6) we have

$$(\text{TSUB})\ \frac{(\text{TGVAR})\ \dfrac{\Gamma(\mathsf{z}) = \tau_1' \longrightarrow^{\Phi_f'} \tau_2'}{\Phi_\emptyset; \Gamma \vdash \mathsf{z} : \tau_1' \longrightarrow^{\Phi_f'} \tau_2' \rightsquigarrow \cdot} \qquad \dfrac{\tau_1 \leq \tau_1' \quad \tau_2' \leq \tau_2 \quad \Phi_f' \leq \Phi_f}{\tau_1' \longrightarrow^{\Phi_f'} \tau_2' \leq \tau_1 \longrightarrow^{\Phi_f} \tau_2} \qquad \Phi_\emptyset \leq \Phi_1}{\Phi_1; \Gamma \vdash \mathsf{z} : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \cdot}$$

Define $\Phi_f \equiv [\alpha_f; \varepsilon_f; \omega_f]$ and $\Phi'_f \equiv [\alpha'_f; \varepsilon'_f; \omega'_f]$.
Let $\Gamma' = \Gamma$, $\mathcal{R}' = \cdot$ and choose $\Phi' = [\Phi_1^\alpha \cup \{\mathsf{z}\}; \varepsilon_f; \Phi_3^\omega]$. Since $\mathsf{z} \in \varepsilon'_f$ (by $n; \Gamma \vdash H$) and $\varepsilon'_f \subseteq \varepsilon_f$ (by $\Phi'_f \leq \Phi_f$) we have $\varepsilon_f \cup \{\mathsf{z}\} \subseteq (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f)$. The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \{\mathsf{z}\}$, $\Phi'^\varepsilon \cup \{\mathsf{z}\} \subseteq \Phi^\varepsilon$, and $\Phi'^\omega = \Phi^\omega$. For 1., we have $n; \Gamma \vdash H'$ by assumption; for the remainder we have to prove $\Phi'; \Gamma \vdash e[x \mapsto v] : \tau_2 \rightsquigarrow \cdot$. First, we must prove that $\Phi'_f \leq \Phi'$. Note that since $\{\mathsf{z}\} \subseteq \alpha_f$ by $n; \Gamma \vdash H'$, from $\Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi$ and choice of $\Phi'$ we get $\Phi_3'^\alpha \cup \{\mathsf{z}\} \subseteq \alpha_f$. We have:

$$\begin{array}{lcll}
\Phi' & \equiv & [\Phi_1^\alpha \cup \{\mathsf{z}\}; \varepsilon_f; \Phi_3^\omega] & \text{(by choice of } \Phi') \\
\Phi_f & \equiv & [\alpha_f; \varepsilon_f; \omega_f] & \\
\Phi'_f & \equiv & [\alpha'_f; \varepsilon'_f; \omega'_f] & \\
\varepsilon'_f & \subseteq & \varepsilon_f & \text{(by } \Phi'_f \leq \Phi_f) \\
\alpha_f & \subseteq & \alpha'_f & \text{(by } \Phi'_f \leq \Phi_f) \\
\omega_f & \subseteq & \omega'_f & \text{(by } \Phi'_f \leq \Phi_f) \\
\Phi_3'^\alpha \cup \{\mathsf{z}\} & \subseteq & \alpha_f & \text{(by assumption and choice of } \Phi') \\
\Phi_3'^\alpha & = & \Phi_1^\alpha \cup \Phi_1^\varepsilon \cup \Phi_2'^\varepsilon & \text{(by } \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi) \\
\Phi_3'^\omega & \subseteq & \omega_f & \text{(by assumption and choice of } \Phi')
\end{array}$$

Thus we have the result by [TSUB]

$$\frac{\Phi'_f; \Gamma \vdash e[x \mapsto v] : \tau'_2 \rightsquigarrow \cdot \qquad \tau'_2 \leq \tau_2 \qquad \Phi'_f \leq \Phi'}{\Phi'; \Gamma \vdash e[x \mapsto v] : \tau_2}$$

By assumption, we have $\Phi_2; \Gamma \vdash v : \tau_1 \rightsquigarrow \cdot$. By value typing and $\tau_1 \leq \tau'_1$ we have $\Phi'; \Gamma \vdash v : \tau'_1 \rightsquigarrow \cdot$.
Finally by substitution we have $\Phi'; \Gamma \vdash e[x \mapsto v] : \tau_2 \rightsquigarrow \cdot$.
For part 2., we need to prove $\Phi', \cdot; H \vdash (n'', \sigma')$ where $\sigma' = \sigma \cup (\mathsf{z}, \nu)$ and $n'' = n'$, hence:

$$\text{(TC1)}\ \frac{\begin{array}{c}\mathsf{f} \in (\sigma \cup (\mathsf{z}, \nu)) \Rightarrow \mathsf{f} \in \Phi^\alpha \cup \{\mathsf{z}\} \\ \mathsf{f} \in \varepsilon_f \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi', \cdot; H \vdash (n'', \sigma')}$$

The first premise is true by assumption and the fact that $\{\mathsf{z}\} \subseteq \{\mathsf{z}\}$. The second premise is true by assumption.
For part 3., we need to prove $traceOK(\sigma \cup (\mathsf{z}, \nu))$; we have $traceOK(\sigma)$ by assumption, hence need to prove that $n' \in \nu$. Since by assumption we have that $\mathsf{f} \in \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f \Rightarrow n' \in ver(H, \mathsf{f})$ and $\{\mathsf{z}\} \subseteq \varepsilon_f$, we have $n' \in \nu$.

