A Monadic Analysis of Information Flow Security with Mutable State

Karl Crary   Aleksey Kliger   Frank Pfenning

July 2003

CMU-CS-03-164

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We explore the logical underpinnings of higher-order, security-typed languages with mutable state. Our analysis is based on a logic of information flow derived from lax logic and the monadic metalanguage. Thus, our logic deals with mutation explicitly, with impurity reflected in the types, in contrast to most higher-order security-typed languages, which deal with mutation implicitly via side-effects.

More importantly, we also take a *store-oriented* view of security, wherein security levels are associated with regions of the mutable store. In contrast, most other accounts are *value-oriented,* in that security levels are associated with individual values. Our store-oriented viewpoint allows us to address information flow security while still using a largely conventional logic, but we show that it does not lessen the expressive power of the logic. An interesting feature of our analysis lies in its treatment of upcalls (low-security computations that include high-security ones), employing an "informativeness" judgment indicating under what circumstances a type carries useful information.

This material is based on work supported in part by NSF grants CCR-9984812 and CCR-0121633. Any opinions, findings, and conclusions or recommendations in this publication are those of the authors and do not reflect the views of this agency.

Keywords: Type theory; language-based security; non-interference; mutable state; monads

Contents

1 Introduction

Security-typed languages use a type system to track the flow of information within a program to provide properties such as secrecy and integrity. Secrecy states that high-security information does not flow to low-security agents, and integrity dually states that low-security agents cannot corrupt high-security information. In this paper, we will restrict our attention to secrecy properties. A variety of security-typed languages have been proposed, and several of them are both higher-order (*i.e.,* support first-class functions) and provide mutable state [3, 8, 10, 12, 19].

In this paper, we explore the logical underpinnings of higher-order, security-typed languages with mutable state. Although most such languages rely on side-effects to manage mutable state, a logical analysis requires us to make store effects explicit. Thus, an appropriate programming language is Moggi's monadic metalanguage [6, 7], and the corresponding logic (via a Curry-Howard isomoprhism) is *lax logic* [2].

Our presentation of lax logic is based on that of Pfenning and Davies [9]. The principal distinctive feature of Pfenning and Davies's account is a syntactic distinction between *terms* and *expressions,* where terms are pure and expressions are (possibly) effectful. They show that this distinction allows the logic to possess some desirable properties (local soundness and local completeness) that state in essence that the logic's presentation is canonical. Although these properties are not particularly important here, the distinction also provides a clean separation between the pure and effectful parts of our analysis, which greatly simplifies our system.

A basic but important novelty of our account lies in the way we structure the security discipline. Most security-typed languages associate security levels with values, thus producing a situation in which some values are better than others. Although one could doubtless build a logic based on this structure (perhaps building on Abadi *et al.* [1]), it would certainly differ from the conventional logic in which one value is as good as another.

Instead, we adopt a structure in which security is associated with the mutable store and with operations on that store. Not only does this provide a more conventional logic, it also meshes nicely with our focus on effects. A natural question is whether this store-oriented security discipline limits the expressive power of our account relative to ones based on a value-oriented discipline, but we show (in Section 6) that it does not.

Overview The static semantics of our analysis is based on two typing judgments, one for terms (M) and one for expressions (E). Recall that terms are pure and that security is associated with effects, so the typing judgment for terms makes no mention of security levels. Thus, the typing judgment takes the form $\Sigma; \Gamma \vdash M : A$ (where Γ is the usual context and Σ assigns a type to the store).

Expressions, on the other hand, may have effects and therefore may interact with the security discipline. Each location in the store has a security level associated with it indicating the least security level that is authorized to read that location. Thus, the typing judgment for expressions tracks the security levels of all locations an expression reads or writes. Only the reads are of direct importance to the security discipline (recall that we do not address integrity), but writes must also be tracked since they provide a means of information flow. The judgment takes the form:

$$\Sigma; \Gamma \vdash E \div_{(r,w)} A$$

indicating that r is a upper bound to the levels of E's reads, and w is a *lower* bound to the levels of its writes, and also that E has type A. Naturally we require that $r \sqsubseteq w$, or else E could manifestly be leaking information.

In lax logic, expressions are internalized as terms using the monadic type $\bigcirc A$. A term of type $\bigcirc A$ is a suspended expression of type A. Thus, the introduction form for the monadic type is a term construct, and the elimination form (which releases the suspended expression) is an expression construct. Similarly, our expressions are internalized as terms using a monadic type written $\bigcirc_{(r,w)} A$. Since the effects of the suspended expression will be released when the monad is eliminated, the levels of those effects must be recorded in the monad type.

Most of the rules in our account follow from the intuitions above. One remaining novelty deals with the information content of types. Ordinarily, an expression would be deemed to be leaking information if it were to read from a high-security location, use the result of the read to form a value, and pass that value to a low-security computation. However, that expression would *not* be leaking information if one could show that the type of that value contained no information, or contained information usable only by a high-security computation (who could have performed the read anyway). Thus the type system contains a judgment $\vdash A \nearrow a$ stating that the type A contains information only for computations at the level a at least. This notion of *informativeness* is essential to accounting for the key issue of upcalls (low-security computations that include high-security computations).

The remainder of this paper is organized as follows: In Section 2 we present our basic logical account, including static and dynamic semantics, but omitting the key issue of upcalls. In Section 3 we extend our account to deal with upcalls. In Section 5 we state and prove a non-interference theorem. In Section 6 we show that our store-oriented account provides at least the expressive power of value-oriented accounts by embedding several previous approaches into our language. Section 7 discusses some related work, Section 8 offers some concluding remarks.

$$
\begin{array}{rcllll}
A,B,C & \in & \textit{types} & ::= & \ 1 \mid \mathsf{bool} \mid A \to B & \\
 & & & & \mid \mathsf{ref}_a\, A \mid \mathsf{refr}_a\, A \mid \mathsf{refw}_a\, A & \\
 & & & & \mid \bigcirc_o\, A & \\
M,N & \in & \textit{terms} & ::= & \ x & \text{variables} \\
 & & & & \mid * & \text{unit} \\
 & & & & \mid \mathsf{true} \mid \mathsf{false} & \text{boolean values} \\
 & & & & \mid \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 & \text{conditional} \\
 & & & & \mid \lambda x : A.M & \text{abstraction} \\
 & & & & \mid MN & \text{application} \\
 & & & & \mid \ell & \text{store location} \\
 & & & & \mid \mathsf{val}\ E & \text{suspended computation} \\
E,F & \in & \textit{expressions} & ::= & \ [M] & \text{return} \\
 & & & & \mid \mathsf{let\ val}\ x = M\ \mathsf{in}\ E & \text{sequencing} \\
 & & & & \mid \mathsf{ref}_a\,(M : A) & \text{store allocation} \\
 & & & & \mid\, !M & \text{store read} \\
 & & & & \mid M := N & \text{store write} \\
 & & & & & \\
\Gamma & \in & \textit{contexts} & ::= & \cdot \mid \Gamma, x : A & \\
\Sigma & \in & \textit{store types} & ::= & \{\} \mid \Sigma\{\ell : A\} & \\
 & & & & & \\
V & \in & \textit{values} & ::= & \ * \mid \mathsf{true} \mid \mathsf{false} & \\
 & & & & \mid \lambda x : A.M \mid \ell \mid \mathsf{val}\ E & \\
H & \in & \textit{stores} & ::= & \{\} \mid H\{\ell \mapsto V\} & \\
S & \in & \textit{computation states} & ::= & (H, \Sigma, E) &
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{let}\ x = E\ \mathsf{in}\ F & \equiv & \mathsf{let\ val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F \\
\mathsf{run}\ M & \equiv & \mathsf{let\ val}\ x = M\ \mathsf{in}\ [x]
\end{array}
$$

Figure 1: Syntax

2 Secure Monadic Calculus

We now describe the syntax, typing rules and operational semantics of our language.

2.1 Syntax

As in other work on information flow, we have in mind an arbitrary fixed lattice (that is, a partial order $(\mathcal{L}, \sqsubseteq)$ equipped with a join $\sqcup$, meet $\sqcap$, and least $\perp$ and greatest $\top$ elements) of security levels. We use the meta-variables a, b, c, r, w, ζ to range over elements of $\mathcal{L}$.

The full syntax of our language is given in Figure 1. The language is split into two syntactic categories: terms M and the expressions E, following Pfenning and Davies [9] . The terms are pure and evaluated to values V, while the expressions are executed for effect (but also return a value).

Operation levels To track the flow of information, we classify expressions not only by the value that they return, but also by the security levels of their effects. In particular, we keep track of an *operation level* $o = (r, w)$, for each expression. The security level r is an upper bound on the security levels of the store locations that the expression reads, while w is a lower bound on the security level of the store locations to which it writes.

Since expressions that write at a security level below their read level are obviously insecure, we restrict the operation levels to be elements of the set $\mathcal{O}$:

$$\mathcal{O} = \{(r, w) \in \mathcal{L} \times \mathcal{L} \mid r \sqsubseteq w\}$$

Henceforth, when we write an operation level (r, w), we will implicitly assume that it is an element of $\mathcal{O}$.

The operation levels have a natural ordering $(r, w) \preceq (r', w')$. Given some expression E, if it reads from level at most r, then it surely reads from level at most r', provided that $r \sqsubseteq r'$. Similarly, if it writes at level at least w, then it writes at level at least w', provided that $w' \sqsubseteq w$. That is, operation levels are covariant in the reads and contravariant in the writes:

$$(r, w) \preceq (r', w') \text{ iff } (r \sqsubseteq r' \text{ and } w' \sqsubseteq w)$$

There is a subsumption principle for operation levels: if expression E has operation level o, and $o \preceq o'$, then E has operation level o'.

Terms At the term level, we have variables, unit, booleans and conditional terms, function abstractions and applications. In support of our operational semantics, store locations are also terms. With each location ℓ is associated a fixed security level $\mathsf{Level}(\ell)$. The store associates locations with the values they contain. A subtyping relation (explored later in this paper), allows us to treat store cells as either read-write, read-only, or write-only.

The term $\mathsf{val}\ E$ allows expressions to be included at the term level as an element of the monadic type $\bigcirc_o A$. Since terms are pure, a $\mathsf{val}\ E$ does not execute the expression E, but rather represents a suspended computation.

Expressions The expressions include a trivial return expression $[M]$. The return expression has no effect, and simply returns the value to which M evaluates. In general, when an expression has no read effects, we say its read level is $\bot$, and if an expression has no write effects, we say its write level is $\top$. Accordingly, the operation level of $[M]$ is $(\bot, \top)$. Note that $(\bot, \top)$ is the least element in the $\preceq$ ordering, so our subsumption principle will let us weaken the operation level of $[M]$ to any operation level.

The sequencing expression $\mathsf{let}\ \mathsf{val}\ x = M\ \mathsf{in}\ F$ evaluates M down to some $\mathsf{val}\ E$, and executes E followed by F. The return value of expression E is bound to the variable x in F. If E and F both have operation level o, then so does the sequencing expression.

We will often write $\mathsf{let}\ x = E\ \mathsf{in}\ F$ as syntactic sugar for $\mathsf{let}\ \mathsf{val}\ x = \mathsf{val}\, E\ \mathsf{in}\ F$, and $\mathsf{run}\ M$ for $\mathsf{let}\ \mathsf{val}\ y = M\ \mathsf{in}\ [y]$. The derived typing rules are given in Appendix A.2.

In addition, there are expressions that allocate, read from, and write to the store. A read expression $!M$ has operation level $(a, \top)$, where a is the security level of the store location being read, and returns the contents of the store location. Dually, a write expression $M := N$ has operation level $(\bot, a)$ and updates the store location with the value of N; it does not return an interesting value (*i.e.,* it returns unit).

Store allocation $\mathsf{ref}_a\,(M : A)$ specifies the security level a and type A of the new store location.

Allocation cannot leak information. Evidently, it is not a read operation. Less obviously, it is not a write operation either. With a write, another expression may learn something about the current computation by observing a change in the value stored at a particular store location. However, the key to this scenario is that the same location is mentioned by more than one expression. On the other hand, allocation creates a new location that is mentioned nowhere else. Thus, there can be no implicit flow of information via an allocation expression. As a result, allocation has operation level $(\bot, \top)$.

States A computation state is a partially executed program, and consists of a triple (H, Σ, E) of a store H, a store type Σ and a closed expression E. The store maps locations to values, and the store type maps locations to the types of those values.

We assume that in a state (H, Σ, E), the store binds occurrences of store locations ℓ in H and E, and we identify computation states up to renaming of store locations. In addition, as usual, we identify all constructs up to renaming of bound variables.

2.2 Static Semantics

The type system of our language consists of two main mutually recursive judgments for typing terms and expressions, and some judgments for typechecking stores, and computation states that are summarized in Table 1. The first judgment

$$\Sigma; \Gamma \vdash M : A$$

says that the term M has type A in the context Γ, where the store has type Σ. The second judgment typechecks expressions

$$\Sigma; \Gamma \vdash E \div_o A$$

says that the expression E returns a value of type A and performs only operations within level o, as discussed above. Each rule is given with its rule number, and the full set of rules appears in Appendix A.2.

We assume that contexts Γ are well-formed, that is, they contain at most one occurrence of each variable x. We tacitly rename bound variables prior to

Table 1: Typing judgments

Judgment	**Meaning**
$\Sigma; \Gamma \vdash M : A$	Term M has type A
$\Sigma; \Gamma \vdash E \div_o A$	Expression E has type A and operation level o
$\vdash A \leq B$	Type A is a subtype of B
$\vdash H : \Sigma$	Store H has type Σ
$\vdash S \div_o A$	Computation state S is well-typed

adding them to a context to maintain well-formedness. Similarly, we assume that store types are well-formed, that is, they contain at most one occurrence of each store location ℓ.

Terms The typing rules for terms are unsurprising for a simply-typed lambda calculus with unit, abstraction and applications. A store location ℓ (provided that it is in dom(Σ)) has a ref-type with its security level:

$$\frac{}{\Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\ \Sigma(\ell)} \quad (24)$$

A computation term val E has the type $\bigcirc_o A$, provided the expression E has type A and operation level o:

$$\frac{\Sigma; \Gamma \vdash E \div_o A}{\Sigma; \Gamma \vdash \mathsf{val}\ E : \bigcirc_o A} \quad (27)$$

Expressions The typing rules for expressions follow our informal description. Trivial computations have the type of their return value, and operation level $(\bot, \top)$:

$$\frac{\Sigma; \Gamma \vdash M : A}{\Sigma; \Gamma \vdash [M] \div_{(\bot,\top)} A} \quad (29)$$

The sequencing expression typechecks provided both of the sub-computations have the same operation level (which may require using the weakening rule for operation levels):

$$\frac{\Sigma; \Gamma \vdash M : \bigcirc_o A \quad \Sigma; \Gamma, x : A \vdash E \div_o A}{\Sigma; \Gamma \vdash \mathsf{let\ val}\ x = M\ \mathsf{in}\ E \div_o A} \quad (30)$$

Allocation returns a new read/write store location:

$$\frac{\Sigma; \Gamma \vdash M : A}{\Sigma; \Gamma \vdash \mathsf{ref}_a\ (M : A) \div_{(\bot,\top)} \mathsf{ref}_a\ A} \quad (31)$$

For read and write expressions we only require that the corresponding store location is readable or writable, respectively:

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{refr}_a\ A}{\Sigma; \Gamma \vdash !M \div_{(a,\top)} A} \quad (32)$$

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{refw}_a\, A \quad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash M := N \div_{(\bot, a)} 1} \; (33)$$

In general we may weaken the operation level of a computation (indeed, as noted above, this is often necessary for the letval typing rule to apply):

$$\frac{\Sigma; \Gamma \vdash E \div_o A \quad o \preceq o'}{\Sigma; \Gamma \vdash E \div_{o'} A} \; (34)$$

Subtyping A subsumption rule allows us to weaken the type A of a term M or an expression E, provided A is a subtype of B:

$$\frac{\Sigma; \Gamma \vdash M : A \quad \vdash A \leq B}{\Sigma; \Gamma \vdash M : B} \; (28)$$

$$\frac{\Sigma; \Gamma \vdash E \div_o A \quad \vdash A \leq B}{\Sigma; \Gamma \vdash E \div_o A} \; (36)$$

Read-only store cells are covariant in the type of their contents and in their security level, and dually write-only cells are contravariant in each:

$$\frac{\vdash A \leq B \quad a \sqsubseteq b}{\vdash \mathsf{refr}_a\, A \leq \mathsf{refr}_b\, B} \; (17) \qquad \frac{\vdash B \leq A \quad b \sqsubseteq a}{\vdash \mathsf{refw}_a\, A \leq \mathsf{refw}_b\, B} \; (18)$$

Read/write store cells are neither covariant nor contravariant, but may be weakened to read-only or write-only cells:

$$\frac{\vdash A \leq B \quad a \sqsubseteq b}{\vdash \mathsf{ref}_a\, A \leq \mathsf{refr}_b\, B} \; (15) \qquad \frac{\vdash B \leq A \quad b \sqsubseteq a}{\vdash \mathsf{ref}_a\, A \leq \mathsf{refw}_b\, B} \; (16)$$

Finally, the monadic type $\bigcirc_o A$ is covariant in the return value type and operation level

$$\frac{\vdash A \leq B \quad o \preceq o'}{\vdash \bigcirc_o A \leq \bigcirc_{o'} B} \; (14)$$

Stores and states A store H is well-typed with store type Σ, provided that each value V_i in the store is well typed under Σ and the empty context, where Σ has the same domain as H

$$\frac{\mathrm{dom}(\Sigma) = \{\ell_1, \ldots, \ell_n\} \quad \Sigma; \cdot \vdash V_i : \Sigma(\ell_i) \text{ for } 1 \leq i \leq n}{\vdash \{\ell_1 \mapsto V_1, \ldots \ell_n \mapsto V_n\} : \Sigma} \; (37)$$

(Note that since Σ appears on the left in the premise of the rule, it must be well-formed).

A computation state (H, Σ, E) is well-typed provided that the store and the expression are each well-typed with the same store type:

$$\frac{\vdash H : \Sigma \quad \Sigma; \cdot \vdash E \div_o A}{\vdash (H, \Sigma, E) \div_o A} \; (38)$$

2.3 Operational Semantics

A computation state is called *terminal* if it is of the form $(H, \Sigma, [V])$. An evaluation relation $S \rightarrow S'$ gives the small-step operational semantics for computation states. We write $S \downarrow$ if for some terminal state S', $S \rightarrow^* S'$. Since terms are pure and do not have an effect on the store, their evaluation rules may be given simply by the relation $M \rightarrow M'$ (no store is required). The entire set of evaluation rules is given in Appendix B.

We write $M[N/x]$ and $E[N/x]$ for the capture-avoiding substitution of N for x in the term M or expression E. We write $H\{\ell \mapsto V\}$ for finite map that extends H with V at ℓ.

It is instructive to consider how a computation in state $S_0 = (H, \Sigma, \mathsf{let\ val}\ x = M\ \mathsf{in}\ F)$ would evaluate. There are three stages:

1. LETVAL1 is repeatedly applied until M is evaluated down to a value $\mathsf{val}\ E$, $S_1 = (H, \Sigma, \mathsf{let\ val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F)$
2. LETVALVAL is then applied until the subcomputation (H, Σ, E) is evaluated to a terminal state $(H', \Sigma', [V])$, $S_2 = (H', \Sigma', \mathsf{let\ val}\ x = \mathsf{val}\ [V]\ \mathsf{in}\ F)$
3. LETVAL substitutes the value V for x in F and computation continues in state $S_2 = (H', \Sigma', F[V/x])$.

For the proof of non-interference (specifically for the proof of the Hexagon Lemma), it will be useful to have the following lemma. It says that if a term evaluates to a value (or if a computation state evaluates to a terminal state) then the syntactic subterms (or subexpressions) of the given term (or computation state) will likewise evaluate to values (or terminal states). That is, our account is call-by-value.

Lemma 2.1 (Subterm/Subexpression Termination).