**case** [CONG] **:**

**case** $\mathbb{E}\ e$ **:**
$\langle n; \Sigma; H; e_1\ e_2\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e'_1\ e_2\rangle$ follows from $\langle n; \Sigma; H; e_1\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e'_1\rangle$.
Since $e_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot$ by assumption, by Lemma B.10 we have $\Phi_1, \mathcal{R}_1; H \vdash \Sigma$ hence we can apply induction:
(i) $\Phi'_1; \Gamma' \vdash e'_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \mathcal{R}'_1$ and
(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi'_1, \mathcal{R}'_1; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$
for some $\Gamma' \supseteq \Gamma$ and some $\Phi'_1 \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon'_1; \Phi_1^\omega]$ where $\varepsilon'_1 \cup \varepsilon_0 \subseteq \varepsilon_1$ and $\Phi_1^\omega \equiv \Phi_2^\varepsilon \cup \varepsilon_f \cup \Phi_3^\omega$.
Let $\begin{array}{lcl}\Phi'_2 & \equiv & [\Phi_1^\alpha \cup \varepsilon'_1 \cup \varepsilon_0; \Phi_2^\varepsilon; \varepsilon_f \cup \Phi_3^\omega] \\ \Phi'_3 & \equiv & [\Phi_1^\alpha \cup \varepsilon'_1 \cup \varepsilon_0 \cup \Phi_2^\varepsilon; \varepsilon_f; \Phi_3^\omega]\end{array}$
Thus $\Phi_3'^\varepsilon = \varepsilon_f$, $\Phi'_1 \rhd \Phi'_2 \rhd \Phi'_3 \hookrightarrow \Phi'$, $\Phi_3'^\alpha \subseteq \Phi_f^\alpha$ and $\Phi_3'^\omega \subseteq \Phi_f^\omega$ (since $\Phi_3'^\alpha \cup \varepsilon_0 \subseteq \Phi_3^\alpha$ and $\Phi_3'^\omega = \Phi_3^\omega$). We have $\Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon'_1 \cup \Phi_2^\varepsilon \cup \varepsilon_f; \Phi_3^\omega]$. The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \varepsilon_0$, $(\varepsilon'_1 \cup \varepsilon_f \cup \varepsilon_2) \cup \varepsilon_0 \subseteq (\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_f)$ i.e., $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$ and $\Phi'^\omega = \Phi^\omega$ as required).
To prove 1., we have $n; \Gamma' \vdash H'$ by (ii), and apply (TAPP):

$$\text{(TAPP)}\ \frac{\text{(TSUB)}\ \dfrac{\Phi'_2; \Gamma' \vdash e_2 : \tau_1 \rightsquigarrow \mathcal{R}_2 \qquad \tau_1 \leq \tau_1 \qquad \dfrac{\Phi'_1; \Gamma' \vdash e'_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \mathcal{R}'_1 \qquad \begin{array}{c}\Phi_1^\alpha \cup \varepsilon'_1 \cup \varepsilon_0 \subseteq \Phi_1^\alpha \cup \Phi_1^\varepsilon \\ \Phi_2^\varepsilon \subseteq \Phi_2^\varepsilon \\ \varepsilon_f \cup \Phi_3^\omega \subseteq \varepsilon_f \cup \Phi_3^\omega\end{array}}{\Phi_2 \leq \Phi'_2}}{\Phi'_2; \Gamma' \vdash e_2 : \tau_1 \rightsquigarrow \mathcal{R}_2} \qquad \begin{array}{c}\Phi'_1 \rhd \Phi'_2 \rhd \Phi'_3 \hookrightarrow \Phi' \\ \Phi_3'^\varepsilon = \Phi_f^\varepsilon \qquad \Phi_3'^\alpha \subseteq \Phi_f^\alpha \qquad \Phi_3'^\omega \subseteq \Phi_f^\omega \\ e'_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi'; \Gamma' \vdash e'_1\ e_2 : \tau_2 \rightsquigarrow \mathcal{R}'_1 \bowtie \mathcal{R}_2}$$

Note that $\Phi_2; \Gamma' \vdash e_2 : \tau_1 \rightsquigarrow \mathcal{R}_2$ follows from $\Phi_2; \Gamma \vdash e_2 : \tau_1 \rightsquigarrow \mathcal{R}_2$ by weakening (Lemma B.1). The last premise holds vacuously as $\mathcal{R}_2 \equiv \cdot$ by assumption.
To prove part 2., we must show that $\Phi', \mathcal{R}'; H' \vdash \Sigma'$. The proof is similar to the (TASSIGN)-[CONG] proof, case $\mathbb{E} := e$ but substituting $\varepsilon_f$ for $\varepsilon_r$.
Part 3. follows directly from (iv).

**case** $v\ \mathbb{E}$ **:**
$\langle n; \Sigma; H; v\ e_2\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; v\ e'_2\rangle$ follows from $\langle n; \Sigma; H; e_2\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e'_2\rangle$.

For convenience, we make $\Phi_1^\varepsilon \equiv \emptyset$; if $\Phi_1^\varepsilon \not\equiv \emptyset$, we can always construct a typing derivation of $v$ that uses value typing to make $\Phi_1^\varepsilon \equiv \emptyset$. Note that $\Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi$ would still hold since Lemma B.7 allows us to decrease $\Phi_2^\alpha$ to satisfy $\Phi_2^\alpha = \Phi_1^\alpha \cup \Phi_1^\varepsilon$; similarly, since $\Phi_3^\alpha = \Phi_1^\alpha \cup \Phi_1^\varepsilon \cup \Phi_2^\varepsilon$ we know that $\Phi_3^\alpha \subseteq \Phi_f^\alpha$ would still hold if $\Phi_3^\alpha$ was smaller as a result of shrinking $\Phi_1^\varepsilon$ to be $\emptyset$.

Since $e_1 \equiv v$, by inversion $\mathcal{R}_1 \equiv \cdot$ and by Lemma B.10 (which we can apply since $\Phi_1^\varepsilon \equiv \emptyset$), we have $\Phi_2, \mathcal{R}_2; H \vdash \Sigma$; hence by induction:

(i) $\Phi_2'; \Gamma' \vdash e_2' : \tau_1 \rightsquigarrow \mathcal{R}_2'$
(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi_2', \mathcal{R}_2'; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$

for some $\Gamma' \supseteq \Gamma$ and some $\Phi_2' \equiv [\Phi_2^\alpha \cup \varepsilon_0; \varepsilon_2'; \Phi_2^\omega]$ where $(\varepsilon_2' \cup \varepsilon_0) \subseteq \Phi_2^\varepsilon$; note $\Phi_2^\alpha \equiv \Phi_1^\alpha$ (since $\Phi_1^\varepsilon \equiv \emptyset$) and $\Phi_2^\omega \equiv \varepsilon_3 \cup \Phi_3^\omega$.

Let $\begin{array}{lcl} \Phi_1' & \equiv & [\Phi_1^\alpha \cup \varepsilon_0; \emptyset; \varepsilon_2' \cup \varepsilon_f \cup \Phi_3^\omega] \\ \Phi_3' & \equiv & [\Phi_1^\alpha \cup \varepsilon_0 \cup \varepsilon_2'; \varepsilon_f; \Phi_3^\omega] \end{array}$

Thus $\Phi_3'^\varepsilon = \varepsilon_f$, $\Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi'$, $\Phi_3'^\alpha \subseteq \Phi_f^\alpha$ and $\Phi_3'^\omega \subseteq \Phi_f^\omega$ (since $\Phi_3'^\alpha \cup \varepsilon_0 \subseteq \Phi_3^\alpha$ and $\Phi_3'^\omega = \Phi_3^\omega$). We have $\Phi' \equiv [\Phi_1^\alpha \cup \varepsilon_0; \varepsilon_2' \cup \varepsilon_f; \Phi_3^\omega]$ and $(\varepsilon_2' \cup \varepsilon_f) \cup \varepsilon_0 \subseteq (\Phi_2^\varepsilon \cup \varepsilon_f)$. The choice of $\Phi'$ is acceptable since $\Phi'^\alpha = \Phi^\alpha \cup \varepsilon_0$, $\varepsilon' \cup \varepsilon_0 \subseteq \Phi^\varepsilon$ and $\Phi'^\omega = \Phi^\omega$ as required).

To prove 1., we have $n; \Gamma' \vdash H'$ by (ii), and we can apply [TApp]:

$$\text{(TAPP)}\frac{\begin{array}{c} \Phi_1'; \Gamma' \vdash v : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \cdot \qquad \Phi_2'; \Gamma' \vdash e_2' : \tau_1 \rightsquigarrow \mathcal{R}_2' \\ \Phi_1' \rhd \Phi_2' \rhd \Phi_3' \hookrightarrow \Phi' \\ \Phi_3'^\varepsilon = \Phi_f^\varepsilon \qquad \Phi_3'^\alpha \subseteq \Phi_f^\alpha \qquad \Phi_3'^\omega \subseteq \Phi_f^\omega \\ v \not\equiv v' \Rightarrow \mathcal{R}_2' = \cdot \end{array}}{\Phi'; \Gamma' \vdash v\ e_2' : \tau_2 \rightsquigarrow \cdot \bowtie \mathcal{R}_2'}$$

(Note that $\cdot \bowtie \mathcal{R}_2' = \mathcal{R}_2'$.)

The first premise follows by value typing and weakening; the second by (i); the third—sixth by choice of $\Phi'$, $\Phi_1'$, $\Phi_2'$, $\Phi_3'$; the last holds vacuously since $\mathcal{R}_1 \equiv \cdot$ by assumption.

To prove part 2., we must show that $\Phi', \mathcal{R}'; H' \vdash \Sigma'$. The proof is similar to the (TASSIGN)-[CONG] proof, case $r := \mathbb{E}$ but substituting $\varepsilon_f$ for $\varepsilon_r$.

Part 3. follows directly from (iv).

**case** (TSUB) **:**

We have

$$\text{(TSUB)}\frac{\begin{array}{c} \Phi''; \Gamma \vdash e : \tau'' \rightsquigarrow \mathcal{R} \\ \Phi'' \equiv [\alpha; \varepsilon''; \omega] \qquad \Phi \equiv [\alpha; \varepsilon; \omega] \\ \tau'' \leq \tau \qquad \varepsilon'' \subseteq \varepsilon \end{array}}{\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}}$$

since by flow effect weakening (Lemma B.7) we know that $\alpha$ and $\omega$ are unchanged in the use of (TSUB).

We have $\langle n; \Sigma; H; e\rangle \longrightarrow_\varepsilon \langle n; \Sigma'; H'; e'\rangle$. To apply induction we must show that $n; \Gamma \vdash H$, which we have by assumption, $\Phi''; \Gamma \vdash e : \tau'' \rightsquigarrow \mathcal{R}$, which we also have by assumption, and $\Phi'', \mathcal{R}; H \vdash \Sigma$, which follows easily since $\varepsilon'' \subseteq \varepsilon$.