- *If* $(H, \Sigma, E) \downarrow$ *in* n *steps, then*
  1. *if* $E = [M]$ *then* $M \rightarrow^n V$
  2. *if* $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$ *then* $M \rightarrow^k \mathsf{val}\ E'$,
  $$(H, \Sigma, E') \downarrow \text{ in } m \text{ steps}$$
  and $k + m < n$
  3. *if* $E = \mathsf{ref}_a\,(M : A)$ *then* $M \rightarrow^k V$ *and* $k < n$
  4. *if* $E = !M$ *then* $M \rightarrow^k V$ *and* $k < n$
  5. *if* $E = M := N$ *then* $M \rightarrow^k V_1$, $N \rightarrow^m V_2$ *and* $k + m < n$
- *If* $M \rightarrow^n V$ *then*
  1. *If* $M = N_1 N_2$, *then* $N_1 \rightarrow^k V_1$ *and* $V_1 N_2 \rightarrow^m V_1 V_2$ *and* $k + m < n$
  2. *If* $M = \mathsf{if}\ N_1\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$ *then* $N_1 \rightarrow^k V_1$ *and* $k < n$

Proof by induction on the number of steps in the evaluation relation, by cases on the last rule. The details are given in Appendix C.

Our operational semantics are deterministic. Of course computation states are only deterministic up to renaming of store locations: recall that we consider store locations to be bound by the store in a computation state. We allow a bound store location ℓ to be renamed ℓ', as long as $\mathsf{Level}(\ell) = \mathsf{Level}(\ell')$. (Alternately, think of each security level as determining a collection of store locations; each bound store location may be renamed only to a location within the same collection.) Determinacy is used in the proof of non-interference.

Lemma 2.2 (Determinacy). *If* $M \to M_1$ *and* $M \to M_2$ *then* $M_1 = M_2$*. If* $S \to S_1$ *and* $S \to S_2$ *then* $S_1 = S_2$

Proof. by induction on the evaluation relations. By cases on $M \to M_1$ (or $S \to S_1$).

In each case, by the structure of M (resp., S), there is a single evaluation rule for $M \to M_2$ (resp., $S \to S_2$), then by IH. □

Since allocation extends the store, the following lemma shows that in any sequence of evaluation steps (of a not-necessarily well-typed state), the store type only grows. We use this fact in the HSS Lemma.

Lemma 2.3 (Store Size). *If* $(H, \Sigma, E) \to^* (H', \Sigma', E')$ *then* $\Sigma' \supseteq \Sigma$

Proof. Suffices to show for one step: if $(H, \Sigma, E) \to (H', \Sigma', E')$ then $\Sigma' \supseteq \Sigma$. The multi-step result follows because $\supseteq$ is reflexive and transitive. We proceed by induction on the evaluation derivation $(H, \Sigma, E) \to (H', \Sigma', E')$ Consider the last evaluation rule used:

- Case Ref: Evidently, $\Sigma' \supseteq \Sigma$.
- Case Letvalval: By IH
- In the remaining cases, the store type is unchanged.

□

3 Upcalls

Although the approach discussed so far is secure, it falls short of a practical language. There is no way to include a computation that reads from the high-security store in a larger low security computation. In any program with a high security read, the read level of the entire program is pushed up. However, many programs that contain *upcalls* to high security computations followed by low security code are secure.

Consider the program $\mathsf{let}\ z = P\ \mathsf{in}\ E$ where $P \div_{(\top,\top)} 1$ and E has operation level $(\bot, \bot)$. As we argued in the introduction, P does not leak information because 1 carries no information. Thus we would like to give the entire program the operation level $(\perp, \perp)$. However the type system we have presented so far would instead promote the operation level of E and the entire program to $(\top, \top)$.

In order to have a logic of information flow, we must offer an account of upcalls. Indeed, the power to perform high security computations interspersed in a larger low-security computation is the *sine qua non* of useful secure programming languages. We offer a detailed analysis of two cases where upcalls do not violate our intuitive notion of security. From these examples, we develop a general principle for treating upcalls. We take up the question of non-interference in Section 5.

3.1 An example with unit

Let E be some expression with type A and operation level (r, w) (recall that this implies that $r \sqsubseteq w$). In general, E may read values from store locations with security level below r, write values to store locations with security level at least w, and return some value of type A.

Suppose that $A = 1$. In that case, no matter what E does, if it terminates, it must return $*$. *The return value is not informative.*[1] Any other computation F that may gain information through the execution of E must be able to read store locations at security level at least w. But since $r \sqsubseteq w$, F could just directly read any store locations that E reads. On the other hand, any computation with operation level (r', w') where $w \not\sqsubseteq r'$ can neither observe E's effects nor gain any information from its (uninformative) return value.

As a result, in either case, we can say that E has an effective read level of $\perp$ just as if it had no reads:

$$\frac{\Sigma; \Gamma \vdash E \div_{(r,w)} 1}{\Sigma; \Gamma \vdash E \div_{(\perp,w)} 1} \ (*)$$

Note that the read level now refers only to informative reads, not all reads.

The new rule allows us to have some high-security computations prior to low security ones. Suppose $\Sigma; \cdot \vdash E \div_{(\top,\top)} 1$, and $\Sigma; x : 1 \vdash F \div_{(\perp,\perp)} A$ for some A. That is, E is a high-security computation, and F is a low-security one. With the new rule, the upcall to E, followed by the low-security computation F, can be type checked using the new rule $(*)$, E has operation level $(\perp, \top)$, which can be weakened to $(\perp, \perp)$ by rule (34), and thus:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Sigma; \cdot \vdash E \div_{(\perp,\perp)} 1}}{\Sigma; \cdot \vdash \mathsf{val}\ E : \bigcirc_{(\perp,\perp)} 1}\ (27) \qquad \Sigma; x : 1 \vdash F \div_{(\perp,\perp)} A}{\Sigma; \cdot \vdash \mathsf{let\ val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F \div_{(\perp,\perp)} A}\ (30)$$

[1] We are dealing here with weak non-interference: the knowledge that E terminated at all is deemed not to carry any information.

Note that the rule $(*)$ does not alter the write level of the expression (that is, the operation level in the conclusion is not $(\bot, \top)$). Such a rule would allow programs to leak information.

3.2 A more general example

Now consider a computation E with operation level (r, w), but this time, suppose that E has type $\mathsf{ref}_a\, B$ for some type B. Are there any situations where E may be given a different operation level?

Suppose that $r \sqsubseteq a$. In that case, any computation that may read the $\mathsf{ref}_a\, B$ is also able to read any store locations that E may read. Again, any computation can either do what E does itself, or it cannot gain information from E's return value.

On the other hand, consider the case where $r \not\sqsubseteq a$. The particular value of type $\mathsf{ref}_a\, B$ that E returns may carry information from store locations at security level r. For example, E may return one of two such store locations ℓ_1 or ℓ_2 from level a based on some boolean value V from a store location at security level r. In that case, a computation that reads at security level a may learn something about E's reads (at level r) by reading from E's return value. Since $r \not\sqsubseteq a$, this represents a violation of secure information flow.

So if E returns a $\mathsf{ref}_a\, B$, we can demote its reading level whenever $r \sqsubseteq a$, because any computation that wishes to make use of that return value would need a read level of at least r. In other words, a $\mathsf{ref}_a\, B$ is informative only to computations that may read at least at some security level (namely a) above r.

Thus, we may wish to add a new rule for $\mathsf{ref}_a\, B$:

$$\frac{\Sigma; \Gamma \vdash E \div_{(r,w)} \mathsf{ref}_a\, B \quad r \sqsubseteq a}{\Sigma; \Gamma \vdash E \div_{(\bot,w)} \mathsf{ref}_a\, B}\ (**)$$

However, instead we add a general rule that allows us to demote the reading level of an expression E:

$$\frac{\Sigma; \Gamma \vdash E \div_{(r,w)} A \quad \vdash A \nearrow r}{\Sigma; \Gamma \vdash E \div_{(\bot,w)} A}\ (35)$$

where the new judgment $\vdash A \nearrow r$ formalizes the idea that values of type A, if they are informative at all, are informative only at level r or above.[2]

In terms of our new notation, our earlier observations are that $\vdash 1 \nearrow r$ for any r, and $\vdash \mathsf{ref}_a\, A \nearrow r$ whenever $r \sqsubseteq a$.

3.3 Informativeness

We now consider some properties of the new judgment $\vdash A \nearrow a$. Several structural rules for the judgment are immediate. If A is any type at all, then

$$\frac{}{\vdash A \nearrow \bot}\ (1)$$

[2] Informativeness is closely related to *protectedness* in DCC [1]. We discuss the relationship in Section 7.

That is, if A is informative at all, then it's informative only at $\perp$ or above. In the interest of brevity, in the sequel we will say "informative only above a" to mean "informative only at a and above."

Also, if A is informative only above a and if $b \sqsubseteq a$, then A is informative only above b. That is, we may choose to discard some knowledge about when a type is informative

$$\frac{\vdash A \nearrow a \quad b \sqsubseteq a}{\vdash A \nearrow b} \; (10)$$

Finally, suppose A is informative only above a, and A is informative only above b. Then for any r if values of type A are informative to computations that read at r, we know that both $a \sqsubseteq r$ and $b \sqsubseteq r$. Therefore, for any such r, $a \sqcup b \sqsubseteq r$. So in fact, A is informative only above $a \sqcup b$:

$$\frac{\vdash A \nearrow a \quad \vdash A \nearrow b}{\vdash A \nearrow a \sqcup b} \; (11)$$

With the structural rules in place, we may consider each of the types in our language. We should keep in mind, that by adding rules to the judgment $\vdash A \nearrow a$ we increase the expressive power of the language by allowing more programs to be well-typed. It is always safe to add more restrictive rules in place of more liberal ones. Below we take the most permissive rules that still maintain non-interference, although it is not clear in all cases that there exist programs which need the added flexibility of certain rules.

A value of type bool is informative for any computation at all, since it may be trivially analyzed with a conditional. So aside from the structural axiom $\vdash A \nearrow \perp$, there should be no other rules for bool. We would give a similar account of other type constructors that may be analyzed by cases. For example sum types $A + B$ or integers int.

A value of type $A \to B$ is used by applying it to some value and using the result. So $A \to B$ is informative exactly when B is:

$$\frac{\vdash B \nearrow a}{\vdash A \to B \nearrow a} \; (3)$$

One straightforward rule for refs says that a ref is only informative if we can get at the value within it.

$$\frac{}{\vdash \mathsf{ref}_a\, A \nearrow a} \; (5)$$

However there is another rule for refs. Even if a computation can read from a store location of type $\mathsf{ref}_b\, A$ (*i.e.,* its read level is above b), only if A is informative at its operation level, can $\mathsf{ref}_b\, A$ be informative:

$$\frac{\vdash A \nearrow a}{\vdash \mathsf{ref}_b\, A \nearrow a} \; (6)$$

Read-only store locations are useful only to computations that may read from them. Consequently, by an argument similar to the one for read-write store cells, we have the two rules:

$$\frac{\vdash A \nearrow a}{\vdash \mathsf{refr}_b\, A \nearrow a}\ (7) \qquad \frac{}{\vdash \mathsf{refr}_b\, A \nearrow b}\ (8)$$

For write-only store cells $\mathsf{refw}_a\, A$, we have to consider aliasing. One way that a computation may learn whether two store locations are aliases is by writing a known value to one of them, and then reading out the value from the other. Because of subtyping, if a lower-security computation has a store location ℓ of type $\mathsf{refr}_a\, A$, a value of type $\mathsf{refw}_a\, A$ may be informative if the computation can read from (the seemingly unrelated) ℓ. As a result, we have the following rule:

$$\frac{}{\vdash \mathsf{refw}_a\, A \nearrow a}\ (9)$$

It is instructive to consider in detail the problem with write-only store locations $\mathsf{refw}_a\, A$. Suppose that instead of the rule (9), we had the following rule

$$\frac{}{\vdash \mathsf{refw}_a\, A \nearrow b}\ (\textit{incorrect})$$

That is, the same as the rule for unit: a value of type $\mathsf{refw}_a\, A$ is only informative above some security level b, for any b, *i.e.* not informative.

The following computation shows that with the incorrect rule, it is possible to leak high security information (whether the value of *secret*, a $\top$-security bool, is true) to a low security computation[3]:

$$\begin{array}{l}
\mathsf{let}\ x = \mathsf{ref}_\bot\,(\mathsf{false} : \mathsf{bool})\ \mathsf{in} \\
\mathsf{let}\ y = \mathsf{ref}_\bot\,(\mathsf{false} : \mathsf{bool})\ \mathsf{in} \\
\mathsf{let}\ z = (\mathsf{let}\ q = !\mathit{secret}\ \mathsf{in} \\
\qquad\quad [\mathsf{if}\ q\ \mathsf{then}\ x\ \mathsf{else}\ y])\ \mathsf{in} \\
\mathsf{let}\ _ = z := \mathsf{true}\ \mathsf{in} \\
\mathsf{run}\ !x
\end{array}$$

The program lets z alias either x or y depending on the value of secret. The computation whose value is assigned to z may be subsumed to type $\mathsf{refw}_\bot\, \mathsf{bool}$, and by the incorrect rule, $\vdash \mathsf{refw}_\bot\, \mathsf{bool} \nearrow \top$, so the operation level of that computation can be dropped to $(\bot, \top)$ (and subsumed to $(\bot, \bot)$). Then by writing a known value to z, whether we can observe a change in another alias of the same location is sufficient to learn about *secret*. We can give the entire computation the operation level $(\bot, \top)$ while it demonstrably returns the high-security value.

Finally, consider the type $\bigcirc_{(r,w)} A$. A value of this type is informative both to computations that may read at least security level w (that is, the level the suspended expression writes to), and to computations for which the type A is informative:

$$\frac{\vdash A \nearrow a}{\vdash \bigcirc_{(r,w)} A \nearrow w \sqcap a}\ (4)$$

$\lambda c : \bigcirc_{(\top,\top)}\mathsf{bool}.$
val
$\mathsf{let}\ wref = \mathsf{ref}_\top\ (\mathsf{val}\ [*] : \bigcirc_{(\bot,\top)}1)\ \mathsf{in}$
$\mathsf{let}\ w = [\mathsf{val}\ (\mathsf{let}\ b = \mathsf{run}\ c\ \mathsf{in}\ \mathsf{run}\ (\mathsf{if}\ b\ \mathsf{then}\ \mathsf{val}\ (\mathsf{let}\ w' = !wref\ \mathsf{in}\ \mathsf{run}\ w')\ \mathsf{else}\ \mathsf{val}\ [*]))]\ \mathsf{in}$
$\mathsf{let}\ \_ = wref := w\ \mathsf{in}$
$\mathsf{run}\ w$

Figure 2: $untilFalse : \bigcirc_{(\top,\top)}\mathsf{bool} \to \bigcirc_{(\bot,\top)}1$

With informativeness in hand, many more useful terms become well-typed. Consider, for example, the term in Figure 2. The function $untilFalse$ takes as argument a computation that reads and writes high before returning a boolean, and runs that computation repeatedly until it returns false. Recursion is accomplished using backpatching: a store location with a dummy value is allocated and is bound to $wref$, recursive calls in the body of the loop dereference $wref$ and run the contents. The recursive knot is tied by overwriting the contents of $wref$ with the real loop body w.

Interestingly, although $untilFalse$ takes a high-security computation as an argument, our type system is able to give it the type $\bigcirc_{(\top,\top)}\mathsf{bool} \to \bigcirc_{(\bot,\top)}1$, that is its return type is a low-security computation. Intuitively, even if f is a high-security computation, $untilFalse\ f$ does not leak any information to low-security since any information gained from f's return value is used only within the loop. To formally show that $untilFalse$ is well-typed, observe that $\Gamma \vdash \mathsf{let}\ b = \mathsf{run}\ c\ \mathsf{in}\ \mathsf{run}\ (\dots) \div_{(\top,\top)} 1$, and since $\vdash 1 \nearrow \top$, it can be given operation level $(\bot, \top)$. The rest of the typing derivation is straightforward.

4 Type Safety

Our language enjoys the usual type safety property: well-typed computations do not become stuck. We may show type safety in the usual manner — using Canonical Forms, Preservation and Progress lemmas.

Lemma 4.1 (Canonical Forms). *If $\Sigma; \cdot \vdash V : A$ and*

1. *if $A = 1$ then $V = *$*
2. *if $A = \mathsf{bool}$ then $V = \mathsf{true}$ or $V = \mathsf{false}$*
3. *if $A = B \to C$ then $V = \lambda x : B'.M$*
4. *if $A = \mathsf{ref}_a\, B$ then $V = \ell$ and $\ell \in \mathrm{dom}(\Sigma)$*
5. *if $A = \mathsf{refr}_a\, B$ then $V = \ell$ and $\ell \in \mathrm{dom}(\Sigma)$*

[3] Recall that $\mathsf{let}\ x = E\ \mathsf{in}\ F$ is syntactic sugar for $\mathsf{let}\ \mathsf{val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F$

6. *if* $A = \mathsf{refw}_a\, B$ *then* $V = \ell$ *and* $\ell \in \mathrm{dom}(\Sigma)$

7. *if* $A = \bigcirc_o B$ *then* $V = \mathsf{val}\ E$

Proof. by induction on the typing derivation; by inspection of the last typing rule used. □

Lemma 4.2 (Preservation). *If* $\vdash S \div_o A$ *and* $S \to S'$ *then* $\vdash S' \div_o A$

Proof by induction on the evaluation relation. Proof given in Appendix C.1.4.

Lemma 4.3 (Progress). *If* $\vdash S \div_o A$ *then either* S *is terminal, or* $\exists S'$ *such that* $S \to S'$

Proof by induction on the typing derivation. Proof given in Appendix C.1.4.

Theorem 4.4 (Type Safety). *If* $\vdash S$ *and* $S \to^* S'$ *then* S' *is not stuck.*

Proof. By induction on the number of evaluation steps. If S takes zero steps, then by Progress, it is not stuck. If S takes $n+1$ steps, then by Preservation it takes a step to some well-typed state, and so by the induction hypothesis, S' is not stuck. □

5 Non-interference

Informally, non-interference says that computations that have a low read level do not depend on values in high security store locations. As in similar arguments [19, 18], "low" means below some fixed security level ζ, and "high" means not below ζ.

Operationally, the low security sub-computations of a program should behave identically irrespective of the values in the high security store locations. On the other hand, it is okay for high security sub-computations to behave differently depending on values in high security store locations. However once a high security sub-computation completes, the low security behavior should again be identical modulo the parts of the computation state that are "out of view" of the low security part of the program.

Formally, we define an equivalence property of computation states such that two states are equivalent whenever they agree on the "in view" parts of the computation state. Then, in the style of a confluence proof modulo an equivalence relation [5], we show that this property is preserved under evaluation.

5.1 Equivalence relation

We axiomatize the desired property as a collection of equivalence judgments (on states, stores, terms and expressions) that are summarized in Table 2.

Stores and States Certainly values in high security store locations are out of view. Less obviously, some values in the low security locations are out of view as well: if a low security store location appears only out of view, its value is also out of view. We parametrize the store equivalence judgment by a set U of in-view store locations. Two (well-typed) stores are equivalent only if their in view values are equivalent:

$$\frac{\begin{array}{l}\vdash H_1 : \Sigma_1 \quad \vdash H_2 : \Sigma_2 \\ \Sigma_1 \restriction U = \Sigma_2 \restriction U\end{array} \qquad \Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : \Sigma_1(\ell) \text{ for } \ell \in U}{\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)} \quad (58)$$

Where the notation $\Sigma \restriction X$ is the restriction of Σ to just the locations in the set X.

For a pair of computation states, only low security locations that are common to both computations are in-view. Since allocation does leak information, it is possible for two programs to allocate different low security locations while executing high security sub-computations. However such locations are out of view for the low security sub-computation.

Pairs of computation states are equivalent if their stores are equivalent on the in-view locations, and if they have equivalent expressions:

$$\frac{\vdash (H_1 : \Sigma_1) \approx_\zeta^{\mathrm{dom}(H_1) \cap \mathrm{dom}(H_2) \cap \downarrow(\zeta)} (H_2 : \Sigma_2) \quad \Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o A}{\vdash (H_1, \Sigma_1, E_1) \approx_\zeta (H_2, \Sigma_2, E_2) \div_o A} \quad (59)$$

Where $\downarrow(\zeta) = \{\ell \mid \mathsf{Level}(\ell) \sqsubseteq \zeta\}$ is the set of all low security locations.