Hence we have:

(i) $\Phi'''; \Gamma' \vdash e' : \tau'' \rightsquigarrow \mathcal{R}'$ and
(ii) $n; \Gamma' \vdash H'$
(iii) $\Phi''', \mathcal{R}'; H' \vdash \Sigma'$
(iv) $traceOK(\Sigma')$

for some $\Gamma' \supseteq \Gamma$, $\Phi'''$ such that $\Phi'''^\alpha = \alpha \cup \varepsilon_0$, $\Phi'''^\varepsilon \cup \varepsilon_0 \subseteq \varepsilon''$ Let $\Phi' \equiv \Phi'''$, and thus $\Phi'^\alpha = \alpha \cup \varepsilon_0$, $\Phi'^\varepsilon \cup \varepsilon_0 \subseteq \varepsilon$ since $\varepsilon'' \subseteq \varepsilon$, and $\Phi'^\omega = \omega$ as required. All results follow by induction.

□

**Lemma B.16** (Progress)**.** *If $n \vdash H, e : \tau$ (such that $\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}$ and $n; \Gamma \vdash H$) and for all $\Sigma$ such that $\Phi, \mathcal{R}; H \vdash \Sigma$ and $traceOK(\Sigma)$, then either e is a value, or there exist $n', H', \Sigma', e'$ such that $\langle n; \Sigma; H;\ e\rangle \longrightarrow_\eta \langle n'; \Sigma'; H';\ e'\rangle$.*

*Proof.* Induction on the typing derivation $n \vdash H, e : \tau$; consider each possible rule for the conclusion of this judgment:

**case** (TINT-TGVAR-TLOC) **:**

These are all values.

**case** (TVAR) **:**

Can't occur, since local values are substituted for.

**case** (TREF) **:**

We must have that

$$\text{(TREF)}\frac{\Phi; \Gamma \vdash e' : \tau \rightsquigarrow \mathcal{R}}{\Phi; \Gamma \vdash \mathsf{ref}\ e' : ref^{\,\varepsilon}\ \tau \rightsquigarrow \mathcal{R}}$$

There are two possible reductions, depending on the shape of $e$:

**case** $e' \equiv v$ **:**

By inversion on $\Phi; \Gamma \vdash v : \tau \rightsquigarrow \cdot$ we know that $\mathcal{R} \equiv \cdot$ hence by inversion on $\Phi, \mathcal{R}; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$. We have that $\langle n; (n', \sigma); H; \mathsf{ref}\ v\rangle \longrightarrow n; (n', \sigma); H'; r$ where $r \notin \mathrm{dom}(H)$ and $H' = H, r \mapsto (\cdot, v, \emptyset)$ by (REF).

**case** $e' \not\equiv v$ **:**

By induction, $\langle n; \Sigma; H; e'\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e''\rangle$ and thus $\langle n; \Sigma; H; (\mathsf{ref}\ \_)[e']\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; (\mathsf{ref}\ \_)[e'']\rangle$ by [CONG].

**case** (TDEREF) **:**

We know that

$$(\text{TDEREF})\ \frac{\begin{array}{c}\Phi_1; \Gamma \vdash e : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R} \\ \Phi_2^\varepsilon = \varepsilon_r \qquad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash !\, e : \tau \rightsquigarrow \mathcal{R}}$$

Consider the shape of $e$:

**case** $e' \equiv v$ **:**

Since $v$ is a value of type $\mathit{ref}^{\varepsilon_r}\ \tau$, we must have $v \equiv \mathsf{z}$ or $v \equiv r$.

**case** $e' \equiv \mathsf{z}$ **:**

We have

$$(\text{TDEREF})\ \frac{\begin{array}{c}\Phi_1; \Gamma \vdash \mathsf{z} : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \cdot \\ \Phi_2^\varepsilon = \varepsilon_r \qquad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash !\, \mathsf{z} : \tau \rightsquigarrow \cdot}$$

where by subtyping derivations (Lemma B.6) we have

$$(\text{TSUB})\ \frac{(\text{TGVAR})\ \dfrac{\Gamma(\mathsf{z}) = \mathit{ref}^{\varepsilon_r'}\ \tau'}{\Phi_\emptyset; \Gamma \vdash \mathsf{z} : \mathit{ref}^{\varepsilon_r'}\ \tau' \rightsquigarrow \cdot} \qquad \dfrac{\tau' \le \tau \qquad \tau \le \tau' \qquad \varepsilon_r' \subseteq \varepsilon_r}{\mathit{ref}^{\varepsilon_r'}\ \tau' \le \mathit{ref}^{\varepsilon_r}\ \tau} \qquad \Phi_\emptyset \le \Phi_1}{\Phi_1; \Gamma \vdash \mathsf{z} : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \cdot}$$

By inversion on $\Phi, \cdot; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$. By $n; \Gamma \vdash H$ we have $\mathsf{z} \in \mathrm{dom}(H)$ (and thus $H \equiv H'', \mathsf{z} \mapsto (\mathit{ref}^{\varepsilon_r'}\ \tau', v, \nu)$) since $\Gamma(\mathsf{z}) = \mathit{ref}^{\varepsilon_r'}\ \tau'$. Therefore, we can reduce via [GVAR-DEREF]:

$$\langle n; (n', \sigma); (H'', \mathsf{z} \mapsto (\mathit{ref}^{\varepsilon_r'}\ \tau', v, \nu));\ !\mathsf{z}\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); (H'', \mathsf{z} \mapsto (\mathit{ref}^{\varepsilon_r'}\ \tau', v, \nu)); v\rangle$$

**case** $e' \equiv r$ **:**

Similar to the $e' \equiv \mathsf{z}$ case above, but reduce using [DEREF].

**case** $e' \not\equiv v$ **:**

Let $\mathbb{E} \equiv\ !\_$ so that $e \equiv \mathbb{E}[e']$. To apply induction, we have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ by Lemma B.9. Thus we get $\langle n; \Sigma; H; e'\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e''\rangle$, hence we have that $\langle n; \Sigma; H; \mathbb{E}[e']\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; \mathbb{E}[e'']\rangle$ by [CONG].

**case** (TASSIGN) **:**

$$(\text{TASSIGN})\ \frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \mathit{ref}^{\varepsilon_r}\ \tau \rightsquigarrow \mathcal{R}_1 \qquad \Phi_2; \Gamma \vdash e_2 : \tau \rightsquigarrow \mathcal{R}_2 \\ \Phi_3^\varepsilon = \varepsilon_r \qquad\qquad \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ e_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi; \Gamma \vdash e_1 := e_2 : \tau \rightsquigarrow \mathcal{R}_1 \bowtie \mathcal{R}_2}$$

Depending on the shape of $e$, we have:

**case** $e_1 \equiv v_1, e_2 \equiv v_2$ **:**

Since $v_1$ is a value of type $\mathit{ref}^{\varepsilon_r}\ \tau$, we must have $v_1 \equiv \mathsf{z}$ or $v_1 \equiv r$. The results follow by reasoning quite similar to [TDEREF] above.

**case** $e_1 \equiv v_1, e_2 \not\equiv v$ **:**

Let $\mathbb{E} \equiv v_1 := \_$ so that $e \equiv \mathbb{E}[e_2]$. Since $e_1$ is a value, $\mathcal{R}_1 \equiv \cdot$ hence we have $\Phi_2, \mathcal{R}; H \vdash \Sigma$ by Lemma B.10 and we can apply induction. We have $\langle n; \Sigma; H; e_2\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e_2'\rangle$, and thus $\langle n; \Sigma; H; \mathbb{E}[e_2]\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; \mathbb{E}[e_2']\rangle$ by [CONG].

**case** $e_1 \not\equiv v$ **:**

Since $e_1$ is a not value, $\mathcal{R}_2 \equiv \cdot$ hence we have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ by Lemma B.10 and we can apply induction. The rest follows by an argument similar to the above case.

**case** (TUPDATE) **:**

By inversion on $\Phi; \Gamma \vdash \mathsf{update}^{\alpha,\omega} : \mathit{int} \rightsquigarrow \mathcal{R}$ we have that $\mathcal{R} \equiv \cdot$, hence by inversion on $\Phi, \cdot; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$. If $\mathit{updateOK}(\mathit{upd}, H, (\alpha, \omega), \mathit{dir}) = \mathtt{tt}$, then $\mathsf{update}^{\alpha,\omega}$ reduces via [UPDATE], otherwise $\mathsf{update}^{\alpha,\omega}$ reduces via [NO-UPDATE].

**case** (TIF) **:**

$$(\text{TIF})\ \frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \mathit{int} \rightsquigarrow \mathcal{R} \\ \Phi_2; \Gamma \vdash e_2 : \tau \rightsquigarrow \cdot \qquad \Phi_2; \Gamma \vdash e_3 : \tau \rightsquigarrow \cdot \\ \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi\end{array}}{\Phi; \Gamma \vdash \mathsf{if0}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau \rightsquigarrow \mathcal{R}}$$

Depending on the shape of $e$, we have:

**case** $e_1 \equiv v$ **:**

This implies $\mathcal{R} \equiv \cdot$ so by inversion on $\Phi, \cdot; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$. Since the type of $v$ is $\mathit{int}$, we know $v$ must be an integer $n$. Thus we can reduce via either [IF-T] or [IF-F].

**case** $e_1 \not\equiv v$ **:**

Let $\mathbb{E} \equiv \mathsf{if0}\ \_\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$ so that $e \equiv \mathbb{E}[e_1]$. To apply induction, we have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ by Lemma B.9. We have $\langle n; \Sigma; H; e_1\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e_1'\rangle$ and thus $\langle n; \Sigma; H; \mathbb{E}[e_1]\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; \mathbb{E}[e_1']\rangle$ by [CONG].

**case** (TTRANSACT) **:**

We know that:

$$(\text{TTRANSACT})\frac{\Phi'; \Gamma \vdash e : \tau \rightsquigarrow \cdot \quad \Phi^\alpha \subseteq \Phi'^\alpha \quad \Phi^\omega \subseteq \Phi'^\omega}{\Phi; \Gamma \vdash \mathsf{tx}\ e : \tau \rightsquigarrow \cdot}$$

By inversion on $\Phi, \cdot; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$. Thus we can reduce by [TX-START].

**case** (TINTRANS) **:**

We know that:

$$(\text{TINTRANS})\frac{\Phi'; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R} \quad \Phi^\alpha \subseteq \Phi'^\alpha \quad \Phi^\omega \subseteq \Phi'^\omega}{\Phi; \Gamma \vdash \mathsf{intx}\ e : \tau \rightsquigarrow \Phi', \mathcal{R}}$$

Consider the shape of $e$:

**case** $e \equiv v$ **:**

Thus

$$(\text{TINTRANS})\frac{\Phi'; \Gamma \vdash v : \tau \rightsquigarrow \cdot \quad \Phi^\alpha \subseteq \Phi'^\alpha \quad \Phi^\omega \subseteq \Phi'^\omega}{\Phi; \Gamma \vdash \mathsf{intx}\ v : \tau \rightsquigarrow \Phi', \cdot}$$

We have $\Phi, \Phi', \cdot; H \vdash \Sigma$ by assumption:

$$(\text{TC2})\frac{\begin{array}{c}\Phi', \cdot; H \vdash \Sigma \\ \Phi \equiv [\alpha; \varepsilon; \omega] \\ \mathsf{f} \in \sigma \Rightarrow \mathsf{f} \in \alpha \\ \mathsf{f} \in \varepsilon \Rightarrow n' \in ver(H, \mathsf{f})\end{array}}{\Phi, \Phi', \cdot; H \vdash ((n', \sigma'), (n'', \sigma''))}$$

By inversion we have $\Sigma \equiv ((n', \sigma'), (n'', \sigma''))$; by assumption we have $traceOK(n'', \sigma'')$ so we can reduce via [TX-END].