Terms and Expressions High security sub-computations of a program may return different values to the low security sub-computations. However, by the upcall rule, the type of those values must be informative only at high security.

Values of a type that is informative only at high security are out of view. As a result, any two values of such a type are equivalent since two such values vacuously agree on their in view parts:

$$\frac{\Sigma_1; \Gamma \vdash V_1 : A \quad \Sigma_2; \Gamma \vdash V_2 : A \quad \vdash A \nearrow a \quad a \not\sqsubseteq \zeta}{\Sigma_1; \Sigma_2; \Gamma \vdash V_1 \approx_\zeta V_2 : A} \quad (39)$$

The remaining rules for term and expression equivalence are congruence rules that merely require corresponding sub-terms or sub-expressions to be equivalent. They are listed in Appendix A.3.

Having defined the equivalence judgments, we establish several structural properties. First we show that it is reflexive, symmetric and transitive, so that it is indeed a partial equivalence relation. The proofs are given in Appendix C.2.

Next, we establish inversion and functionality. Inversion will let us by cases in subsequent proof. Functionality is the analog of a substitution for the equivalence judgment.

Table 2: Equivalence judgments

Judgment	**Meaning**
$\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$	Term Equivalence
$\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o A$	Expression Equivalence
$\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)$	Store Equivalence
$\vdash S_1 \approx_\zeta S_2 \div_o A$	State Equivalence

Lemma 5.1 (Equivalent Term Inversion). *If $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$ then either*

> *there exists a B, such that $\vdash B \leq A$ and $\vdash B \nearrow a$ and $a \not\sqsubseteq \zeta$ and M_1 and M_2 are values and $\Sigma_i; \Gamma \vdash M_i : B$ for $i = 1, 2$,*

or

1. *if $M_1 = x$ then $\vdash \Gamma(x) \leq A$ and $M_2 = x$.*
2. *if $M_1 = *$ then $\vdash 1 \leq A$ and $M_2 = *$.*
3. *if $M_1 = \mathsf{true}$ then $\vdash \mathsf{bool} \leq A$ and $M_2 = \mathsf{true}$.*
4. *if $M_1 = \mathsf{false}$ then $\vdash \mathsf{bool} \leq A$ and $M_2 = \mathsf{false}$.*
5. *if $M_1 = \mathsf{if}\ N_1\ \mathsf{then}\ P_{11}\ \mathsf{else}\ P_{12}$ then $M_2 = \mathsf{if}\ N_2\ \mathsf{then}\ P_{21}\ \mathsf{else}\ P_{22}$ and $\Sigma_1; \Sigma_2; \Gamma \vdash N_1 \approx_\zeta N_2 : \mathsf{bool}$ and $\Sigma_1; \Sigma_2; \Gamma \vdash P_{11} \approx_\zeta P_{21} : B$ and $\Sigma_1; \Sigma_2; \Gamma \vdash P_{12} \approx_\zeta P_{22} : B'$ and $\vdash B \leq A, \vdash B' \leq A$*
6. *if $M_1 = \ell$ then $\vdash \mathsf{ref}_b\, B \leq A$ and $M_2 = \ell$ and $b \sqsubseteq \zeta$ and $\Sigma_i(\ell) = B$ for $i = 1, 2$ and $\mathsf{Level}(\ell) = b$*
7. *if $M_1 = \lambda x : B.N_1$ then $\vdash B \to C \leq A$ and $M_2 = \lambda x : B.N_2$ and $\Sigma_1; \Sigma_2; \Gamma, x : B \vdash N_1 \approx_\zeta N_2 : C$*
8. *if $M_1 = \mathsf{val}\ E_1$ then $\vdash \bigcirc_o B \leq A$ and $M_2 = \mathsf{val}\ E_2$ and $\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o B$*
9. *if $M_1 = N_1 P_1$ then $M_2 = N_2 P_2$ and $\Sigma_1; \Sigma_2; \Gamma \vdash N_1 \approx_\zeta N_2 : B \to C$ and $\Sigma_1; \Sigma_2; \Gamma \vdash P_1 \approx_\zeta P_2 : B$ and $\vdash C \leq A$*

Proof. by induction on the derivation. □

Lemma 5.2 (Functionality). *If $\Sigma_1; \Sigma_2; \Gamma, \Gamma' \vdash M_1 \approx_\zeta M_2 : A$ then*

1. *if $\Sigma_1; \Sigma_2; \Gamma, x : A, \Gamma' \vdash N_1 \approx_\zeta N_2 : C$ then $\Sigma_1; \Sigma_2; \Gamma, \Gamma' \vdash N_1[M_1/x] \approx_\zeta N_2[M_2/x] : C$*
2. *if $\Sigma_1; \Sigma_2; \Gamma, x : A, \Gamma' \vdash E_1 \approx_\zeta E_2 \div_o C$ then $\Sigma_1; \Sigma_2; \Gamma, \Gamma' \vdash E_1[M_1/x] \approx_\zeta E_2[M_2/x] \div_o C$.*

$$
\begin{tikzcd}
& \vdash\ M_1 \arrow[dl] & \approx_\zeta & M_2 \ :\ A \arrow[dr] & \\
M_1' \arrow[dr, dotted, "*"] & & & & M_2' \arrow[dl, dotted, "*"'] \\
& \vdash\ M_1'' & \approx_\zeta & M_2'' \ :\ A &
\end{tikzcd}
$$

Figure 3: Informal statement of the Term Hexagon Lemma

Proof. by induction on the TD. □

Although we established Functionality for arbitrary terms to be substituted for x, as befits a call by value language, we only substitute values in the proof of non-interference.

5.2 Hexagon lemmas

Non-interference will follow as a consequence of a pair of Hexagon Lemmas: one for terms and one for expressions. We show that by starting with some two related terms (or expressions) that both take a step, we can find zero or more steps that each of them could take so that we get back to related states (respectively, expression).

The lemma for terms is summarized in Figure 3, the name "Hexagon Lemma" is motivated by the shape of this diagram.

Lemma 5.3 (Term Hexagon Lemma). *For all ζ, if $\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : A$ and $M_1 \to M_1'$ and $M_2 \to M_2'$ and $M_1' \downarrow$ and $M_2' \downarrow$, then there exist M_1'', M_2'' such that $M_1' \to^* M_1''$, $M_2' \to^* M_2''$, $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : A$*

Proof by induction on the given derivation. Most cases are vacuous. In the cases of function application and if-then-else, proceed by subcases on $M_1 \to M_1'$. The full proof is given in Appendix C.3.

Before we prove the Hexagon Lemma for expressions, we need the following technical lemma. Intuitively it says that a computation running above the security level of the observer does not affect the store in any way that is visible to the observer.

Lemma 5.4 (Single High Security Step). *If $\vdash (H, \Sigma, E) \div_o A$, $o = (r, w)$ and $w \not\sqsubseteq \zeta$, and $(H, \Sigma, E) \to (H', \Sigma', E')$ then $\vdash (H : \Sigma) \approx_\zeta^{\mathrm{dom}(\Sigma) \cap \downarrow(\zeta)} (H' : \Sigma')$.*

Proof by induction on the evaluation relation. The details are given in Appendix C.4.

Corollary 5.5 (Multiple High Security Steps). *If* $\vdash (H, \Sigma, E) \div_o A$*,* $o = (r, w)$ *and* $w \not\sqsubseteq \zeta$*, and* $(H, \Sigma, E) \rightarrow^n (H', \Sigma', E')$ *then* $\vdash (H : \Sigma) \approx_\zeta^{\mathrm{dom}(\Sigma) \cap \downarrow(\zeta)} (H' : \Sigma')$*.*

Proof. by induction on n, the number of steps.

By inversion,

- $\vdash H : \Sigma$
- $\Sigma; \cdot \vdash E \div_o A$

If $n = 0$, the result follows by Reflexivity.
If $n > 0$, then $(H, \Sigma, E) \rightarrow (H'', \Sigma'', E'') \rightarrow^{n-1} (H', \Sigma', E')$.

1. By Single High Security Step, $\vdash (H : \Sigma) \approx_\zeta^U (H'' : \Sigma'')$ where $U = \mathrm{dom}(\Sigma) \cap \downarrow(\zeta)$
2. By Preservation, $\vdash (H'', \Sigma'', E'') \div_o A$
3. By IH, $\vdash (H'' : \Sigma'') \approx_\zeta^{U'} (H' : \Sigma')$ where $U' = \mathrm{dom}(\Sigma') \cap \downarrow(\zeta)$
4. By Store Size, $\Sigma' \supseteq \Sigma$
5. Therefore, $U' \supseteq U$
6. By Store Equivalence Coarsening, $\vdash (H'' : \Sigma'') \approx_\zeta^U (H' : \Sigma')$
7. By Transitivity $\vdash (H : \Sigma) \approx_\zeta^U (H' : \Sigma')$ since $U \supseteq U' \cap U$

□

The following corollary of the Multiple High Security Steps Lemma is used in the proof of the hexagon lemma for expressions. Essentially it says that given two related stores, executing any two computations (even ones at different types!) that are high security with respect to a ζ-observer will produce stores that are indistinguishable by a ζ-observer.

One complication in this corollary is that evaluation of two distinct computation states S_1, S_2 may inadvertently allocate the same store location ℓ for distinct purposes. However we will show that for each such ℓ, we may choose an element of the α-equivalence class of S_1 or S_2 such that all such accidental sharing is eliminated.

Corollary 5.6 (High Security Step (HSS)). *Given* (H_1, Σ_1, E_1) *and* (H_2, Σ_2, E_2) *such that*

- $\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)$ *where* $U = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap \downarrow(\zeta)$*,*
- $\Sigma_i; \cdot \vdash E_i \div_{o_i} C_i$ *for some* $o_i = (r_i, w_i), C_i$ *with* $w_i \not\sqsubseteq \zeta$ *for* $i = 1, 2$*,*

and if $(H_1, \Sigma_1, E_1) \to^* (H_1', \Sigma_1', E_1')$ *and* $(H_2, \Sigma_2, E_2) \to^* (H_2', \Sigma_2', E_2')$ *then*

- $\vdash (H_1' : \Sigma_1') \approx_\zeta^U (H_2' : \Sigma_2')$
- *and* $U = \mathrm{dom}(\Sigma_1') \cap \mathrm{dom}(\Sigma_2') \cap {\downarrow}(\zeta)$

Proof. 1. By Regularity of Equivalence, $\vdash (H_i : \Sigma_i)$ for $i = 1, 2$

2. By Multiple High Security Steps, for $i = 1, 2$:
   - $\vdash (H_i : \Sigma_i) \approx_\zeta^{U_i} (H_i' : \Sigma_i')$ where $U_i = \mathrm{dom}(\Sigma_i) \cap {\downarrow}(\zeta)$
3. By Regularity, $\vdash H_i' : \Sigma_i'$ for $i = 1, 2$
4. Note also that $U \subseteq U_i$ for $i = 1, 2$
5. Consider $\ell \in U$
   (a) $\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : A$, $\Sigma_1(\ell) = \Sigma_2(\ell) = A$
   (b) Evidently, also $\ell \in U_i$
   (c) $\Sigma_i; \Sigma_i'; \cdot \vdash H_i(\ell) \approx_\zeta H_i'(\ell) : A$, $\Sigma_i(\ell) = \Sigma_i'(\ell) = A$, for $i = 1, 2$
   (d) By Symmetry, $\Sigma_1'; \Sigma_1; \cdot \vdash H_1'(\ell) \approx_\zeta H_1(\ell) : A$
   (e) By Transitivity, $\Sigma_1'; \Sigma_2'; \cdot \vdash H_1'(\ell) \approx_\zeta H_2'(\ell) : A$
6. So, by rule (58), $\vdash (H_1' : \Sigma_1') \approx_\zeta^U (H_2' : \Sigma_2')$
7. By Regularity, $\mathrm{dom}(H_i') = \mathrm{dom}(\Sigma_i')$ for $i = 1, 2$
8. Now let $U' = \mathrm{dom}(\Sigma_1') \cap \mathrm{dom}(\Sigma_2') \cap {\downarrow}(\zeta)$
9. By Store Size, $\Sigma_i' \supseteq \Sigma_i$, so $U' \supseteq U$
10. Suppose $\ell \in U' \setminus U$
    (a) Since $\ell \in U'$, $\ell \in \mathrm{dom}(\Sigma_i')$ for $i = 1, 2$
    (b) Since $\ell \notin U$, then $\ell \notin \mathrm{dom}(\Sigma_i)$ for at least one of $i = 1$ or $i = 2$
    (c) Suppose $\ell \notin \mathrm{dom}(\Sigma_1)$ (the other case is similar)
    (d) Choose a fresh store location $\ell' \notin \mathrm{dom}(\Sigma_1') \cup \mathrm{dom}(\Sigma_2')$ with $\mathsf{Level}(\ell') = \mathsf{Level}(\ell)$, and systematically rename ℓ with ℓ' in (H_1', Σ_1', E_1').
    (e) Evidently we have an element of the α-equivalence class of (H_1', Σ_1', E_1') where $\ell \notin \mathrm{dom}(H_1')$
11. So $U' = U$

□

We may now show a hexagon lemma for expressions.

Lemma 5.7 (Hexagon Lemma). *For all* ζ*, if* $o = (r, w)$ *with* $r \sqsubseteq \zeta$*, and if*

- $\vdash S_1 \approx_\zeta S_2 \div_o C$

- $S_1 \to S_1'$, $S_2 \to S_2'$
- $S_1' \downarrow$, $S_2' \downarrow$

then there exist S_1'', S_2'' *such that*

- $S_1' \to^* S_1''$, $S_2' \to^* S_2''$
- $\cdot \vdash S_1'' \approx_\zeta S_2'' \div_o C$

The full proof is given in Appendix C.5. We highlight several important cases below.

We proceed first by Inversion on $\vdash S_1 \approx_\zeta S_2 \div_o C$, to get that

- $S_1 = (H_1, \Sigma_1, E_1)$, $S_2 = (H_2, \Sigma_2, E_2)$
- $\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)$ where $U = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap \downarrow(\zeta)$
- $\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C$

And then, by induction on the derivation of $\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C$. In each case we exhibit the appropriate $S_i'' = (H_1'', \Sigma_1'', E_1''), S_2'' = (H_2'', \Sigma_2'', E_2'')$.

- Case
$$\frac{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_{(r',w)} C \quad \vdash C \nearrow r'}{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_{(\bot,w)} C} \quad (51)$$

  By pattern matching, $r = \bot$

  Consider two subcases: either $r' \sqsubseteq \zeta$ or $r' \not\sqsubseteq \zeta$. The former case follows eventually by the induction hypothesis. In the latter case,

  1. Since $r' \sqsubseteq w$, then $w \not\sqsubseteq \zeta$
  2. Since $S_i' \downarrow$, $(H_i, \Sigma_i, E_i) \to^+ (H_i'', \Sigma_i'', [V_i])$ for some $S_i'' = (H_i'', \Sigma_i'', [V_i])$ for $i = 1, 2$
  3. Therefore we can apply HSS to get
     - $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$
     - $U = \mathrm{dom}(H_1'') \cap \mathrm{dom}(H_2'') \cap \downarrow(\zeta)$
  4. By repeatedly applying Preservation, $\Sigma_i''; \cdot \vdash [V_i] \div_{(r',w)} C$ for $i = 1, 2$
  5. And by various typing rules, $\Sigma_1''; \Sigma_2''; \cdot \vdash [V_1] \approx_\zeta [V_2] \div_o C$

- Case
$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : A}{\Sigma_1; \Sigma_2; \cdot \vdash \mathsf{ref}_a\,(M_1 : A) \approx_\zeta \mathsf{ref}_a\,(M_2 : A) \div_{(\bot,\top)} \mathsf{ref}_a\,A} \quad (55)$$

  By pattern matching, $E_i = \mathsf{ref}_a\,(M_i : A)$, $o = (\bot, \top)$, $C = \mathsf{ref}_a\,A$

  There are two possible evaluation rules for $(H_1, \Sigma_1, E_1) \to (H_1', \Sigma_1', E_1')$

  - Subcase Ref1 follows eventually from the Term Hexagon Lemma.

– Subcase Ref: M_1 value, $H_1' = H_1\{\ell_1 \mapsto M_1\}$, $\Sigma_1' = \Sigma_1\{\ell_1 : A\}$, $E_1' = [\ell_1]$, where $\ell_1 \notin \mathrm{dom}(H_1)$, $\mathsf{Level}(\ell_1) = a$

1. By Equivalent Values, M_2 is a value
2. Only Ref rule is applicable to $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$: $H_2' = H_2\{\ell_2 \mapsto M_2\}$, $\Sigma_2' = \Sigma_2\{\ell_2 : A\}$, $E_2' = [\ell_2]$, where $\ell_2 \notin \mathrm{dom}(H_2)$, $\mathsf{Level}(\ell_2) = a$
3. Consider two subcases now, either $a \sqsubseteq \zeta$ or $a \not\sqsubseteq \zeta$:
   * Subcase $a \sqsubseteq \zeta$
   (a) Since in both S_1' and S_2', ℓ_1 and ℓ_2 are freshly allocated, we may α-vary S_1', S_2' such that $\ell_1 = \ell_2 = \ell$ for an appropriate ℓ
   (b) Then $\mathsf{Level}(\ell) = a$, $\ell \notin \mathrm{dom}(H_1) \cup \mathrm{dom}(H_2)$
   (c) Let $S_i'' = (H_i\{\ell \mapsto M_i\}, \Sigma_i\{\ell : A\}, [\ell])$ for $i = 1, 2$
   (d) The result follows since the freshly allocated location is (by construction) in the set $U'' = U \cup \{\ell\}$ of common locations between S_1'' and S_2'', and it contains equivalent values.
   * Subcase $a \not\sqsubseteq \zeta$
   In this case the newly allocated locations ℓ_1, ℓ_2 are not in the common set of S_1'' and S_2'' since they have high security levels. Furthermore $\Sigma_1\{\ell_1 : A\}; \Sigma_2\{\ell_2 : A\}; \cdot \vdash \ell_1 \approx_\zeta \ell_2 : \mathsf{ref}_a\, A$, since $\vdash \mathsf{ref}_a\, A \nearrow a$ and $a \not\sqsubseteq \zeta$. The result follows.

• Case
$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : \mathsf{refr}_a\, C}{\Sigma_1; \Sigma_2; \cdot \vdash !M_1 \approx_\zeta !M_2 \div_{(a,\top)} C} \; (56)$$

By pattern matching, $E_i = !M_i$, $o = (r, w) = (a, \top)$. Recall that $a = r \sqsubseteq \zeta$

There are two applicable rules for $(H_1, \Sigma_1, E_1) \to (H_1', \Sigma_1', E_1')$

– Subcase Bang1 follows by the Term Hexagon Lemma

– Subcase Bang: $M_1 = \ell_1$, $H_1' = H_1$, $\Sigma_1' = \Sigma_1$, $E_1' = [H_1(\ell_1)]$

1. By Equivalent Values, M_2 is a value
2. The single applicable evaluation rule for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$ is Bang: $M_2 = \ell_2$, $H_2' = H_2$, $\Sigma_2' = \Sigma_2$, $E_2' = [H_2(\ell_2)]$
3. Let $S_i'' = (H_i, \Sigma_i, [H_i(\ell_i)])$ for $i = 1, 2$
4. So it only remains to show that $\Sigma_1; \Sigma_2; \cdot \vdash H_i(\ell_1) \approx_\zeta H_2(\ell_2) : C$
5. By Equivalent Term Inversion on $\Sigma_1; \Sigma_2; \cdot \vdash \ell_1 \approx_\zeta \ell_2 : \mathsf{refr}_a\, C$, there are two possibilities:
   * Either $\Sigma_i; \cdot \vdash \ell_i : B$ and $\vdash B \leq \mathsf{refr}_a\, C$ and $\vdash B \nearrow b$ and $b \not\sqsubseteq \zeta$
   It follows by inversions that B is either $\mathsf{refr}_{b'}\, B'$ or $\mathsf{ref}_{b'}\, B'$ and in either case $\vdash B' \nearrow c$ for some $c \not\sqsubseteq \zeta$. The result follows.