**case** $e \not\equiv v$ **:**

We have $\Phi, \Phi', \mathcal{R}; H \vdash \Sigma$ by assumption. By induction we have $\langle n; \Sigma'; H; e'\rangle \longrightarrow_\eta \langle n'; \Sigma''; H'; e''\rangle$, hence by [TX-CONG-2]:

$$\langle n; \Sigma'; H; \mathsf{intx}\ e'\rangle \longrightarrow_\emptyset \langle n'; \Sigma''; H'; \mathsf{intx}\ e''\rangle$$

**case** (TLET) **:**

We know that:

$$(\text{TLET})\frac{\Phi_1; \Gamma \vdash e_1 : \tau_1 \rightsquigarrow \mathcal{R} \quad \Phi_2; \Gamma, x : \tau_1 \vdash e_2 : \tau_2 \rightsquigarrow \cdot \quad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash \mathsf{let}\ x : \tau_1 = e_1\ \mathsf{in}\ e_2 : \tau_2 \rightsquigarrow \mathcal{R}}$$

Consider the shape of $e$:

**case** $e_1 \equiv v$ **:**

Thus $\Phi_1; \Gamma \vdash v : \tau \rightsquigarrow \cdot$ and by inversion on $\Phi, \cdot; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$.

We can reduce via [LET].

**case** $e_1 \not\equiv v$ **:**

Let $\mathbb{E} \equiv \mathsf{let}\ x : \tau_1 = \_\ \mathsf{in}\ e_2$ so that $e \equiv \mathbb{E}[e_1]$. To apply induction, we have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ by Lemma B.9. We have $\langle n; \Sigma; H; e_1\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e_1'\rangle$ and so $\langle n; \Sigma; H; \mathbb{E}[e_1]\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; \mathbb{E}[e_1']\rangle$ by [CONG].

**case** (TAPP) **:**

$$(\text{TAPP})\frac{\begin{array}{c}\Phi_1; \Gamma \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \mathcal{R}_1 \quad \Phi_2; \Gamma \vdash e_2 : \tau_1 \rightsquigarrow \mathcal{R}_2 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \Phi_3^\varepsilon = \Phi_f^\varepsilon \qquad \Phi_3^\alpha \subseteq \Phi_f^\alpha \qquad \Phi_3^\omega \subseteq \Phi_f^\omega \\ e_1 \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi; \Gamma \vdash e_1\ e_2 : \tau_2 \rightsquigarrow \mathcal{R}_1 \bowtie \mathcal{R}_2}$$

Depending on the shape of $e$, we have:

**case** $e_1 \equiv v_1, e_2 \equiv v_2$ **:**

Since $v_1$ is a value of type $\tau_1 \longrightarrow^\Phi \tau_2$, we must have $v_1 \equiv \mathsf{z}$, hence

$$(\text{TAPP})\frac{\begin{array}{c}\Phi_1; \Gamma \vdash \mathsf{z} : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \cdot \quad \Phi_2; \Gamma \vdash v : \tau_1 \rightsquigarrow \cdot \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \Phi_3^\varepsilon = \Phi_f^\varepsilon \qquad \Phi_3^\alpha \subseteq \Phi_f^\alpha \qquad \Phi_3^\omega \subseteq \Phi_f^\omega \\ \mathsf{z} \not\equiv v \Rightarrow \mathcal{R}_2 = \cdot\end{array}}{\Phi; \Gamma \vdash \mathsf{z}\ v : \tau_2 \rightsquigarrow \cdot}$$

where by subtyping derivations (Lemma B.6) we have

$$\text{(TSUB)}\frac{\text{(TGVAR)}\dfrac{\Gamma(\mathsf{z}) = \tau_1' \longrightarrow^{\Phi_f'} \tau_2'}{\Phi_\emptyset; \Gamma \vdash \mathsf{z} : \tau_1' \longrightarrow^{\Phi_f'} \tau_2' \rightsquigarrow \cdot} \qquad \dfrac{\tau_1 \leq \tau_1' \quad \tau_2' \leq \tau_2 \quad \Phi_f' \leq_f \Phi_f}{\tau_1' \longrightarrow^{\Phi_f'} \tau_2' \leq \tau_1 \longrightarrow^{\Phi_f} \tau_2} \quad \Phi_\emptyset \leq \Phi_1}{\Phi_1; \Gamma \vdash \mathsf{z} : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \rightsquigarrow \cdot}$$

By inversion on $\Phi, \cdot; H \vdash \Sigma$ we have $\Sigma \equiv (n', \sigma)$. By $n; \Gamma \vdash H$ we have $\mathsf{z} \in \mathrm{dom}(H)$ and $H \equiv (H'', \mathsf{z} \mapsto (\tau_1' \longrightarrow^{\Phi_f'} \tau_2', \lambda(x).e'', \nu))$ since $\Gamma(\mathsf{z}) = \tau_1' \longrightarrow^{\Phi_f'} \tau_2'$. By [CALL], we have:

$$\langle n; (n', \sigma); (H'', \mathsf{z} \mapsto (\tau_1' \longrightarrow^{\Phi_f'} \tau_2', \lambda(x).e'', \nu)); \mathsf{z}\ v\rangle \longrightarrow_{\{\mathsf{z}\}} \langle n; (n', \sigma \cup (\mathsf{z}, \nu)); (H'', \mathsf{z} \mapsto (\tau_1' \longrightarrow^{\Phi_f'} \tau_2', \lambda(x).e'', \nu)); e''[x \mapsto v]\rangle$$

**case** $e_1 \not\equiv v$ **:**
Let $\mathbb{E} \equiv \_\ e_2$ so that $e \equiv \mathbb{E}[e_1]$. Since $e_1$ is a not value, $\mathcal{R}_2 \equiv \cdot$ hence we have $\Phi_1, \mathcal{R}; H \vdash \Sigma$ by Lemma B.10 and we can apply induction and we have: $\langle n; \Sigma; H; e_1\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e_1'\rangle$, and thus $\langle n; \Sigma; H; \mathbb{E}[e_1]\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; \mathbb{E}[e_1']\rangle$ by [CONG].