* Or $\ell_1 = \ell_2 = \ell$ where $\mathsf{Level}(\ell) = b \sqsubseteq \zeta$, and $\vdash \mathsf{ref}_b\, \Sigma_1(\ell) \leq \mathsf{refr}_a\, C$ and $\Sigma_1(\ell) = \Sigma_2(\ell)$. This case follows since ℓ is in the common set of Σ_1, Σ_2 and since the stores are equivalent.

- Case

$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : \mathsf{refw}_a\, A \quad \Sigma_1; \Sigma_2; \cdot \vdash N_1 \approx_\zeta N_2 : A}{\Sigma_1; \Sigma_2; \cdot \vdash M_1 := N_1 \approx_\zeta M_2 := N_2 \div_{(\bot, a)} 1} \quad (57)$$

By pattern matching, $E_i = M_i := N_i$, $o = (r, w) = (\bot, a)$, $C = 1$

There are three applicable rules for $(H_1, \Sigma_1, E_1) \to (H_1', \Sigma_1', E_1')$. If the rule was Assn1 or Assn2, the result follows from the Term Hexagon Lemma.

Otherwise, the rule was Assn, and we have: $M_1 = \ell_1$, N_1 value, $H_1' = H_1\{\ell_1 \mapsto N_1\}$, $\Sigma_1' = \Sigma_1$, $E_1' = [*]$

1. By Equivalent Values, M_2, N_2 are values
2. The only applicable evaluation rule for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$ is Assn, and we have: $M_2 = \ell_2$, $H_2' = H_2\{\ell_2 \mapsto N_2\}$, $\Sigma_2' = \Sigma_2$, $E_2' = [*]$
3. Let $S_i'' = (H_i\{\ell_i \mapsto N_i\}, \Sigma_i, [*])$. It suffices to show that the updated stores are still equivalent.
4. By Equivalent Term Inversion on $\Sigma_1; \Sigma_2; \cdot \vdash \ell_1 \approx_\zeta \ell_2 : \mathsf{refw}_a\, A$, there are two possibilities:
   - Either $\Sigma_i; \cdot \vdash \ell_i : B$ and $\vdash B \leq \mathsf{refw}_a\, A$, $\vdash B \nearrow b$ and $b \not\sqsubseteq \zeta$
     By Subtyping Inversion, either $B = \mathsf{refw}_{b'}\, B'$ or $B = \mathsf{ref}_{b'}\, B'$ and in either case $\vdash A \leq B'$ and $a \sqsubseteq b'$
     * If $B = \mathsf{refw}_{b'}\, B'$, then it eventually follows from inversions that $\mathsf{Level}(\ell_i) \not\sqsubseteq \zeta$, and so the ℓ_i are not in the common set U of locations, and the result follows.
     * If $B = \mathsf{ref}_{b'}\, B'$
     (a) By Subtyping Inversion, $B' = \Sigma_i(\ell_i)$ and $b' = \mathsf{Level}(\ell_i)$ for $i = 1, 2$
     (b) By Informativeness Inversion, $b \sqsubseteq b' \sqcup c$ and $\vdash B' \nearrow c$ for some c
     (c) Since $b \not\sqsubseteq \zeta$, either $b' \not\sqsubseteq \zeta$ or $c \not\sqsubseteq \zeta$
     (d) If $b' \not\sqsubseteq \zeta$, we can use the same argument as the previous subcase: $B = \mathsf{refw}_{b'}\, B'$.
     (e) So instead suppose $b' \sqsubseteq \zeta$; it must be the case that $c \not\sqsubseteq \zeta$.
     (f) Consider ℓ_1 (the argument for ℓ_2 is symmetric)
     (g) Evidently $\mathsf{Level}(\ell_1) = b' \sqsubseteq \zeta$, so suppose $\ell_1 \in U$ (if not, same argument as previous subcase)

(h) If $\ell_1 = \ell_2$ then the situation is the same as the next subcase ($\ell_1 = \ell_2 = \ell$, ...) below; so suppose ℓ_1 differs from ℓ_2

(i) So $\ell_1 \in \mathrm{dom}(\Sigma_2) = \mathrm{dom}(H_2)$

(j) By heap typing inversion, $\Sigma_2; \cdot \vdash H_2(\ell_1) : \Sigma_2(\ell_1)$

(k) Since $\ell_1 \in U$, $\Sigma_2(\ell_1) = \Sigma_1(\ell_1) = B'$

(l) By rule (39), $\Sigma_1; \Sigma_2; \cdot \vdash N_1 \approx_\zeta H_2(\ell_1) : \Sigma_1(\ell_1)$

(m) Therefore for all $\ell \in U$, $\Sigma_1''; \Sigma_2''; \cdot \vdash H_1''(\ell) \approx_\zeta H_2''(\ell) : \Sigma_1''(\ell)$

(n) So by rule (58), $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$

– Or $\ell_1 = \ell_2 = \ell$ and $\mathsf{Level}(\ell) \sqsubseteq \zeta$ and $\Sigma_1(\ell) = \Sigma_2(\ell)$, and $\vdash \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma_1(\ell) \leq \mathsf{refw}_a\, A$
We can show that ℓ is in the common set U of locations; the result follows by a straightforward derivation.

5.3 Non-interference theorem

Finally we are ready to prove non-interference. Starting with some initial store H (well-typed with store type Σ) and an expression to execute E with a free variable x, if we plug in different values V_1, V_2 for x, then provided that the in-view parts of V_1, V_2 are equivalent, we expect that if the resulting programs $(H, \Sigma, E[V_1/x])$, $(H, \Sigma, E[V_2/x])$ run to termination, the resulting terminal states will be equivalent on their in view parts.

Theorem 5.8 (Non-interference). *If* $\vdash H : \Sigma$ *and* $\Sigma; x : A \vdash E \div_{(r,w)} B$ *and if* $\Sigma; \Sigma; \cdot \vdash V_1 \approx_r V_2 : A$ *then if* $(H, \Sigma, E[V_1/x]) \rightarrow^* S_1$ *and* $(H, \Sigma, E[V_2/x]) \rightarrow^* S_2$ *and both* S_1, S_2 *are terminal, then* $\vdash S_1 \approx_r S_2 \div_{(r,w)} B$

Proof. By Reflexivity and Functionality, we can show that

$$\Sigma; \Sigma; \cdot \vdash E[V_1/x] \approx_r E[V_2/x] \div_{(r,w)} B$$

By repeated application of the Hexagon Lemma, the two computations evaluate to equivalent terminal states. Since the operational semantics are deterministic, those terminal states are S_1 and S_2, respectively. □

6 Encoding value-oriented secure languages

Our account differs substantially from prior secure programming languages where each value has a security level. In such languages, terms are classified by security types: pairs of an ordinary type and a security level. The type system ensures that each term is assigned a security level at least as high as the security level of the terms contributing to it. In our account only the store provides security. A natural question is whether we sacrifice expressive power in comparison to value-oriented secure languages.

We will show that our language is at least as expressive by showing how to embed several value-oriented secure languages in our account. The embeddings are not only type correct, but also preserve security properties of the source languages. We assume some familiarity with SLam [3].

$$
\begin{array}{llll}
t & \in & \textit{types} & ::= & 1 \mid \mathsf{bool} \mid s_1 \to s_2 \\
s & \in & \textit{security types} & ::= & (t, a) \\
\\
e & \in & \textit{expressions} & ::= & x \mid *_a \mid \mathsf{true}_a \mid \mathsf{false}_a \\
 & & & \mid & \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \\
 & & & \mid & (\lambda x : s.e)_a \\
 & & & \mid & \mathsf{protect}_a e
\end{array}
$$

Figure 4: SLam Calculus Syntax

6.1 A purely functional language: SLam calculus

Our presentation of the SLam calculus is based on the purely-functional call-by-value variant presented in Abadi, *et al.* [1]. The types of SLam include unit, booleans and functions from security annotated values to security annotated results. The syntax of expressions is summarized in Figure 4. In addition to variables, SLam has a security-annotated unit value, and security-annotated boolean values, and functions. Since functions are themselves values, abstractions are annotated not only with the security type of the argument, but also with a security level for the function itself. One additional operation $\mathsf{protect}_a e$ is used to increase the security level of the value of e.

The complete operational semantics and typing rules of SLam are presented in Appendix D.

The idea of the embedding is to translate expressions e of SLam to expressions E (not terms) of our language in such a manner that the operation level of E is $(\perp, \top)$. That is, the translated expression does not read (informative) values above $\perp$ nor does it write below $\top$.

In order to get the desired effect, we store the result of e in a ref cell. The ref has the effect of sealing the return value of e at its corresponding security level: values stored in a ref cell are informative only at the security level of the ref. So for example if e had security type $(\mathsf{bool}, \top)$, the result of the translation would be a newly allocated store location of type $\mathsf{ref}_\top\ \mathsf{bool}$.

Finally, since a $\mathsf{ref}_a\ A$ is informative only above a, we may demote the read level of E to $\perp$. And since SLam is purely functional, there are no writes in the translation, so the write level is $\top$.

Suppose x is a SLam variable with security type $(\mathsf{bool}, \top)$. The SLam expression

$$\mathsf{if}\ x\ \mathsf{then}\ \mathsf{false}_\top\ \mathsf{else}\ \mathsf{true}_\top$$

negates x. In the translation, x stands for a $\top$-level store location containing that contains a boolean: $\mathsf{refr}_\top\ \mathsf{bool}$. The expression $\mathsf{if}\ x\ \mathsf{then}\ \mathsf{false}\ \mathsf{else}\ \mathsf{true}$ is translated into the expression:

$$\begin{array}{l} \mathsf{let}\ y = !x\ \mathsf{in} \\ \mathsf{let\ val}\ y' = (\mathsf{if}\ y\ \mathsf{then\ val\ ref}_\top\ (\mathsf{false} : \mathsf{bool}) \\ \qquad\qquad \mathsf{else\ val\ ref}_\top\ (\mathsf{true} : \mathsf{bool}))\ \mathsf{in} \\ [y'] \end{array}$$

First y is bound to the value stored in x. That sub-expression has operation level $(\top, \top)$, there so must the other sub-expressions for the translation to be well-typed. Based on the value of y, y' is bound to the result of one of the two allocation expressions, and the value of y' is the result of the whole expression. The entire expression has operation level $(\top, \top)$ and return type $\mathsf{ref}_\top\ \mathsf{bool}$. Since that type is informative only at $\top$, the operation level may be lowered to $(\perp, \top)$.

The Encoding We define the function $\overline{\cdot}$ to formalize our embedding of SLam security types:

$$\begin{array}{rcl} \overline{(t, a)} & = & \mathsf{refr}_a\ \overline{t} \\ \\ \overline{1} & = & 1 \\ \overline{\mathsf{bool}} & = & \mathsf{bool} \\ \overline{s_1 \to s_2} & = & \overline{s_1} \to \bigcirc_{(\perp,\top)} \overline{s_2} \end{array}$$

Since SLam security types are translated to store types in our language, the translation of SLam functions must be able to read from the store. So the translation of the SLam function type has a monadic codomain.

The encoding for SLam expressions is given by a judgment $\Gamma \vdash e : s \Rightarrow E$ in Figure 5). We assume that the metavariable y stands for variables in our calculus that are not used in SLam expressions. We use $\mathsf{run}\ M$ as syntactic sugar for $\mathsf{let\ val}\ y = M\ \mathsf{in}\ [y]$

Here $s_1 \leq s_2$ is the SLam subtyping judgment, given in Figure 6. The following lemma shows that type translation translates SLam subtypes into subtypes in our language.

Lemma 6.1. *If* $s_1 \leq s_2$ *then* $\vdash \overline{s_1} \leq \overline{s_2}$

Proof. by induction on the derivation of $s_1 \leq s_2$ □

Let $\overline{\Gamma}$ be defined by $\overline{\cdot} = \cdot$, $\overline{\Gamma, x : s} = \overline{\Gamma}, x : \overline{s}$. We show that the translation from SLam is type-correct. Then type-correctness of our embedding is given by the following theorem:

Theorem 6.2 (Type-correct translation). *If* $\Gamma \vdash e : s \Rightarrow E$ *then* $\{\}; \overline{\Gamma} \vdash E \div_{(\perp,\top)} \overline{s}$

The proof is by induction on the given derivation; it is given in Appendix D.3.

$$\boxed{\Gamma \vdash e : s \Rightarrow E}$$

$$\frac{}{\Gamma \vdash x : \Gamma(x) \Rightarrow [x]}$$

$$\frac{}{\Gamma \vdash *_a : (1, a) \Rightarrow \mathsf{ref}_a\,(* : 1)}$$

$$\frac{}{\Gamma \vdash \mathsf{true}_a : (\mathsf{bool}, a) \Rightarrow \mathsf{ref}_a\,(\mathsf{true} : \mathsf{bool})}$$

$$\frac{}{\Gamma \vdash \mathsf{false}_a : (\mathsf{bool}, a) \Rightarrow \mathsf{ref}_a\,(\mathsf{false} : \mathsf{bool})}$$

$$\frac{\Gamma \vdash e_1 : (\mathsf{bool}, a) \Rightarrow E_1 \quad \Gamma \vdash e_2 : s \Rightarrow E_2 \quad \Gamma \vdash e_3 : s \Rightarrow E_3}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : s \sqcup a \Rightarrow \begin{array}{l}\mathsf{let}\ y_1 = E_1\ \mathsf{in} \\ \mathsf{let}\ y_2 = !y_1\ \mathsf{in} \\ \mathsf{run}\ (\mathsf{if}\ y_2 \\ \qquad \mathsf{then}\ \mathsf{val}\ E_2 \\ \qquad \mathsf{else}\ \mathsf{val}\ E_3)\end{array}}$$

$$\frac{\Gamma, x : s \vdash e : s' \Rightarrow E}{\Gamma \vdash (\lambda x : s.e)_a : (s \to s', a) \Rightarrow \mathsf{ref}_a\,(\lambda x : \overline{s}.\mathsf{val}\ E : \overline{s} \to \bigcirc_{(\bot,\top)}\overline{s'})}$$

$$\frac{\Gamma \vdash e_1 : (s \to s', a) \Rightarrow E_1 \quad \Gamma \vdash e_2 : s \Rightarrow E_2}{\Gamma \vdash e_1 e_2 : s' \sqcup a \Rightarrow \begin{array}{l}\mathsf{let}\ y_1 = E_1\ \mathsf{in} \\ \mathsf{let}\ y_2 = !y_1\ \mathsf{in} \\ \mathsf{let}\ y_3 = E_2\ \mathsf{in} \\ \mathsf{run}\ (y_2\ y_3)\end{array}}$$

$$\frac{\Gamma \vdash e : s \Rightarrow E}{\Gamma \vdash \mathsf{protect}_a e : s \sqcup a \Rightarrow E}$$

$$\frac{\Gamma \vdash e : s_1 \Rightarrow E \quad s_1 \leq s_2}{\Gamma \vdash e : s_2 \Rightarrow E}$$

Figure 5: SLam encoding

$\boxed{s_1 \leq s_2}$

$$\frac{}{s_1 \leq s_1} \qquad \frac{s_1 \leq s_2 \quad s_2 \leq s_3}{s_1 \leq s_3}$$

$$\frac{a \sqsubseteq a'}{(1, a) \leq (1, a')} \qquad \frac{a \sqsubseteq a'}{(\mathsf{int}, a) \leq (\mathsf{int}, a')}$$

$$\frac{s_1' \leq s_1 \quad s_2 \leq s_2' \quad a \sqsubseteq a'}{(s_1 \to s_2, a) \leq (s_1' \to s_2', a')}$$

Figure 6: SLam subtyping judgment

Security Of course a type correct (but insecure) embedding could be constructed by ignoring the security levels of the source and placing everything at level $\perp$. We wish to show that the embedding is actually secure. To do so, we show that the following property, an instance of SLam's non-interference theorem, is preserved by the embedding:

> if $a \not\sqsubseteq b$, and if $\vdash f : ((t, a) \to (\mathsf{bool}, b), b)$ and $\vdash e_1, e_2 : (t, a)$ then $fe_1 \approxeq fe_2$

Corresponding to the SLam expression f is its translation F in our language, which has type:

$$\mathsf{refr}_b\, (\mathsf{refr}_a\, \overline{t} \to \bigcirc_{(\perp,\top)} \mathsf{refr}_b\, \mathsf{bool})$$

That is, F evaluates to a store location that contains a function from store locations (of appropriate type) to computations that return a store location containing a boolean. We would like to show that given different initial locations, the result store locations contain the same boolean value, since the security level a of the inputs is high in relation to the security level b of the results.

Theorem 6.3 (Adequacy of translation). *Suppose* $\vdash f : ((t, a) \to (\mathsf{bool}, b), b) \Rightarrow F$ *and* $a \not\sqsubseteq b$. *Let* H, Σ *be arbitrary such that* $\vdash H : \Sigma$, *and let* $\Sigma; \cdot \vdash \ell_1, \ell_2 : \mathsf{refr}_a\, \overline{t}$. *Let*

$$E = \begin{array}{l} \mathsf{let}\ y = F\ \mathsf{in} \\ \mathsf{let}\ z = !y\ \mathsf{in} \\ \mathsf{run}\ (z\ y_0) \end{array}$$

for $i = 1, 2$, *and suppose*

$$(H, \Sigma, E[\ell_1/y_0]) \to^* (H_1, \Sigma_1, [V_1])$$

and

$$(H, \Sigma, E[\ell_2/y_0]) \to^* (H_2, \Sigma_2, [V_2])$$

then $V_1 = \ell_1'$ *and* $V_2 = \ell_2'$ *and* $H_1(\ell_1') = H_2(\ell_2')$

Proof. By the type-correctness of the translation,

$$\{\}; \cdot \vdash F \div_{(\perp,\top)} \mathsf{refr}_b\, (\mathsf{refr}_a\, \overline{t} \to \bigcirc_{(\perp,\top)} \mathsf{refr}_b\, \mathsf{bool})$$

By various typing rules and reflexivity, it is easy to show that

$$\Sigma; \Sigma; y_0 : \mathsf{refr}_a\, \bar{t} \vdash E_1 \approx_b E_2 \div_{(b,\top)} \mathsf{refr}_b\, \mathsf{bool}$$

Furthermore, since $a \not\sqsubseteq b$,

$$\Sigma; \Sigma; \cdot \vdash \ell_1 \approx_b \ell_2 : \mathsf{refr}_a\, \bar{t}$$

So by non-interference,

$$\vdash (H_1, \Sigma_1, [V_1]) \approx_b (H_2, \Sigma_2, [V_2]) \div_{(b,\top)} \mathsf{refr}_b\, \mathsf{bool}$$

From which it follows for $i = 1, 2$ that $V_i = \ell'_i$ by a Canonical Forms lemma, and $\mathsf{Level}(\ell'_i) \sqsubseteq b$ by inversion . By inversion on the equivalence derivations, it follows that $\ell'_1 = \ell'_2$ and by inversion on the store equivalence, it follows that

$$\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell'_1) \approx_b H_2(\ell'_2) : \mathsf{bool}$$

And since bool is informative only above $\perp$, by inversion on the equivalence, $H_1(\ell'_1) = H_2(\ell'_2)$ □

6.2 An imperative language: $\lambda_{\mathsf{SEC}}^{\mathsf{REF}}$

When a computation analyzes a value of a datatype by cases, each arm — by virtue of control flow — gains information about the subject of the case expression. In a purely functional setting, that additional information may only be used to compute the return value of the expression. Thus it suffices to require the return type of each arm (and thus the entire case expression) to be at least as secure as the case subject.

On the other hand, in an imperative setting, information gained via control-flow may leave an expression non-locally (*e.g.,* via a write to the store). As a result, it becomes necessary to track such *implicit flows* of information. Secure imperative languages use a so-called *program counter security level*, pc, as a lower bound on the information that a computation may gain via control flow. Consequently, the results and effects of each expression must be at least as secure as any information gained via control flow.

In contrast to value-oriented secure programming languages, in our account we expect that case analysis is at the term level, and thus the arms of the case term do not have side-effects. We show that our approach is at least as expressive as imperative value-oriented secure languages.