**case** $e_1 \equiv v_1, e_2 \not\equiv v$ **:**
Let $\mathbb{E} \equiv v_1\ \_$ so that $e \equiv \mathbb{E}[e_2]$. Since $e_1$ is a value, $\mathcal{R}_1 \equiv \cdot$ hence we have $\Phi_2, \mathcal{R}; H \vdash \Sigma$ by Lemma B.10 and we can apply induction. The rest follows similarly to the above case.

**case** (TSUB) **:**
We know that:

$$\text{(TSUB)}\frac{\Phi_1; \Gamma \vdash e : \tau' \rightsquigarrow \mathcal{R} \qquad \tau' \leq \tau \qquad \Phi_1 \equiv [\alpha; \varepsilon_1; \omega] \qquad \Phi \equiv [\alpha; \varepsilon; \omega] \qquad \varepsilon_1 \subseteq \varepsilon}{\Phi; \Gamma \vdash e : \tau \rightsquigarrow \mathcal{R}}$$

If $e$ is a value $v$ we are done. Otherwise, since $\Phi_1, \mathcal{R}; H \vdash \Sigma$ follows from $\Phi, \mathcal{R}; H \vdash \Sigma$ (by $\Phi_1^\varepsilon \subseteq \Phi^\varepsilon$ and $\Phi_1^\alpha = \Phi^\alpha$); we have $\langle n; \Sigma; H; e\rangle \longrightarrow_\eta \langle n'; \Sigma'; H'; e'\rangle$ by induction.

□

**Lemma B.17** (Substitution)**.**
*If* $\Phi; \Gamma, x : \tau' \vdash e : \tau$ *and* $\Phi; \Gamma \vdash v : \tau'$ *then* $\Phi; \Gamma \vdash e[x \mapsto v] : \tau$.

*Proof.* Induction on the typing derivation of $\Phi; \Gamma \vdash e : \tau$.

**case** (TINT) **:**
Since $e \equiv n$ and $n[x \mapsto v] \equiv n$, the result follows by (TINT).

**case** (TVAR) **:**
$e$ is a variable $y$. We have two cases:

**case** $y = x$ **:**
We have $\tau = \tau'$ and $y[x \mapsto v] \equiv v$, hence we need to prove that $\Phi; \Gamma \vdash v : \tau$ which is true by assumption.

**case** $y \neq x$ **:**
We have $y[x \mapsto v] \equiv y$ and need to prove that $\Phi; \Gamma \vdash y : \tau$. By assumption, $\Phi; \Gamma, x : \tau' \vdash y : \tau$, and thus $(\Gamma, x : \tau')(y) = \tau$; but since $x \neq y$ this implies $\Gamma(y) = \tau$ and we have to prove $\Phi; \Gamma \vdash y : \tau$ which follows by (TVAR).

**case** (TGVAR),(TLOC), (TUPDATE) **:**
Similar to (TINT).

**case** (TREF) **:**
We know that $\Phi; \Gamma, x : \tau' \vdash \mathsf{ref}\ e : \mathit{ref}^\varepsilon\ \tau$ and $\Phi; \Gamma \vdash v : \tau'$, and need to prove that $\Phi; \Gamma \vdash (\mathsf{ref}\ e)[x \mapsto v] : \mathit{ref}^\varepsilon\ \tau$. By inversion on $\Phi; \Gamma, x : \tau' \vdash \mathsf{ref}\ e : \mathit{ref}^\varepsilon\ \tau$ we have $\Phi; \Gamma, x : \tau' \vdash e : \tau$; applying induction to this, we have $\Phi; \Gamma \vdash e[x \mapsto v] : \tau$. We can now apply [TRef]:

$$\text{(TREF)}\frac{\Phi; \Gamma \vdash e[x \mapsto v] : \tau}{\Phi; \Gamma \vdash \mathsf{ref}\ (e[x \mapsto v]) : \mathit{ref}^\varepsilon\ \tau}$$

The desired result follows since $\mathsf{ref}\ (e[x \mapsto v]) \equiv (\mathsf{ref}\ e)[x \mapsto v]$.

**case** (TDEREF) **:**
We know that $\Phi; \Gamma, x : \tau' \vdash !\, e : \tau$ and $\Phi; \Gamma \vdash v : \tau'$ and need to prove that $\Phi; \Gamma \vdash (!\, e)[x \mapsto v] : \tau$. By inversion on $\Phi; \Gamma, x : \tau' \vdash !\, e : \tau$ we have $\Phi_1; \Gamma, x : \tau' \vdash e : \mathit{ref}^{\varepsilon_r}\ \tau$ and $\Phi_2$ such that $\Phi_1 \rhd \Phi_2 \hookrightarrow \Phi$ and $\Phi \equiv \Phi_1 \rhd \Phi_2$. By value typing we have $\Phi_1; \Gamma \vdash v : \tau'$. We can then apply induction, yielding $\Phi_1; \Gamma \vdash e[x \mapsto v] : \mathit{ref}^{\varepsilon_r}\ \tau$. Finally, we apply (TDEREF)

$$\text{(TDEREF)}\frac{\Phi_1; \Gamma \vdash e[x \mapsto v] : \mathit{ref}^{\varepsilon_r}\ \tau \qquad \Phi_2^\varepsilon = \varepsilon_r \qquad \Phi_1 \rhd \Phi_2 \hookrightarrow \Phi}{\Phi; \Gamma \vdash !\, e[x \mapsto v] : \tau}$$

Note that the second premise holds by inversion on $\Phi; \Gamma, x : \tau' \vdash !\, e : \tau$. The desired result follows since $!\, (e[x \mapsto v]) \equiv (!\, e)[x \mapsto v]$.

**case** (TSUB) **:**
We know that $\Phi; \Gamma, x : \tau' \vdash e : \tau$ and $\Phi; \Gamma \vdash v : \tau'$ and need to prove that $\Phi; \Gamma \vdash e[x \mapsto v] : \tau$. By inversion on $\Phi; \Gamma, x : \tau' \vdash e : \tau$ we have $\Phi'; \Gamma, x : \tau' \vdash e : \tau'$. By value typing we have $\Phi'; \Gamma, x : \tau' \vdash v : \tau'$. We can then apply induction, yielding $\Phi'; \Gamma \vdash e[x \mapsto v] : \tau'$.

Finally, we apply (TSUB)

$$(\text{TSUB})\frac{\Phi';\Gamma \vdash e[x \mapsto v] : \tau' \quad \tau' \leq \tau \quad \Phi' \leq \Phi}{\Phi;\Gamma \vdash e[x \mapsto v] : \tau}$$

and get the desired result.

**case** (TTRANSACT),(TINTRANS) **:**

Similar to (TSUB).

**case** (TAPP) **:**

We know that

$$(\text{TAPP})\frac{\begin{array}{c}\Phi_1;\Gamma, x : \tau' \vdash e_1 : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \quad \Phi_2;\Gamma, x : \tau' \vdash e_2 : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \Phi_3^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3^{\omega} \subseteq \Phi_f^{\omega}\end{array}}{\Phi;\Gamma, x : \tau' \vdash e_1\ e_2 : \tau_2}$$

where $\Phi;\Gamma \vdash v : \tau'$, and need to prove that $\Phi;\Gamma \vdash (e_1\ e_2)[x \mapsto v] : \tau_2$. Call the first two premises above (1) and (2), and note that we have (3) $\Phi;\Gamma \vdash v : \tau' \Rightarrow \Phi_1;\Gamma \vdash v : \tau'$ and (4) $\Phi;\Gamma \vdash v : \tau' \Rightarrow \Phi_2;\Gamma \vdash v : \tau'$ by the value typing lemma. By (1), (3) and induction we have $\Phi_1;\Gamma \vdash e_1[x \mapsto v] : \tau_1 \longrightarrow^{\Phi_f} \tau_2$. Similarly, by (2), (4) and induction we have $\Phi_2;\Gamma \vdash e_2[x \mapsto v] : \tau_1$. We can now apply (TAPP):

$$(\text{TAPP})\frac{\begin{array}{c}\Phi_1;\Gamma \vdash e_1[x \mapsto v] : \tau_1 \longrightarrow^{\Phi_f} \tau_2 \quad \Phi_2;\Gamma \vdash e_2[x \mapsto v] : \tau_1 \\ \Phi_1 \rhd \Phi_2 \rhd \Phi_3 \hookrightarrow \Phi \\ \Phi_3^{\varepsilon} = \Phi_f^{\varepsilon} \qquad \Phi_3^{\alpha} \subseteq \Phi_f^{\alpha} \qquad \Phi_3^{\omega} \subseteq \Phi_f^{\omega}\end{array}}{\Phi;\Gamma \vdash e_1[x \mapsto v]\ e_2[x \mapsto v] : \tau_2}$$

Since $e_1[x \mapsto v]\ e_2[x \mapsto v] \equiv (e_1\ e_2)[x \mapsto v]$ we get the desired result.

**case** (TASSIGN-TIF-TLET) **:**

Similar to (TAPP).

□

**Theorem B.18** (Single-step Soundness)**.** *If $\Phi;\Gamma \vdash e : \tau$ where $[\![\Phi;\Gamma \vdash e : \tau]\!] = \mathcal{R}$; and $n;\Gamma \vdash H$; and $\Phi, \mathcal{R}; H \vdash \Sigma$; and $traceOK(\Sigma)$, then either $e$ is a value, or there exist $n'$, $H'$, $\Sigma'$, $\Phi'$, $e'$, and $\eta$ such that $\langle n;\Sigma;H;e\rangle \longrightarrow_\eta \langle n';\Sigma';H';e'\rangle$ and $\Phi';\Gamma' \vdash e' : \tau$ where $[\![\Phi';\Gamma' \vdash e' : \tau]\!] = \mathcal{R}'$; and $n';\Gamma' \vdash H'$; and $\Phi', \mathcal{R}'; H' \vdash \Sigma'$; and $traceOK(\Sigma')$ for some $\Phi', \Gamma', \mathcal{R}'$.*

*Proof.* From progress (Lemma B.16), we know that if $n \vdash H, e : \tau$ then either $e$ is a value, or there exist $n', H', \Sigma', \Phi', e', \eta$ such that $\langle n;\Sigma;H;e\rangle \longrightarrow_\eta \langle n';\Sigma';H';e'\rangle$. If $e$ is a value we are done. If $e$ is not a value, then there are two cases. If $\eta = \mu$ then the result follows from update preservation (Lemma B.13). If $\eta = \varepsilon_0$, then the result follows from preservation (Lemma B.15). □