We consider the language $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ (summarized in Figure 7) of Zdancewic [15]. In addition to unit and boolean types, it has function types that are annotated with a lower bound on the write effects of the function body, and store locations. The base values of $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ are annotated with a security level inside expressions.

The typing rules for $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ are given by a pair of mutually recursive judgments for base values and expressions, given in Figures 8 and 9.

The following key property is maintained by the $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ typing judgment. Intuitively, it captures the idea that the value of an expression is at least as secure as the information that the expression gains via implicit information flow.

$$
\begin{array}{rcll}
t & \in & \textit{types} & ::= \quad 1 \mid \mathsf{bool} \mid s_1 \xrightarrow{\mathsf{pc}} s_2 \mid \mathsf{ref}\ s \\
s & \in & \textit{security types} & ::= \quad (t, a) \\
\\
bv & \in & \textit{base values} & ::= \quad * \mid \mathsf{true} \mid \mathsf{false} \mid \ell \mid \lambda[\mathsf{pc}]x : s.e \\
\\
e & \in & \textit{expressions} & ::= \quad x \mid bv_a \mid \\
 & & & \qquad \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \\
 & & & \qquad \mid e_1 e_2 \\
 & & & \qquad \mid (\mathsf{ref}\ (e : s)) \mid\ !e \mid e := e'
\end{array}
$$

Figure 7: $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ Syntax

$\boxed{\Sigma; \Gamma \vdash bv : t}$

$$\frac{}{\Sigma; \Gamma \vdash * : 1}$$

$$\frac{}{\Sigma; \Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\Sigma; \Gamma \vdash \mathsf{false} : \mathsf{bool}}$$

$$\frac{}{\Sigma; \Gamma \vdash \ell : \Sigma(\ell)}$$

$$\frac{\Sigma; \Gamma, x : s[\mathsf{pc}] \vdash e : s'}{\Sigma; \Gamma \vdash \lambda[\mathsf{pc}]x : s.e \ : s \xrightarrow{\mathsf{pc}} s'}$$

$$\frac{\Sigma; \Gamma \vdash bv : t' \quad \vdash t' \leq t}{\Sigma; \Gamma \vdash bv : t}$$

Figure 8: $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ base value typing.

$\boxed{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s}$

$$\frac{}{\Sigma;\Gamma, x : s[\mathsf{pc}] \vdash x : s \sqcup \mathsf{pc}}$$

$$\frac{\Sigma;\Gamma \vdash bv : t}{\Sigma;\Gamma[\mathsf{pc}] \vdash bv_a : (t, a \sqcup \mathsf{pc})}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{bool}, a) \quad \Sigma;\Gamma[\mathsf{pc} \sqcup a] \vdash e_2 : s \quad \Sigma;\Gamma[\mathsf{pc} \sqcup a] \vdash e_3 : s}{\Sigma;\Gamma[\mathsf{pc}] \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : s}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (s' \xrightarrow{\mathsf{pc}'} s, a) \quad \Sigma;\Gamma[\mathsf{pc}] \vdash e_2 : s' \quad \mathsf{pc} \sqcup a \sqsubseteq \mathsf{pc}'}{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 e_2 : s \sqcup a}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s}{\Sigma;\Gamma[\mathsf{pc}] \vdash (\mathsf{ref}\ (e : s)) : (\mathsf{ref}\ s, \mathsf{pc})}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e : (\mathsf{ref}\ s, a)}{\Sigma;\Gamma[\mathsf{pc}] \vdash !e : s \sqcup a}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{ref}\ (t, b), a) \quad \Sigma;\Gamma[\mathsf{pc}] \vdash e_2 : (t, b) \quad a \sqsubseteq b}{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 := e_2 : (1, \mathsf{pc})}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s' \quad \vdash s' \leq s}{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s}$$

Figure 9: $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ expression typing.

Lemma 6.4. *If* $\Sigma; \Gamma[\mathsf{pc}] \vdash e : (t, a)$ *then* $\mathsf{pc} \sqsubseteq a$.

The proof of this fact appears as Lemma 3.2.1 in Zdancewic's thesis [15].

Encoding As with purely functional SLam, our embedding places source-language values into target-language store cells in order to emulate the sealing behavior of security types. A slight complication arises in the translation of ref types since our language associates a security level with ref cells, but $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ does not. In value-oriented security languages, the contents of ref cells have a security level, however. So we use the security level of the contents as the security level of the ref cell itself in our translation.

Function types are translated into functions types that return monadic types. In a $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ function of type $s \xrightarrow{\mathsf{pc}} s'$ the program counter annotation pc is a conservative approximation of the information gained by the body of the function. Therefore, values written by the body must have security level at least pc. Thus, the corresponding writes in the translation must have write level at least pc. Consequently, the corresponding translated type for a function is $\overline{s} \rightarrow \bigcirc_{(\bot,\mathsf{pc})}\overline{s'}$. The type encoding is summarized below:

$$
\begin{array}{rcl}
\overline{(t,a)} & = & \mathsf{refr}_a\,\overline{t} \\
\\
\overline{1} & = & 1 \\
\overline{\mathsf{bool}} & = & \mathsf{bool} \\
\overline{\mathsf{ref}\ (t,a)} & = & \mathsf{ref}_a\,\overline{(t,a)} \\
\overline{s_1 \xrightarrow{\mathsf{pc}} s_2} & = & \overline{s_1} \rightarrow \bigcirc_{(\bot,\mathsf{pc})}\overline{s_2}
\end{array}
$$

The encoding for SLam expressions is given by a pair of judgments $\Sigma; \Gamma \vdash bv : t \Rightarrow M$ and $\Sigma; \Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E$, shown in Figures 10 and 11. We assume that the metavariable y stands for variables in our calculus that do not appear in $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ programs.

Type-correctness In order to show that our proposed encoding preserves typing, we first have to establish the following facts. The first shows that our encoding judgments agree with $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ typing judgments; the second shows that the encoding preserves subtyping.

Lemma 6.5. *1. If* $\Sigma; \Gamma \vdash bv : t \Rightarrow M$ *then* $\Sigma; \Gamma \vdash bv : t$

2. If $\Sigma; \Gamma \vdash e : s \Rightarrow E$ *then* $\Sigma; \Gamma \vdash e : s$

Proof. By induction on the given derivations. Observe that in each case, the rules of the encoding judgment have the same premises as the corresponding typing rules. □

$\boxed{\Sigma;\Gamma \vdash bv : t \Rightarrow M}$

$$\frac{}{\Sigma;\Gamma \vdash * : 1 \Rightarrow *}$$

$$\frac{}{\Sigma;\Gamma \vdash \mathsf{true} : \mathsf{bool} \Rightarrow \mathsf{true}}$$

$$\frac{}{\Sigma;\Gamma \vdash \mathsf{false} : \mathsf{bool} \Rightarrow \mathsf{false}}$$

$$\frac{}{\Sigma;\Gamma \vdash \ell : \Sigma(\ell) \Rightarrow \ell}$$

$$\frac{\Sigma;\Gamma, x : s_1[\mathsf{pc}] \vdash e : s_2 \Rightarrow E}{\Sigma;\Gamma \vdash \lambda[\mathsf{pc}]x : s_1.e : s_1 \xrightarrow{\mathsf{pc}} s_2 \Rightarrow \lambda x : \overline{s_1}.\mathsf{val}\ E}$$

$$\frac{\Sigma;\Gamma \vdash bv : t' \Rightarrow E \quad \vdash t' \leq t}{\Sigma;\Gamma \vdash bv : t \Rightarrow E}$$

Figure 10: $\lambda_{\text{SEC}}^{\text{REF}}$ base value encoding.

Lemma 6.6 (Subtyping Translation). *1. If $\vdash t' \leq t$ then $\vdash \overline{t'} \leq \overline{t}$*

2. If $\vdash s' \leq s$ then $\vdash \overline{s'} \leq \overline{s}$

Proof. Both parts simultaneously, by induction on the given derivation. □

Finally, we need to extend our type-translation to store types

$$\overline{\Sigma, \ell : s} = \overline{\Sigma}, \ell : \overline{s}$$

We are now ready to show type-correctness.

Theorem 6.7 (Well-typed Translation). *1. If $\Sigma;\Gamma \vdash bv : t \Rightarrow M$ then $\overline{\Sigma};\overline{\Gamma} \vdash M : \overline{t}$*

2. If $\Sigma;\Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E$ then $\overline{\Sigma};\overline{\Gamma} \vdash E \div_{(\bot,\mathsf{pc})} \overline{s}$

The proof is by simultaneous induction on the given derivations. The full proof is available in Appendix E.

Non-interference As with the purely-functional language, we show that our encoding respects an instance of the non-interference theorem for $\lambda_{\text{SEC}}^{\text{REF}}$.

Theorem 6.8 ($\lambda_{\text{SEC}}^{\text{REF}}$ non-interference). *Suppose $\Sigma_0; x : (t, a)[b] \vdash f : (\mathsf{bool}, b) \Rightarrow F$ where $a \not\sqsubseteq b$, and suppose that H, Σ are such that $\Sigma \supseteq \overline{\Sigma_0}$, and $\vdash H : \Sigma$. If $\Sigma; \cdot \vdash \ell_i : \mathsf{refr}_a\,\overline{t}$ for $i = 1, 2$ and if there exist $H_1, H_2, \Sigma_1, \Sigma_2, V_1, V_2$ such that*

$$(H', \Sigma', F[\ell_i/x]) \to^* (H_i, \Sigma_i, [V_i])$$

$$\boxed{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E}$$

$$\frac{}{\Sigma;\Gamma, x : s[\mathsf{pc}] \vdash x : s \sqcup \mathsf{pc} \Rightarrow [x]}$$

$$\frac{\Sigma;\Gamma \vdash bv : t \Rightarrow M}{\Sigma;\Gamma[\mathsf{pc}] \vdash bv_a : (t, a \sqcup \mathsf{pc}) \Rightarrow \mathsf{ref}_{a\sqcup\mathsf{pc}}\,(M : \overline{t})}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{bool}, a) \Rightarrow E_1 \quad \Sigma;\Gamma[\mathsf{pc} \sqcup a] \vdash e_2 : s \Rightarrow E_2 \quad \Sigma;\Gamma[\mathsf{pc} \sqcup a] \vdash e_3 : s \Rightarrow E_3}{\Sigma;\Gamma[\mathsf{pc}] \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : s \Rightarrow \begin{array}{l}\mathsf{let}\ y = E_1\ \mathsf{in} \\ \mathsf{let}\ y' = !y\ \mathsf{in} \\ \ \ \mathsf{run}\ \ \ \mathsf{if}\ y' \\ \qquad\quad \mathsf{then\ val}\ E_2 \\ \qquad\quad \mathsf{else\ val}\ E_3\end{array}}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (s' \xrightarrow{\mathsf{pc}'} s, a) \Rightarrow E_1 \quad \Sigma;\Gamma[\mathsf{pc}] \vdash e_2 : s' \Rightarrow E_2 \quad \mathsf{pc} \sqcup a \sqsubseteq \mathsf{pc}'}{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 e_2 : s \sqcup a \Rightarrow \begin{array}{l}\mathsf{let}\ y_1 = E_1\ \mathsf{in} \\ \mathsf{let}\ y_2 = E_2\ \mathsf{in} \\ \mathsf{let}\ y_1' = !y_1\ \mathsf{in} \\ \mathsf{run}\ (y_1' y_2)\end{array}}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E}{\Sigma;\Gamma[\mathsf{pc}] \vdash \mathsf{ref}\ (e : (t, a)) : (\mathsf{ref}\ (t, a), \mathsf{pc}) \Rightarrow \begin{array}{l}\mathsf{let}\ y = E\ \mathsf{in} \\ \mathsf{ref}_a\,(y : \overline{(t, a)})\end{array}}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e : (\mathsf{ref}\ s, a) \Rightarrow E}{\Sigma;\Gamma[\mathsf{pc}] \vdash !e : s \sqcup a \Rightarrow \mathsf{let}\ y = E\ \mathsf{in}\ \mathsf{let}\ y' = !y\ \mathsf{in}\ !y'}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{ref}\ (t, b), a) \Rightarrow E_1 \quad \Sigma;\Gamma[\mathsf{pc}] \vdash e_2 : (t, b) \Rightarrow E_2 \quad a \sqsubseteq b}{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 := e_2 : (1, \mathsf{pc}) \Rightarrow \begin{array}{l}\mathsf{let}\ y_1 = E_1\ \mathsf{in} \\ \mathsf{let}\ y_2 = E_2\ \mathsf{in} \\ \mathsf{let}\ y_1' = !y_1\ \mathsf{in} \\ \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in} \\ \mathsf{ref}_{\mathsf{pc}}\,(* : 1)\end{array}}$$

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s_1 \Rightarrow E \quad \vdash s_1 \leq s_2}{\Sigma;\Gamma[\mathsf{pc}] \vdash e : s_2 \Rightarrow E}$$

Figure 11: $\lambda_{\mathrm{SEC}}^{\mathrm{REF}}$ expression encoding.

for $i = 1, 2$ then $V_i = \ell'_i$ and $H_1(\ell'_1) = H_2(\ell'_2)$ as booleans.

Proof.

1. By the type-correctness of the translation, and by heap extension, $\Sigma; x : \mathsf{refr}_a\,\overline{t} \vdash F \div_{(\bot,b)} \mathsf{refr}_b\,\mathsf{bool}$

2. $$\frac{\Sigma; x : \mathsf{refr}_a\,\overline{t} \vdash F \div_{(\bot,b)} \mathsf{refr}_b\,\mathsf{bool}}{\Sigma; x : \mathsf{refr}_a\,\overline{t} \vdash F \div_{(b,b)} \mathsf{refr}_b\,\mathsf{bool}}$$

3. $$\frac{\Sigma; \cdot \vdash \ell_i : \mathsf{refr}_a\,\overline{t} \text{ for } i = 1, 2 \quad a \not\sqsubseteq b \quad \vdash \mathsf{refr}_a\,\overline{t} \nearrow a}{\Sigma; \Sigma; \cdot \vdash \ell_1 \approx_b \ell_2 : \mathsf{refr}_a\,\overline{t}} \quad (39)$$

4. Therefore, by non-interference,
$$\vdash (H_1, \Sigma_1, [V_1]) \approx_b (H_2, \Sigma_2, [V_2]) \div_{(b,b)} \mathsf{refr}_b\,\mathsf{bool}$$

5. By inversion,
$$\Sigma_1; \Sigma_2; \cdot \vdash V_1 \approx_b V_2 : \mathsf{refr}_{b'}\,\mathsf{bool}$$
with $b' \sqsubseteq b$

6. By regularity, for $i = 1, 2$,
$$\Sigma_i; \cdot \vdash V_i : \mathsf{refr}_{b'}\,\mathsf{bool}$$

7. By Canonical Forms, for $i = 1, 2$, $V_i = \ell'_i \in \mathrm{dom}(\Sigma_i)$

8. Since $\mathsf{Level}(\ell'_i) = b' \sqsubseteq b$, $\ell'_1 = \ell'_2 \in \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap \downarrow(b)$

9. Therefore, $\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell'_1) \approx_b H_2(\ell'_2) : \mathsf{bool}$

10. Since $\vdash \mathsf{bool} \nearrow \bot$, by equivalent term inversion, $H_1(\ell'_1) = H_2(\ell'_2)$.

□

An encoding into our account is useful not only to show that our language is at least as powerful as value-oriented secure languages, but it may also be used to guide the design of such languages. We explore one alternative design for $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$.

As mentioned above (in Lemma 6.4), $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ typing judgments maintain the invariant that in well-typed programs the program counter security level is a lower bound on the security levels of the values computed by the program. The present formulation of $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ leaves the invariant implicit; it is a consequence of the typing rule for base values and variables:

$$\frac{\Sigma; \Gamma \vdash bv : t}{\Sigma; \Gamma[\mathsf{pc}] \vdash bv_a : (t, a \sqcup \mathsf{pc})} \qquad \frac{}{\Sigma; \Gamma, x : s[\mathsf{pc}] \vdash x : s \sqcup \mathsf{pc}}$$

An alternative formulation would instead check that pc is below the security level of any values propagated by the computation. The proof of type-correctness of the translation into our language may be used as a guide: anywhere that Lemma 6.4 is invoked in the proof, we may instead alter the typing rule to check the pc explicitly and rely on subsumption to raise the security level of values where required:

$$\frac{\Sigma; \Gamma \vdash bv : t}{\Sigma; \Gamma[\mathsf{pc}] \vdash bv_a : (t, a)} \qquad \frac{}{\Sigma; \Gamma, x : s[\mathsf{pc}] \vdash x : s}$$

$$\frac{\Sigma; \Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{bool}, a) \quad \Sigma; \Gamma[\mathsf{pc} \sqcup a] \vdash e_2 : s \quad \Sigma; \Gamma[\mathsf{pc} \sqcup a] \vdash e_3 : s}{\Sigma; \Gamma[\mathsf{pc}] \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_2 : s \sqcup a}$$

$$\frac{\Sigma; \Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{ref}\ (t, b), a) \quad \Sigma; \Gamma[\mathsf{pc}] \vdash e_2 : (t, b) \quad a \sqcup \mathsf{pc} \sqsubseteq b}{\Sigma; \Gamma[\mathsf{pc}] \vdash e_1 := e_2 : (\mathsf{bool}, \bot)}$$

$$\frac{\Sigma; \Gamma[\mathsf{pc}] \vdash e : s}{\Sigma; \Gamma[\mathsf{pc}] \vdash \mathsf{ref}\ (e : s) : (\mathsf{ref}\ s, \bot)}$$

Note in particular that the alternate rule for if-expressions now has a similar form to the corresponding rule in SLam. Only writes depend critically on the program counter security level, since it is only through side-effects that information gained via control flow can escape non-locally (*i.e.,* not in the result of an expression).

7 Related Work

There is a large body of existing work on type systems for secure information flow. Volpano, Smith and Irvine [14] first showed how to formulate an information flow analysis as a type system. An excellent survey by Sabelfeld and Myers [13] outlines the key ideas in the design of secure programming languages.

Prior work on secure languages with imperative features, such as Pottier and Simonet's work on core-ML [11, 12] take the side-effect view of computations: a term of any type A may have a side-effect. In contrast, we have taken a monadic view of computation: only expressions may have an effect.

Our account is most related to the Dependency Core Calculus of Abadi, Banerjee, *et al.* [1]. Like our language, DCC uses a family of monads to reason about information flow. However in DCC, terms of monadic type are used to seal up values at a security level. In our account, monads are used in a more traditional role as a means of threading state through a program.

Central to DCC is the notion of *protectedness* of a type at a security level. If T is protected at a then T is at least as secure as a. This is closely related to our notion of informativeness. (Also, a similar notion called "tampering levels" appears in Honda and Yoshida [4].)

When viewed through the lens of the encoding of SLam, the two relations serve the same purpose, ensuring that a computation's output is at least as secure as its inputs. In DCC, this is done directly. In our account, this occurs indirectly: to access a value carrying information only at a particular level, a computation must adopt a read level at least as high. (However, our account also offers the facility — not employed in the SLam embedding — not to seal all computations' return values in order to obtain a $\perp$ effective read level).

The definitions of protectedness and informativeness are the same on the standard type operators, but do not include the idiosyncratic cases: our language has no analog of DCC's monad, nor does DCC contain references or a traditional (*i.e.,* effects-oriented) monad. Moreover, if it did, we conjecture that DCC's definition for these would be somewhat different from ours.

Nevertheless, the similarity between the two suggests that our account might be profitably combined with DCC to produce a language capable of expressing security in both value-oriented and store-oriented fashions.

8 Conclusion

We give an account of secure information flow in the context of a higher-order language with mutable state. Moreover, motivated by the logical underpinnings of computation, we arrive at an store-oriented approach to security. Rather than sealing values at a security level, we instead associate security with the store. A family of monadic types is used to keep track of the effects of computations. To account for upcalls, we classify the informativeness of types at particular security levels.

Since we treat terms apart from the effectful expressions, our approach can straightforwardly encompass additional type constructors. The question of how to account for additional effects requires further work. From the point of view of non-interference, effects introduce the possibility of different behavior from seemingly related expressions. We expect that by further refining the monadic type to restrict the behavior of related terms, we may be able to account for effects such as I/O or non-local control transfers.

Our formulation of the monadic language is in the style of Pfenning and Davies [9]. One avenue of future work is to study whether there is a formulation of information flow in a modal logic that decomposed our monad into the possibility and necessity modalities.

Several questions remain about our language. In the case that the lattice of security levels $\mathcal{L}$ is finite, it is obvious that the informativeness judgment $\vdash A \nearrow a$ is decidable. Although it is reasonable to assume that in a practical secure programming system, there would only be a finite number of security levels, it is nonetheless interesting to know whether $\vdash A \nearrow a$ is decidable even in the general case where $\mathcal{L}$ may be infinite.

A general open problem in the area of secure programming languages is how to devise a type system for a language with *declassification* operations. Declassification occurs when a low-security computation makes use of a high-security value, but in a way such that the information gained from the high-security value is deemed an acceptable leak. Recently, Zdancewic and Myers [17] show how to characterize so-called *robust declassification* in programs such that an attacker may observe the declassified values, but may not exploit them to gain additional high-security information. Zdancewic [16] then gives a type system for robust declassification. Since declassification is fundamentally an operation, we conjecture that our store-oriented viewpoint could be meshed with Zdancewic and Myers to provide a logic of declassification.

References

[1] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Twenty-Sixth ACM Symposium on Principles of Programming Languages*, pages 147–160, San Antonio, Texas, Jan. 1999.

[2] P. N. Benton, G. M. Bierman, and V. C. V. de Paiva. Computational types from a logical perspective. *Journal of Functional Programming*, 8(2):177–193, 1998.

[3] N. Heintze and J. G. Riecke. The SLam calculus: Programming with secerecy and integrity. In *Twenty-Fifth ACM Symposium on Principles of Programming Languages*, pages 365 – 377, San Diego, California, Jan. 1998.

[4] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Twenty-Ninth ACM Symposium on Principles of Programming Languages*, pages 81–92, Jan. 2002.

[5] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797 – 821, Oct. 1980.

[6] E. Moggi. Computational lambda-calculus and monads. In *Fourth IEEE Symposium on Logic in Computer Science*, pages 14–23, 1989.

[7] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[8] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Twenty-Sixth ACM Symposium on Principles of Programming Languages*, pages 228–241, San Antonio, Texas, Jan. 1999.

[9] F. Pfenning and R. Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11(4):511–540, 2001.

[10] F. Pottier and S. Conchon. Informaiton flow inference for free. In *2000 ACM International Conference on Functional Programming*, pages 46–57, Montréal, Canada, Sept. 2000.

[11] F. Pottier and V. Simonet. Information flow inference for ML. In *Twenty-Ninth ACM Symposium on Principles of Programming Languages*, Jan. 2002.

[12] F. Pottier and V. Simonet. Information flow inference for ML. *ACM Transactions on Programming Languages and Systems*, 25(1):117–158, Jan. 2003.

[13] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5 – 19, Jan. 2003. special issue on Formal Methods in Security.

[14] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[15] S. Zdancewic. *Programming Languages for Information Security*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 2002.

[16] S. Zdancewic. A type system for robust declassification. In *Nineteenth Mathematical Foundations of Programming Semantics*, Electronic Notes in Theoretical Computer Science, Mar. 2003.

[17] S. Zdancewic and A. C. Myers. Robust declassification. In *Fourteenth IEEE Computer Security Foundations Workshop*, pages 15 – 23, Cape Brenton, Nova Scotia, Canada, 2001.

[18] S. Zdancewic and A. C. Myers. Secure information flow and CPS. In *Tenth European Symposium on Programming*, volume 2028 of *Lecture Notes in Computer Science*, pages 46 – 61. Springer-Verlag, Apr. 2001.

[19] S. Zdancewic and A. C. Myers. Secure information flow via linear continuations. *Higher Order and Symbolic Computation*, 15(2-3):209–234, Sept. 2002.

A Judgments

A.1 Informativeness judgment rules

$\boxed{\vdash A \nearrow a}$

$$\frac{}{\vdash A \nearrow \bot}\ (1)$$

$$\frac{}{\vdash 1 \nearrow a}\ (2)$$

$$\frac{\vdash B \nearrow a}{\vdash A \to B \nearrow a}\ (3)$$

$$\frac{\vdash A \nearrow a}{\vdash \bigcirc_{(r,w)} A \nearrow w \sqcap a} \ (4)$$

$$\frac{}{\vdash \mathsf{ref}_b\, A \nearrow b} \ (5)$$

$$\frac{\vdash A \nearrow a}{\vdash \mathsf{ref}_b\, A \nearrow a} \ (6)$$

$$\frac{\vdash A \nearrow a}{\vdash \mathsf{refr}_b\, A \nearrow a} \ (7)$$

$$\frac{}{\vdash \mathsf{refr}_b\, A \nearrow b} \ (8)$$

$$\frac{}{\vdash \mathsf{refw}_a\, A \nearrow a} \ (9)$$

$$\frac{\vdash A \nearrow a \quad b \sqsubseteq a}{\vdash A \nearrow b} \ (10)$$

$$\frac{\vdash A \nearrow a \quad \vdash A \nearrow b}{\vdash A \nearrow a \sqcup b} \ (11)$$

A.2 Typing judgment rules

$\boxed{\vdash A \leq B}$

$$\frac{}{\vdash A \leq A} \ (12)$$

$$\frac{\vdash A \leq A' \quad \vdash B' \leq B}{\vdash A' \to B' \leq A \to B} \ (13)$$

$$\frac{\vdash A \leq B \quad o \preceq o'}{\vdash \bigcirc_o A \leq \bigcirc_{o'} B} \ (14)$$

$$\frac{\vdash A \leq B \quad a \sqsubseteq b}{\vdash \mathsf{ref}_a\, A \leq \mathsf{refr}_{a'}\, B} \ (15)$$

$$\frac{\vdash B \leq A \quad b \sqsubseteq a}{\vdash \mathsf{ref}_a\, A \leq \mathsf{refw}_{a'}\, B} \ (16)$$

$$\frac{\vdash A \leq B \quad a \sqsubseteq b}{\vdash \mathsf{refr}_a\, A \leq \mathsf{refr}_b\, B} \ (17)$$

$$\frac{\vdash B \leq A \quad b \sqsubseteq a}{\vdash \mathsf{refw}_a\, A \leq \mathsf{refw}_b\, B} \ (18)$$

$\boxed{\Sigma; \Gamma \vdash M : A}$

$$\frac{}{\Sigma; \Gamma \vdash * : 1} \ (19)$$

$$\frac{}{\Sigma; \Gamma \vdash x : \Gamma(x)} \ (20)$$

$$\frac{}{\Sigma; \Gamma \vdash \mathsf{true} : \mathsf{bool}} \ (21)$$

$$\frac{}{\Sigma; \Gamma \vdash \mathsf{false} : \mathsf{bool}} \ (22)$$

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{bool} \quad \Sigma; \Gamma \vdash N_1 : A \quad \Sigma; \Gamma \vdash N_2 : A}{\Sigma; \Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : A} \ (23)$$

$$\frac{}{\Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell)} \ (24)$$

$$\frac{\Sigma; \Gamma, x : A \vdash M : B}{\Sigma; \Gamma \vdash \lambda x : A.M : A \to B} \ (25)$$

$$\frac{\Sigma; \Gamma \vdash M : A \to B \quad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash MN : B} \ (26)$$

$$\frac{\Sigma; \Gamma \vdash E \div_o A}{\Sigma; \Gamma \vdash \mathsf{val}\ E : \bigcirc_o A} \ (27)$$

$$\frac{\Sigma; \Gamma \vdash M : A \quad \vdash A \leq B}{\Sigma; \Gamma \vdash M : B} \ (28)$$

$\boxed{\Sigma; \Gamma \vdash E \div_o A}$

$$\frac{\Sigma; \Gamma \vdash M : A}{\Sigma; \Gamma \vdash [M] \div_{(\bot,\top)} A} \ (29)$$

$$\frac{\Sigma; \Gamma \vdash M : \bigcirc_o A \quad \Sigma; \Gamma, x : A \vdash E \div_o B}{\Sigma; \Gamma \vdash \mathsf{let\ val}\ x = M\ \mathsf{in}\ E \div_o B} \ (30)$$

$$\frac{\Sigma; \Gamma \vdash M : A}{\Sigma; \Gamma \vdash \mathsf{ref}_a\, (M : A) \div_{(\bot,\top)} \mathsf{ref}_a\, A} \ (31)$$

$$\frac{\Sigma;\Gamma \vdash M : \mathsf{refr}_a\, A}{\Sigma;\Gamma \vdash !M \div_{(a,\top)} A} \quad (32)$$

$$\frac{\Sigma;\Gamma \vdash M : \mathsf{refw}_a\, A \quad \Sigma;\Gamma \vdash N : A}{\Sigma;\Gamma \vdash M := N \div_{(\bot,a)} 1} \quad (33)$$

$$\frac{\Sigma;\Gamma \vdash E \div_{o'} A \quad o' \preceq o}{\Sigma;\Gamma \vdash E \div_o A} \quad (34)$$

$$\frac{\Sigma;\Gamma \vdash E \div_{(r,w)} A \quad \vdash A \nearrow r}{\Sigma;\Gamma \vdash E \div_{(\bot,w)} A} \quad (35)$$

$$\frac{\Sigma;\Gamma \vdash E \div_o B \quad \vdash B \leq C}{\Sigma;\Gamma \vdash E \div_o C} \quad (36)$$

$\boxed{\vdash H : \Sigma}$

$$\frac{\mathrm{dom}(\Sigma) = \{\ell_1, \ldots, \ell_n\} \quad \Sigma; \cdot \vdash V_i : \Sigma(\ell_i) \ \text{ for } 1 \leq i \leq n}{\vdash \{\ell_1 \mapsto V_1, \ldots, \ell_n \mapsto V_n\} : \Sigma} \quad (37)$$

$\boxed{\vdash S \div_o A}$

$$\frac{\vdash H : \Sigma \quad \Sigma; \cdot \vdash E \div_o A}{\vdash (H, \Sigma, E) \div_o A} \quad (38)$$

Derived typing rules for syntactic sugar

$$\frac{\Sigma;\Gamma \vdash E \div_o A \quad \Sigma;\Gamma, x : A \vdash F \div_o C}{\Sigma;\Gamma \vdash \mathsf{let}\ x = E\ \mathsf{in}\ F \div_o C}$$

$$\frac{\Sigma;\Gamma \vdash M : \bigcirc_o C}{\Sigma;\Gamma \vdash \mathsf{run}\ M \div_o C}$$

A.3 Equivalent view judgments rules

$\boxed{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : A}$

$$\frac{\vdash A \nearrow a \quad a \not\sqsubseteq \zeta \quad \Sigma_1;\Gamma \vdash V_1 : A \quad \Sigma_2;\Gamma \vdash V_2 : A}{\Sigma_1;\Sigma_2;\Gamma \vdash V_1 \approx_\zeta V_2 : A} \quad (39)$$

$$\frac{}{\Sigma_1;\Sigma_2;\Gamma \vdash * \approx_\zeta * : 1} \quad (40)$$

$$\frac{}{\Sigma_1;\Sigma_2;\Gamma \vdash x \approx_\zeta x : \Gamma(x)} \quad (41)$$

$$\frac{}{\Sigma_1;\Sigma_2;\Gamma \vdash \mathsf{true} \approx_\zeta \mathsf{true} : \mathsf{bool}} \quad (42)$$

$$\frac{}{\Sigma_1;\Sigma_2;\Gamma \vdash \mathsf{false} \approx_\zeta \mathsf{false} : \mathsf{bool}} \quad (43)$$

$$\frac{\begin{array}{l}\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : \mathsf{bool} \\ \Sigma_1;\Sigma_2;\Gamma \vdash N_1 \approx_\zeta N_2 : A \\ \Sigma_1;\Sigma_2;\Gamma \vdash P_1 \approx_\zeta P_2 : A\end{array}}{\Sigma_1;\Sigma_2;\Gamma \vdash \begin{array}{ll}\mathsf{if}\ M_1\ \mathsf{then}\ N_1\ \mathsf{else}\ P_1 & \approx_\zeta \\ \mathsf{if}\ M_2\ \mathsf{then}\ N_2\ \mathsf{else}\ P_2 & : A\end{array}} \quad (44)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma, x : A \vdash M_1 \approx_\zeta M_2 : B}{\Sigma_1;\Sigma_2;\Gamma \vdash \lambda x : A.M_1 \approx_\zeta \lambda x : A.M_2 : A \to B} \quad (45)$$

$$\frac{\begin{array}{l}\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : A \to B \\ \Sigma_1;\Sigma_2;\Gamma \vdash N_1 \approx_\zeta N_2 : A\end{array}}{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 N_1 \approx_\zeta M_2 N_2 : B} \quad (46)$$

$$\frac{\mathsf{Level}(\ell) \sqsubseteq \zeta \quad \Sigma_1(\ell) = \Sigma_2(\ell)}{\Sigma_1;\Sigma_2;\Gamma \vdash \ell \approx_\zeta \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\,\Sigma_1(\ell)} \quad (47)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_o A}{\Sigma_1;\Sigma_2;\Gamma \vdash \mathsf{val}\ E_1 \approx_\zeta \mathsf{val}\ E_2 : \bigcirc_o A} \quad (48)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : A \quad \vdash A \leq B}{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : B} \quad (49)$$

$\boxed{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_o C}$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_{o'} C \quad o' \preceq o}{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_o C} \quad (50)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_{(r,w)} C \quad \vdash C \nearrow r}{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_{(\bot,w)} C} \quad (51)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_o B \quad \vdash B \leq C}{\Sigma_1;\Sigma_2;\Gamma \vdash E_1 \approx_\zeta E_2 \div_o C} \quad (52)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : C}{\Sigma_1;\Sigma_2;\Gamma \vdash [M_1] \approx_\zeta [M_2] \div_{(\bot,\top)} C} \quad (53)$$

$$\frac{\begin{array}{c}\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : \bigcirc_o A \\ \Sigma_1;\Sigma_2;\Gamma, x : A \vdash E_1 \approx_\zeta E_2 \div_o C\end{array}}{\Sigma_1;\Sigma_2;\Gamma \vdash \begin{array}{ll}\mathsf{let\ val}\ x = M_1\ \mathsf{in}\ E_1 & \approx_\zeta \\ \mathsf{let\ val}\ x = M_2\ \mathsf{in}\ E_2 & \div_o C\end{array}} \quad (54)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : A}{\Sigma_1;\Sigma_2;\Gamma \vdash \begin{array}{ll}\mathsf{ref}_a\,(M_1 : A) & \approx_\zeta \\ \mathsf{ref}_a\,(M_2 : A) & \div_{(\bot,\top)} \mathsf{ref}_a\,A\end{array}} \quad (55)$$

$$\frac{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : \mathsf{refr}_a\,A}{\Sigma_1;\Sigma_2;\Gamma \vdash !M_1 \approx_\zeta !M_2 \div_{(a,\top)} A} \quad (56)$$

$$\frac{\begin{array}{c}\Sigma_1;\Sigma_2;\Gamma \vdash M_1 \approx_\zeta M_2 : \mathsf{refw}_a\,A \\ \Sigma_1;\Sigma_2;\Gamma \vdash N_1 \approx_\zeta N_2 : A\end{array}}{\Sigma_1;\Sigma_2;\Gamma \vdash M_1 := N_1 \approx_\zeta M_2 := N_2 \div_{(\bot,a)} 1} \quad (57)$$

$\boxed{\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)}$

$$\frac{\begin{array}{l}\vdash H_i : \Sigma_i \text{ for } i = 1,2 \\ \Sigma_1 \upharpoonright U = \Sigma_2 \upharpoonright U \\ \Sigma_1;\Sigma_2;\cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : \Sigma_1(\ell) \text{ for all } \ell \in U\end{array}}{\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)} \quad (58)$$

$\boxed{\vdash S_1 \approx_\zeta S_2 \div_o C}$

$$\frac{\begin{array}{l}\vdash (H_1 : \Sigma_1) \approx_\zeta^{\mathrm{dom}(\Sigma_1)\cap\mathrm{dom}(\Sigma_2)\cap\downarrow(\zeta)} (H_2 : \Sigma_2) \\ \Sigma_1;\Sigma_2;\cdot \vdash E_1 \approx_\zeta E_2 \div_o C\end{array}}{\vdash (H_1, \Sigma_1, E_1) \approx_\zeta (H_2, \Sigma_2, E_2) \div_o C} \quad (59)$$

B Evaluation Rules

$$\boxed{M \to M'}$$

$$\frac{M \to M'}{\mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \to \mathsf{if}\ M'\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2} \ \textsc{If1}$$

$$\frac{}{\mathsf{if\ true\ then}\ N_1\ \mathsf{else}\ N_2 \to N_1} \ \textsc{IfTrue}$$

$$\frac{}{\mathsf{if\ false\ then}\ N_1\ \mathsf{else}\ N_2 \to N_2} \ \textsc{IfFalse}$$

$$\frac{M \rightarrow M'}{MN \rightarrow M'N} \text{ App1}$$

$$\frac{N \rightarrow N'}{VN \rightarrow VN'} \text{ App2}$$

$$\frac{}{(\lambda x : A.M)V \rightarrow M[V/x]} \text{ App}$$

$$\boxed{S \rightarrow S'}$$

$$\frac{M \rightarrow M'}{(H, \Sigma, [M]) \rightarrow (H, \Sigma, [M'])} \text{ Ret1}$$

$$\frac{M \rightarrow M'}{(H, \Sigma, \mathsf{let\ val}\ x = M\ \mathsf{in}\ E) \rightarrow (H, \Sigma, \mathsf{let\ val}\ x = M'\ \mathsf{in}\ E)} \text{ Letval1}$$

$$\frac{(H, \Sigma, E) \rightarrow (H', \Sigma', E')}{(H, \Sigma, \mathsf{let\ val}\ x = \mathsf{val}\ E\ \mathsf{in}\ F) \rightarrow (H', \Sigma', \mathsf{let\ val}\ x = \mathsf{val}\ E'\ \mathsf{in}\ F)} \text{ Letvalval}$$

$$\frac{}{(H, \Sigma, \mathsf{let\ val}\ x = \mathsf{val}\ [V]\ \mathsf{in}\ E) \rightarrow (H, \Sigma, E[V/x])} \text{ Letval}$$

$$\frac{M \rightarrow M'}{(H, \Sigma, \mathsf{ref}_a\ (M : A)) \rightarrow (H, \Sigma, \mathsf{ref}_a\ (M' : A))} \text{ Ref1}$$

$$\frac{\ell \notin \mathrm{dom}(H) \quad \mathsf{Level}(\ell) = a}{(H, \Sigma, \mathsf{ref}_a\ (V : A)) \rightarrow (H\{\ell \mapsto V\}, \Sigma\{\ell : A\}, [\ell])} \text{ Ref}$$

$$\frac{M \rightarrow M'}{(H, \Sigma, !M) \rightarrow (H, \Sigma, !M')} \text{ Bang1}$$

$$\frac{}{(H, \Sigma, !\ell) \rightarrow (H, \Sigma, [H(\ell)])} \text{ Bang}$$

$$\frac{M \to M'}{(H, \Sigma, M := N) \to (H, \Sigma, M' := N)} \text{ Assn1}$$

$$\frac{N \to N'}{(H, \Sigma, V := N) \to (H, \Sigma, V := N')} \text{ Assn2}$$

$$\frac{\ell \in \mathrm{dom}(H)}{(H, \Sigma, \ell := V) \to (H\{\ell \mapsto V\}, \Sigma, [*])} \text{ Assn}$$

C Proofs

Subterm/Subexpression Termination

- If $(H, \Sigma, E) \downarrow$ in n steps, then
  1. if $E = [M]$ then $M \to^n V$
  2. if $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$ then $M \to^k \mathsf{val}\ E'$,

     $$(H, \Sigma, E') \downarrow \text{ in } m \text{ steps}$$

     and $k + m < n$
  3. if $E = \mathsf{ref}_a\,(M : A)$ then $M \to^k V$ and $k < n$
  4. if $E = !M$ then $M \to^k V$ and $k < n$
  5. if $E = M := N$ then $M \to^k V_1$, $N \to^m V_2$ and $k + m < n$
- If $M \to^n V$ then
  1. If $M = N_1 N_2$, then $N_1 \to^k V_1$ and $V_1 N_2 \to^m V_1 V_2$ and $k + m < n$
  2. If $M = \mathsf{if}\ N_1\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$ then $N_1 \to^k V_1$ and $k < n$

Proof. by induction on the number of steps in the evaluation relation.
Part (1)
If $n = 0$, evidently $E = [V]$, so $V \to^0 V$.
If $n > 0$, then $(H, \Sigma, E) \to S$ and $S \downarrow$ in $n - 1$ steps. Proceed by cases on the last rule used in $(H, \Sigma, E) \to S$

- Case Ret1: $E = [M]$, $S = (H, \Sigma, [M'])$, $M \to M'$.
  By IH, $M' \to^{n-1} V$, so $M \to^n V$

- Case Letval1: $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$, $S = (H, \Sigma, \mathsf{let\ val}\ x = M'\ \mathsf{in}\ F)$, $M \to M'$

  By IH, $M' \to^k \mathsf{val}\ E'$, $(H, \Sigma, E') \downarrow$ in m steps, and $k + m < n - 1$. So $M' \to^{k+1} \mathsf{val}\ E'$, and $k + 1 + m < n$.

- Case Letvalval: $E = \mathsf{let\ val}\ x = \mathsf{val}\ E'\ \mathsf{in}\ F$, $S = (H', \Sigma', \mathsf{let\ val}\ x = \mathsf{val}\ E''\ \mathsf{in}\ F)$, $(H, \Sigma, E') \to (H', \Sigma', E'')$.

  By IH, $\mathsf{val}\ E'' \to^0 \mathsf{val}\ E''$, $(H', \Sigma', E'') \downarrow$ in m steps, $m < n - 1$. So $(H', \Sigma', E') \downarrow$ in $m + 1$ steps, $m + 1 < n$.

- Case Letval: $E = \mathsf{let\ val}\ x = \mathsf{val}\ [V]\ \mathsf{in}\ F$, $S = (H, \Sigma, F[V/x])$.

  Evidently $\mathsf{val}\ [V] \to^0 \mathsf{val}\ [V]$, and $(H, \Sigma, [V]) \to^0 (H, \Sigma, [V])$, and $0 < n$.

- Case Ref1: $E = \mathsf{ref}_a\,(M : A)$, $S = (H, \Sigma, \mathsf{ref}_a\,(M' : A))$, $M \to M'$.

  By IH, $M' \to^k V$, $k < n - 1$. So $M \to^{k+1} V$, and $k + 1 < n$.

- Case Ref: $E = \mathsf{ref}_a\,(V : A)$, $S = (H\{\ell \mapsto V\}, \Sigma\{\ell : A\}, [\ell])$, $\ell \notin \mathrm{dom}(H)$.

  Evidently $V \to^0 V$, $0 < n$.

- Cases Bang1, Bang: Similar to Ref1, Ref, respectively.

- Case Assn1, Assn2, Assn: Similar to previous cases.

Part (2)
If $n = 0$, evidently M is a value. Case is vacuously true.
If $n > 0$, then $M \to M'$ and $M' \to^{n-1} V$. Proceed by cases on the last rule used in $M \to M'$.

- Case App1: $M = N_1 N_2$, $M' = N_1' N_2$, $N_1 \to N_1'$

  By IH, $N_1' \to^k V_1$, $N_2 \to^m V_2$, and $k + m < n - 1$. So $N \to^{k+1} V_1$, and $k + 1 + m < n$.

- Case App2: Similar to preceding case.

- Case App: $M = (\lambda x : A.N)V_2$, $M' = N[V_2/x]$

  Evidently $(\lambda x : A.N) \to^0 (\lambda x : A.N)$, and $V_2 \to^0 V_2$ and $0 < n$.

- Case If1: $M = \mathsf{if}\ N_1\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$, $M' = \mathsf{if}\ N_1'\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$, $N_1 \to N_1'$

  By IH, $N_1' \to^k V_1$, $k < n - 1$. So $N_1 \to^{k+1} V_1$, and $k + 1 < n$.

- Cases IfTrue, IfFalse: Evident.

□

C.1 Type safety proof

C.1.1 Properties of informativeness and subtyping

Before we go on to prove type safety and non-interference, we take the time to prove several (standard) lemmas.

Lemma C.1 (Informativeness Inversion). *If* $\vdash A \nearrow a$ *and if*

- $A = \mathsf{bool}$ *then* $a = \bot$
- $A = B \to C$ *then* $\vdash C \nearrow a$
- $A = \bigcirc_{(r,w)} B$ *then* $a \sqsubseteq w \sqcap b$ *and* $\vdash B \nearrow b$
- $A = \mathsf{ref}_b\, B$ *then* $\vdash B \nearrow c$ *and* $a \sqsubseteq c \sqcup b$
- $A = \mathsf{refr}_b\, B$ *then* $\vdash B \nearrow c$ *and* $a \sqsubseteq c \sqcup b$
- $A = \mathsf{refw}_b\, B$ *then* $a \sqsubseteq b$

Proof. by induction on the given derivation. By cases on the last rule used.

- Case rule (1). For each case of A, immediate.
- Case rule (2). Vacuous.
- Case rule (3). Evidently $A = B \to C$, and $\vdash C \nearrow a$.
- Case rule (4). Evidently $A = \bigcirc_{(r,w)} B$ and $a = w \sqcap b$ where $\vdash B \nearrow b$.
- Case rule (5). Evidently $A = \mathsf{ref}_a\, B$. Let $c = \bot$, then $\vdash B \nearrow c$ and $a \sqsubseteq \bot \sqcup a$
- Case rule (6). Evidently $A = \mathsf{ref}_b\, B$ and $\vdash B \nearrow a$. Then $a \sqsubseteq a \sqcup b$.
- Case rules (7), (8). Similar to cases for rules (5), (6), respectively.
- Case rule (9). Evidently $A = \mathsf{refw}_a\, B$.
- Case rule (10). Evidently $\vdash A \nearrow a'$ and $a \sqsubseteq a'$. By subcases on the structure of A:
  - Subcase $A = 1$. Vacuous.
  - Subcase $A = \mathsf{bool}$. By IH, $a' = \bot$, so $a = \bot$.
  - Subcase $A = B \to C$. By IH, $\vdash C \nearrow a'$. By rule (10), $\vdash C \nearrow a$
  - Subcase $A = \bigcirc_{(r,w)} B$. By IH, $a' \sqsubseteq w \sqcap b$ and $\vdash B \nearrow b$. So evidently $a \sqsubseteq w \sqcap b$.
  - Subcase $A = \mathsf{ref}_b\, B$. By IH, $a' \sqsubseteq c \sqcup b$ and $\vdash B \nearrow c$. So evidently $a \sqsubseteq c \sqcup b$.
  - Subcase $A = \mathsf{refr}_b\, B$. Same as previous subcase.

– Subcase $A = \mathsf{refw}_b\, B$. By IH, $a' \sqsubseteq b$. So $a \sqsubseteq b$.

- Case rule (11). Evidently $\vdash A \nearrow a_1$ and $\vdash A \nearrow a_2$ and $a = a_1 \sqcup a_2$. By subcases on the structure of A:
  - Subcase $A = 1$. Vacuous.
  - Subcase $A = \mathsf{bool}$. By IH, $a_1 = a_2 = a = \bot$.
  - Subcase $A = B \to C$. By IH, $\vdash C \nearrow a_1$, $\vdash C \nearrow a_2$. By rule (10), $\vdash C \nearrow a$
  - Subcase $A = \bigcirc_{(r,w)} B$.
    1. By IH, $a_i \sqsubseteq w \sqcap b_i$ and $\vdash B \nearrow b_i$ for $i = 1, 2$.
    2. By rule (11), $\vdash B \nearrow b_1 \sqcup b_2$.
    3. $a_i \sqsubseteq w \sqcap b_i \sqsubseteq w \sqcap (b_1 \sqcup b_2)$ for $i = 1, 2$.
    4. So $a \sqsubseteq w \sqcap (b_1 \sqcup b_2)$.
- Subcase $A = \mathsf{ref}_b\, B$.
  1. By IH, $a_i \sqsubseteq c_i \sqcup b$, and $\vdash B \nearrow c_i$ for $i = 1, 2$
  2. $a_1 \sqcup a_2 \sqsubseteq c_1 \sqcup c_2 \sqcup b$
  3. By rule (11), $\vdash B \nearrow c_1 \sqcup c_2$
- Subcase $A = \mathsf{refr}_b\, B$. Similar to previous subcase
- Subcase $A = \mathsf{refw}_b\, B$. By IH, $a_i \sqsubseteq b$, so $a_1 \sqcup a_2 \sqsubseteq b$.

$\square$

Lemma C.2 (Subtyping Inversion). *If $\vdash A' \leq A$ and if*

- $A = 1$ *then* $A' = 1$
- $A = \mathsf{bool}$ *then* $A' = \mathsf{bool}$
- $A = B \to C$ *then* $A' = B' \to C'$ *and* $\vdash B \leq B'$ *and* $\vdash C' \leq C$
- $A = \mathsf{ref}_a\, B$ *then* $A' = \mathsf{ref}_a\, B$
- $A = \mathsf{refr}_a\, B$ *then*
  - *either* $A' = \mathsf{refr}_{a'}\, B'$ *with* $\vdash B' \leq B$ *and* $a' \sqsubseteq a$
  - *or* $A' = \mathsf{ref}_{a'}\, B'$ *with* $\vdash B' \leq B$ *and* $a' \sqsubseteq a$
- $A = \mathsf{refw}_a\, B$ *then*
  - *either* $A' = \mathsf{refw}_{a'}\, B'$ *with* $\vdash B \leq B'$ *and* $a \sqsubseteq a'$
  - *or* $A' = \mathsf{ref}_{a'}\, B'$ *with* $\vdash B \leq B'$ *and* $a \sqsubseteq a'$
- $A = \bigcirc_o B$ *then* $A' = \bigcirc_{o'} B'$ *and* $\vdash B' \leq B$ *and* $o' \preceq o$

and moreover, all the result derivations are subderivations of the given derivation.

Proof. by cases on the last rule used in the given derivation. Each case follows immediately from the rules. ◻

Lemma C.3 (Transitivity of subtyping). *The following rule is admissible*

$$\frac{\vdash A \leq B \quad \vdash B \leq C}{\vdash A \leq C}$$

Proof. by induction on the derivations.

By cases on $\vdash B \leq C$

- Case

  $$\frac{}{\vdash B \leq B} \ (12)$$

  By pattern matching $B = C$. Immediate

- Case

  $$\frac{\vdash C_1 \leq B_1 \quad \vdash B_2 \leq C_2}{\vdash B_1 \to B_2 \leq C_1 \to C_2} \ (13)$$

  1. By pattern matching $B = B_1 \to B_2$, $C = C_1 \to C_2$.
  2. By inversion, $A = A_1 \to A_2$ and $\vdash B_1 \leq A_1$ and $\vdash A_2 \leq B_2$
  3. By IH, $\vdash C_1 \leq A_1$ and $\vdash A_2 \leq C_2$
  4. By rule (13), $\vdash A \leq C$

- Case

  $$\frac{\vdash B' \leq C' \quad o_b \preceq o_c}{\vdash \bigcirc_{o_b} B' \leq \bigcirc_{o_c} C'} \ (14)$$

  1. By pattern matching $B = \bigcirc_{o_b} B'$, $C = \bigcirc_{o_c} C'$
  2. By inversion, $A = \bigcirc_{o_a} A'$ and $\vdash A' \leq B'$ and $o_a \preceq o_b$
  3. By IH, $\vdash A' \leq C'$
  4. Evidently $o_a \preceq o_c$
  5. By rule (14), $\vdash A \leq C$

- Case

  $$\frac{\vdash B' \leq C' \quad b \sqsubseteq c}{\vdash \mathsf{ref}_b\, B' \leq \mathsf{refr}_c\, C'} \ (15)$$

  1. By pattern matching $B = \mathsf{ref}_b\, B'$ and $C = \mathsf{refr}_c\, C'$
  2. By inversion, $A = B$. Immediate.

- Case

  $$\frac{\vdash C' \leq B' \quad c' \sqsubseteq b'}{\vdash \mathsf{ref}_b\, B' \leq \mathsf{refw}_c\, C'} \ (16)$$

1. By pattern matching $B = \mathsf{ref}_b\, B'$ and $C = \mathsf{refw}_c\, C'$
2. By inversion, $A = B$. Immediate.

- Case
$$\frac{\vdash B' \leq C' \quad b \sqsubseteq c}{\vdash \mathsf{refr}_b\, B' \leq \mathsf{refr}_c\, C'} \ (17)$$

1. By pattern matching $B = \mathsf{refr}_b\, B'$, $C = \mathsf{refr}_c\, C'$
2. By inversion, either $A = \mathsf{ref}_a\, A'$ or $A = \mathsf{refr}_a\, A'$, and in either case $a \sqsubseteq b$ and $\vdash A' \leq B'$
3. By IH, $\vdash A' \leq C'$
4. Evidently $a \sqsubseteq c$
5. By either rule (15) or (17), $\vdash A \leq C$

- Case
$$\frac{\vdash C' \leq B' \quad c \sqsubseteq b}{\vdash \mathsf{refw}_b\, B' \leq \mathsf{refw}_c\, C'} \ (18)$$

1. By pattern matching, $B = \mathsf{refw}_b\, B'$, $C = \mathsf{refw}_c\, C'$
2. By inversion, either $A = \mathsf{ref}_a\, A'$ or $A = \mathsf{refw}_a\, A'$, and in either case $b \sqsubseteq a$ and $\vdash B' \leq A'$
3. By IH, $\vdash C' \leq A'$
4. Evidently $c \sqsubseteq a$
5. By either rule (16) or (18), $\vdash A \leq C$

$\square$

C.1.2 Typing judgment properties

Lemma C.4 (Substitution). *If $\Sigma; \Gamma, \Gamma' \vdash M : A$ and*

1. *if $\Sigma; \Gamma, x : A, \Gamma' \vdash N : B$ then $\Sigma; \Gamma, \Gamma' \vdash N[M/x] : B$*
2. *if $\Sigma; \Gamma, x : A, \Gamma' \vdash E \div_o B$ then $\Sigma; \Gamma, \Gamma' \vdash E[M/x] \div_o B$*

Proof. Parts (1) and (2) simultaneously by induction on $\Sigma; \Gamma, x : A, \Gamma' \vdash N : B$ (or $\Sigma; \Gamma, x : A, \Gamma' \vdash E \div_o B$). By cases on the last rule used.

Part (1)

- Case rules (19), (21), (22), (24): Immediate.
- Case rules (23), (26), (27), (28): By IH.
- Case
$$\frac{}{\Sigma; \Gamma, x : A, \Gamma' \vdash x' : (\Gamma, x : A, \Gamma')(x')} \ (20)$$

If $x' \neq x$ then $x'[M/x] = x'$, and by rule (20), $\Sigma; \Gamma, \Gamma' \vdash x' : (\Gamma, \Gamma')(x')$
If $x' = x$ then $x'[M/x] = N$, so $(\Gamma, x : A, \Gamma')(x') = A$, and $\Sigma; \Gamma, \Gamma' \vdash N : A$.

- Case
$$\frac{\Sigma; \Gamma, x : A, \Gamma', y : C \vdash P : D}{\Sigma; \Gamma, x : A, \Gamma' \vdash \lambda y : B.P : C \to D} \ (25)$$

  1. By IH, $\Sigma; \Gamma, \Gamma', y : C \vdash P[M/x] : D$
  2. By rule (25), $\Sigma; \Gamma, \Gamma' \vdash \lambda y : C.P[M/x] : C \to D$
  3. By properties of substitution, $N[M/x] = (\lambda y : C.P)[M/x] = \lambda y : C.P[M/x]$
  4. So, $\Sigma; \Gamma, \Gamma' \vdash N[M/x] : B$

Part (2)

- Case rules (29), (31), (32), (33), (34), (35), (36): By IH.
- Case
$$\frac{\Sigma; \Gamma, x : A, \Gamma' \vdash P : \bigcirc_o C \quad \Sigma; \Gamma, x : A, \Gamma', y : C \vdash F \div_o B}{\Sigma; \Gamma, x : A, \Gamma' \vdash \textsf{let val } y = P \textsf{ in } F \div_o B} \ (30)$$

  1. By IH, $\Sigma; \Gamma, \Gamma' \vdash P[M/x] : \bigcirc_o C$
  2. By IH, $\Sigma; \Gamma, \Gamma', y : C \vdash F[M/x] \div_o B$
  3. By rule (30), $\Sigma; \Gamma, \Gamma' \vdash \textsf{let val } y = P[M/x] \textsf{ in } F[M/x] \div_o B$
  4. By properties of substitution, $E[M/x] = \textsf{let val } y = P \textsf{ in } F[M/x] = \textsf{let val } y = P[M/x] \textsf{ in } F[M/x]$
  5. So $\Sigma; \Gamma, \Gamma' \vdash E[M/x] \div_o B$

□

Lemma C.5 (Inversion). *Two parts:*

- *If* $\Sigma; \Gamma \vdash M : A$ *and*
  1. *if* $M = x$ *then* $\vdash \Gamma(x) \leq A$
  2. *if* $M = *$ *then* $\vdash 1 \leq A$
  3. *if* $M = \textsf{true}$ *or* $M = \textsf{false}$ *then* $\vdash \textsf{bool} \leq A$
  4. *if* $M = \textsf{if } N_1 \textsf{ then } N_2 \textsf{ else } N_3$ *then* $\Sigma; \Gamma \vdash N_1 : \textsf{bool}$, $\Sigma; \Gamma \vdash N_2 : B$, $\Sigma; \Gamma \vdash N_3 : B'$, *and* $\vdash B \leq A$, $\vdash B' \leq A$
  5. *if* $M = \lambda x : B.N$ *then* $\Sigma; \Gamma, x : B \vdash N : C$ *and* $\vdash B \to C \leq A$
  6. *if* $M = NP$ *then* $\Sigma; \Gamma \vdash N : B \to C$ *and* $\Sigma; \Gamma \vdash P : B$ *and* $\vdash C \leq A$
  7. *if* $M = \ell$ *then* $\vdash \textsf{ref}_{\textsf{Level}(\ell)}\, \Sigma(\ell) \leq A$
  8. *if* $M = \textsf{val } E$ *then* $\Sigma; \Gamma \vdash E \div_o B$ *and* $\vdash \bigcirc_o B \leq A$
- *If* $\Sigma; \Gamma \vdash E \div_o A$ *and*
  1. *if* $E = [M]$ *then* $\Sigma; \Gamma \vdash M : A$

2. *if* $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$ *then* $\Sigma; \Gamma \vdash M : \bigcirc_{o'} B$ *and* $\Sigma; \Gamma, x : B \vdash F \div_{o'} C$, $\vdash C \leq A$ *and* $o' = (r', w')$ *with either* $o' \preceq o$ *or* $\vdash C \nearrow r'$ *and* $(\bot, w') \preceq o$
3. *if* $E = \mathsf{ref}_a\,(M : B)$ *then* $\vdash \mathsf{ref}_a\, B \leq A$ *and* $\Sigma; \Gamma \vdash M : B$
4. *if* $E = !M$ *then* $\Sigma; \Gamma \vdash M : \mathsf{refr}_a\, B$ *and* $\vdash B \leq C$, $\vdash C \leq A$ *and either* $(a, \top) \preceq o$ *or* $\vdash C \nearrow a$
5. *if* $E = M := N$ *then* $\Sigma; \Gamma \vdash M : \mathsf{refw}_a\, B$, $\Sigma; \Gamma \vdash N : B$, $\vdash 1 \leq A$, *and* $(\bot, a) \preceq o$

Proof. by induction on the given derivation. By cases on the last rule used.

For part (1), in cases of rules (19) – (27) the result is immediate, by rule (12). In case of rule (28), the result follows by IH, and transitivity.

For part (2), consider the following cases:

- Case rules (29), (30), (31), (32), (33): Immediate
- Case
$$\frac{\Sigma; \Gamma \vdash E \div_{o'} A \quad o' \preceq o}{\Sigma; \Gamma \vdash E \div_o A}\ (34)$$
By IH, there are five subcases
  - Subcase $E = [M]$, $\Sigma; \Gamma \vdash M : A$. Immediate
  - Subcase $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$, $\Sigma; \Gamma \vdash M : \bigcirc_{o''} B$, $\Sigma; \Gamma, x : B \vdash F \div_{o''} C$, $\vdash C \leq A$, and $o'' = (r'', w'')$ with either $o'' \preceq o'$ or $\vdash C \nearrow r''$ and $(\bot, w'') \preceq o'$. Evident, since $o' \preceq o$
  - Subcase $E = \mathsf{ref}_a\,(M : B)$, $\vdash \mathsf{ref}_a\, B \leq A$ $\Sigma; \Gamma \vdash M : B$. Immediate
  - Subcase $E = !M$, $\Sigma; \Gamma \vdash M : \mathsf{refr}_a\, B$ and $\vdash B \leq C$,$\vdash C \leq A$ and either $(a, \top) \preceq o'$ or $\vdash C \nearrow a$. Evident, since $o' \preceq o$
  - Subcase $E = M := N$, $\Sigma; \Gamma \vdash M : \mathsf{refw}_a\, B$, $\Sigma; \Gamma \vdash N : B$, $\vdash 1 \leq A$, and $(\bot, a) \preceq o'$. Evident since $o' \preceq o$
- Case
$$\frac{\Sigma; \Gamma \vdash E \div_{(r,w)} A \quad \vdash A \nearrow r}{\Sigma; \Gamma \vdash E \div_{(\bot,w)} A}\ (35)$$
By pattern matching, $o = (\bot, w)$

  By IH, there are five subcases
  - Subcase $E = [M]$, $\Sigma; \Gamma \vdash M : A$. Immediate
  - Subcase $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$, $\Sigma; \Gamma \vdash M : \bigcirc_{o'} B$, $\Sigma; \Gamma, x : B \vdash F \div_{o'} C$, $\vdash C \leq A$, and $o' = (r', w')$ with either $o' \preceq (r, w)$ or $\vdash C \nearrow r'$ and $(\bot, w') \preceq (r, w)$.
    - If $o' \preceq (r, w)$
      1. By definition, $r' \sqsubseteq r$, and $w \sqsubseteq w'$

2. By rule (36), $\Sigma; \Gamma, x : B \vdash F \div_{o'} A$
3. By rule (12), $\vdash A \leq A$
4. By rule (10), $\vdash A \nearrow r'$
5. By definition, $(\bot, w') \preceq o$

* If $\vdash A \nearrow r'$ and $(\bot, w') \preceq (r, w)$,
  By definition, $(\bot, w') \preceq o$

– Subcase $E = \mathsf{ref}_a\,(M : B)$, $\vdash \mathsf{ref}_a\, B \leq A$ $\Sigma; \Gamma \vdash M : B$. Immediate

– Subcase $E = !M$, $\Sigma; \Gamma \vdash M : \mathsf{refr}_a\, B$ and $\vdash B \leq C$, $\vdash C \leq A$ and either $(a, \top) \preceq (r, w)$ or $\vdash C \nearrow a$.

* If $(a, \top) \preceq (r, w)$
  1. By definition, $a \sqsubseteq r$
  2. By rule (12), $\vdash A \leq A$
  3. By rule (10), $\vdash A \nearrow a$
* If $\vdash C \nearrow a$
  Immediate

– Subcase $E = M := N$, $\Sigma; \Gamma \vdash M : \mathsf{refw}_a\, B$, $\Sigma; \Gamma \vdash N : B$, $\vdash 1 \leq A$, and $(\bot, a) \preceq (r, w)$.
  By definition, $w \sqsubseteq a$. So $(\bot, a) \preceq (\bot, w)$

- Case
$$\frac{\Sigma; \Gamma \vdash E \div_o B \quad \vdash B \leq A}{\Sigma; \Gamma \vdash E \div_o A}\ (36)$$

By IH, there are five subcases

– Subcase $E = [M]$, $\Sigma; \Gamma \vdash M : B$
  By rule (28), $\Sigma; \Gamma \vdash M : A$

– Subcase $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$, $\Sigma; \Gamma \vdash M : \bigcirc_{o'} C$ and $\Sigma; \Gamma, x : C \vdash F \div_{o'} D$, $\vdash D \leq B$ and $o' = (r', w')$ with either $o' \preceq o$ or $\vdash D \nearrow r'$ and $(\bot, w') \preceq o$
  By transitivity, $\vdash D \leq A$.

– Subcase $E = \mathsf{ref}_a\,(M : C)$, $\vdash \mathsf{ref}_a\, C \leq B$ $\Sigma; \Gamma \vdash M : C$
  By transitivity, $\vdash \mathsf{ref}_a\, C \leq A$

– Subcase $E = !M$, $\Sigma; \Gamma \vdash M : \mathsf{refr}_a\, C$ and $\vdash C \leq D$, $\vdash D \leq B$ and either $(a, \top) \preceq o$ or $\vdash D \nearrow a$
  By transitivity, $\vdash D \leq A$

– Subcase $E = M := N$, $\Sigma; \Gamma \vdash M : \mathsf{refw}_a\, C$, $\Sigma; \Gamma \vdash N : C$, $\vdash 1 \leq B$, and $(\bot, a) \preceq o$
  By transitivity, $\vdash 1 \leq A$

□

Lemma C.6 (Canonical Forms). *If* $\Sigma; \cdot \vdash V : A$ *and*

1. *if* $A = 1$ *then* $V = *$
2. *if* $A = \mathsf{bool}$ *then* $V = \mathsf{true}$ *or* $V = \mathsf{false}$
3. *if* $A = B \to C$ *then* $V = \lambda x : B'.M$
4. *if* $A = \mathsf{ref}_a\, B$ *then* $V = \ell$ *and* $\ell \in \mathrm{dom}(\Sigma)$
5. *if* $A = \mathsf{refr}_a\, B$ *then* $V = \ell$ *and* $\ell \in \mathrm{dom}(\Sigma)$
6. *if* $A = \mathsf{refw}_a\, B$ *then* $V = \ell$ *and* $\ell \in \mathrm{dom}(\Sigma)$
7. *if* $A = \bigcirc_o B$ *then* $V = \mathsf{val}\ E$

Proof. by induction on the typing derivation; by inspection of the last typing rule used. □

C.1.3 Store properties

Lemma C.7 (Store Weakening). *If* $\Sigma' \supseteq \Sigma$ *and* Σ' *well-formed, and*

- *if* $\Sigma; \Gamma \vdash M : A$ *then* $\Sigma'; \Gamma \vdash M : A$
- *if* $\Sigma; \Gamma \vdash E \div_o C$ *then* $\Sigma'; \Gamma \vdash E \div_o C$

Proof. by simultaneous induction on the given derivations. By cases on the last rule used.

- Case
$$\frac{}{\Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell)}\ (24)$$
  1. Since Σ' is well-formed, there is at most one occurrence of ℓ in Σ'
  2. Evidently $\ell \in \mathrm{dom}(\Sigma)$, therefore $\ell \in \mathrm{dom}(\Sigma')$.
  3. Since $\Sigma' \supseteq \Sigma$, $\Sigma'(\ell) = \Sigma(\ell)$.
  4. By rule (24), $\Sigma'; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma'(\ell)$.
- All the remaining cases are straightforward by IH.

□

Corollary C.8 (Allocation Safety). *If* $\Sigma; \cdot \vdash V : A$, $\vdash H : \Sigma$ *and if* $\ell \notin \mathrm{dom}(H)$ *then* $\vdash H\{\ell \mapsto V\} : \Sigma\{\ell : A\}$

Proof. 1. By inversion, $\mathrm{dom}(H) = \mathrm{dom}(\Sigma)$, and $\Sigma; \cdot \vdash H(\ell) : \Sigma(\ell')$ for each $\ell' \in \mathrm{dom}(H)$

2. Therefore $\ell \notin \mathrm{dom}(\Sigma)$, and therefore $\Sigma' = \Sigma\{\ell : A\}$ is well-formed.
3. By Store Weakening, $\Sigma'; \cdot \vdash V : A$
4. By Store Weakening, $\Sigma'; \cdot \vdash H(\ell') : \Sigma(\ell')$ for $\ell' \in \mathrm{dom}(H)$

5. Since $\Sigma' \supseteq \Sigma$, $\Sigma'(\ell') = \Sigma(\ell')$ for $\ell' \in \mathrm{dom}(H)$

6. Evidently $\mathrm{dom}(H\{\ell \mapsto V\}) = \mathrm{dom}(\Sigma')$

7. By rule (37), $\vdash H\{\ell \mapsto V\} : \Sigma'$

$\square$

Lemma C.9 (Store Update). *If* $\vdash H : \Sigma$ *and if* $\ell \in \mathrm{dom}(\Sigma)$ *and* $\Sigma; \cdot \vdash V : \Sigma(\ell)$ *then* $\vdash H\{\ell \mapsto V\} : \Sigma$

Proof. 1. By Inversion, $\mathrm{dom}(H) = \mathrm{dom}(\Sigma)$

2. So $\mathrm{dom}(H\{\ell \mapsto V\}) = \mathrm{dom}(\Sigma)$

3. Consider $\ell' \in \mathrm{dom}(\Sigma)$

4. If $\ell' = \ell$, then $H\{\ell \mapsto V\}(\ell') = V$, so $\Sigma; \cdot \vdash H\{\ell \mapsto V\}(\ell') : \Sigma(\ell')$

5. If $\ell' \neq \ell$, then $H\{\ell \mapsto V\}(\ell') = H(\ell')$ so $\Sigma; \cdot \vdash H\{\ell \mapsto V\}(\ell') : \Sigma(\ell')$

6. By rule (37), $\vdash H\{\ell \mapsto V\} : \Sigma$

$\square$

C.1.4 Preservation, Progress and Type safety

Lemma C.10 (Term Preservation). *If* $\Sigma; \cdot \vdash M : A$ *and* $M \rightarrow M'$ *then* $\Sigma; \cdot \vdash M' : A$

Proof. by induction on the evaluation relation. By cases on the last rule used.

- Case IF1: $M = \mathsf{if}\ N_1\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$, $M' = \mathsf{if}\ N_1'\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$, $N_1 \rightarrow N_1'$

  1. By Inversion, $\Sigma; \cdot \vdash N_1 : \mathsf{bool}$, $\Sigma; \cdot \vdash N_2 : B$, $\Sigma; \cdot \vdash N_3 : B'$ and $\vdash B \leq A, \vdash B' \leq A$
  2. By IH, $\Sigma; \cdot \vdash N' : \mathsf{bool}$
  3.

$$\dfrac{\Sigma; \cdot \vdash N_1' : \mathsf{bool} \quad \dfrac{\Sigma; \cdot \vdash N_2 : B \quad \vdash B \leq A}{\Sigma; \cdot \vdash N_2 : A}\,(28) \quad \dfrac{\Sigma; \cdot \vdash N_3 : B' \quad \vdash B' \leq A}{\Sigma; \cdot \vdash N_3 : A}\,(28)}{\Sigma; \cdot \vdash M' : A}\,(23)$$

- Case IFTRUE: $M = \mathsf{if\ true\ then}\ N_2\ \mathsf{else}\ N_3$, $M' = N_2$

  1. By Inversion, $\Sigma; \cdot \vdash \mathsf{true} : \mathsf{bool}$, $\Sigma; \cdot \vdash N_2 : B$, $\Sigma; \cdot \vdash N_3 : B', \vdash B \leq A$, $\vdash B' \leq A$.
  2. By rule (28), $\Sigma; \cdot \vdash N_2 : A$

- Case IFFALSE: $M = \mathsf{if\ false\ then}\ N_2\ \mathsf{else}\ N_3$, $M' = N_3$

  Similar to previous case.

- Case App1: $M = NP$, $N \to N'$, $M' = N'P$
  1. By Inversion, $\Sigma; \cdot \vdash N : B \to C$ and $\Sigma; \cdot \vdash P : B$ and $\vdash C \leq A$
  2. By IH, $\Sigma; \cdot \vdash N' : B \to C$
  3. $$\cfrac{\cfrac{\Sigma; \cdot \vdash N' : B \to C \quad \Sigma; \cdot \vdash P : B}{\Sigma; \cdot \vdash N'P : C}\ (26) \qquad \vdash C \leq A}{\Sigma; \cdot \vdash M' : A}\ (28)$$

- Case App2: $M = NP$, $P \to P'$, $M' = NP'$
  1. By Inversion, $\Sigma; \cdot \vdash N : B \to C$ and $\Sigma; \cdot \vdash P : B$ and $\vdash C \leq A$
  2. By IH, $\Sigma; \cdot \vdash P' : B$
  3. $$\cfrac{\cfrac{\Sigma; \cdot \vdash N : B \to C \quad \Sigma; \cdot \vdash P' : B}{\Sigma; \cdot \vdash NP' : C}\ (26) \qquad \vdash C \leq A}{\Sigma; \cdot \vdash M' : A}\ (28)$$

- Case App: $M = (\lambda x : B.N)P$, $M' = N[P/x]$
  1. By Inversion, $\Sigma; \cdot \vdash \lambda x : B.N : B' \to C'$ and $\Sigma; \cdot \vdash P : B'$ and $\vdash C' \leq A$
  2. By Inversion, $\Sigma; x : B \vdash N : C$ and $\vdash B \to C \leq B' \to C'$
  3. By Subtyping Inversion, $\vdash B' \leq B \vdash C \leq C'$
  4. By transitivity, $\vdash C \leq A$
  5. By rule (28), $\Sigma; \cdot \vdash P : B$
  6. By Substitution, $\Sigma; \cdot \vdash N[P/x] : C$
  7. By rule (28, $\Sigma; \cdot \vdash M' : A$

□

Preservation If $\vdash S \div_o A$ and $S \to S'$ then $\vdash S' \div_o A$

Proof. by induction on the evaluation relation.
By pattern matching, $S = (H, \Sigma, E)$, $S' = (H', \Sigma', E')$, $o = (r, w)$
By Inversion,

- $\vdash H : \Sigma$
- $\Sigma; \cdot \vdash E \div_o A$

Now proceed by cases on the last rule used in $S \to S'$

- Case Ret1: $E = [M]$, $H' = H$, $\Sigma' = \Sigma$, $E' = [M']$, $M \to M'$
  1. By Inversion, $\Sigma; \cdot \vdash M : A$

2. By Term Preservation, $\Sigma; \cdot \vdash M' : A$
3. $$\dfrac{\dfrac{\Sigma; \cdot \vdash M' : A}{\Sigma; \cdot \vdash [M'] \div_{(\bot,\top)} A}\,(29) \qquad (\bot,\top) \preceq o}{\Sigma; \cdot \vdash E' \div_o A}\,(34)$$
4. By rule (38), $\vdash S' \div_o A$

- Case LETVAL1: $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$, $H' = H$, $\Sigma' = \Sigma$, $E' = \mathsf{let\ val}\ x = M'\ \mathsf{in}\ F$, $M \to M'$
  1. By Inversion, $\Sigma; \cdot \vdash M : \bigcirc_{o'} B$ and $\Sigma; x : B \vdash F \div_{o'} C, \vdash C \leq A$ with $o' = (r', w')$ and either $o' \preceq o$ or both $\vdash C \nearrow r'$ and $(\bot, w') \preceq o$
  2. By Term Preservation, $\Sigma; \cdot \vdash M' : \bigcirc_{o'} B$
  3. By rule (30) $\Sigma; \cdot \vdash \mathsf{let\ val}\ x = M'\ \mathsf{in}\ F \div_{o'} C$
  4. If $o' \preceq o$
  $$\dfrac{\dfrac{\Sigma; \cdot \vdash E' \div_{o'} C \quad o' \preceq o}{\Sigma; \cdot \vdash E' \div_o C}\,(34) \qquad \vdash C \leq A}{\Sigma; \cdot \vdash E' \div_o A}\,(36)$$
  5. If $\vdash C \nearrow r'$ and $(\bot, w') \preceq o$
  $$\dfrac{\dfrac{\dfrac{\Sigma; \cdot \vdash E' \div_{o'} C \quad \vdash C \nearrow r'}{\Sigma; \cdot \vdash E' \div_{(\bot,w')} C}\,(35) \qquad (\bot, w') \preceq o}{\Sigma; \cdot \vdash E' \div_o C}\,(34) \qquad \vdash C \leq A}{\Sigma; \cdot \vdash E' \div_o A}\,(36)$$
  6. By rule (38), $\vdash S' \div_o A$

- Case LETVALVAL: $E = \mathsf{let\ val}\ x = \mathsf{val}\, E_1\ \mathsf{in}\ F$, $E' = \mathsf{let\ val}\ x = \mathsf{val}\ E_1'\ \mathsf{in}\ F$, $(H, \Sigma, E_1) \to (H', \Sigma', E_1')$
  1. By Inversion, $\Sigma; \cdot \vdash \mathsf{val}\ E_1 : \bigcirc_{o'} B$ and $\Sigma; x : B \vdash F \div_{o'} C, \vdash C \leq A$ with $o' = (r', w')$ and either $o' \preceq o$ or both $\vdash C \nearrow r'$ and $(\bot, w') \preceq o$
  2. By Inversion, $\Sigma; \cdot \vdash E_1 \div_{o''} B'$ and $\vdash \bigcirc_{o''} B' \leq \bigcirc_{o'} B$
  3. By rule (38), $\vdash (H, \Sigma, E_1) \div_{o''} B'$
  4. By IH, $\vdash (H', \Sigma', E_1') \div_{o''} B'$
  5. By inversion, $\vdash (H' : \Sigma')$, $\Sigma'; \cdot \vdash E_1' \div_{o''} B'$
  6. By rule (27), $\Sigma'; \cdot \vdash \mathsf{val}\ E_1' : \bigcirc_{o''} B'$
  7. By rule (28), $\Sigma'; \cdot \vdash \mathsf{val}\ E_1' : \bigcirc_{o'} B$
  8. By Store Size, $\Sigma' \supseteq \Sigma$
  9. By Store Weakening, $\Sigma'; \cdot \vdash F \div_{o'} C$

10. By rule (30) $\Sigma'; \cdot \vdash \mathsf{let\ val}\ x = \mathsf{val}\ E'_1\ \mathsf{in}\ F \div_{o'} C$
11. If $o' \preceq o$

$$\frac{\Sigma'; \cdot \vdash E' \div_{o'} C \quad o' \preceq o}{\Sigma'; \cdot \vdash E' \div_o C}\ (34)$$

12. If $\vdash C \nearrow r'$ and $(\bot, w') \preceq o$

$$\frac{\dfrac{\Sigma'; \cdot \vdash E' \div_{o'} C \quad \vdash C \nearrow r'}{\Sigma'; \cdot \vdash E' \div_{(\bot,w')} C}\ (35) \qquad (\bot, w') \preceq o}{\Sigma'; \cdot \vdash E' \div_o C}\ (34)$$

13. By rule (36), $\Sigma'; \cdot \vdash E' \div_o A$
14. By rule (38), $\vdash S' \div_o A$

- Case Letval: $E = \mathsf{let\ val}\ x = \mathsf{val}\ [V]\ \mathsf{in}\ F$, $H' = H$, $\Sigma' = \Sigma$, $E' = F[V/x]$

1. By Inversion, $\Sigma; \cdot \vdash \mathsf{val}\ [V] : \bigcirc_{o'} B$ and $\Sigma; x : B \vdash F \div_{o'} C, \vdash C \leq A$, with $o' = (r', w')$ and either $o' \preceq o$ or both $\vdash C \nearrow r'$ and $(\bot, w') \preceq o$
2. By Inversion, $\Sigma; \cdot \vdash [V] \div_{o''} B'$ and $\vdash \bigcirc_{o''} B' \leq \bigcirc_{o'} B$
3. By Inversion, $\Sigma; \cdot \vdash V : B'$
4. By Subtyping Inversion, $\vdash B' \leq B$ and $o'' \preceq o'$
5. By rule (28), $\Sigma; \cdot \vdash V : B$
6. By Substitution, $\Sigma; \cdot \vdash F[V/x] \div_{o'} C$
7. If $o' \preceq o$

$$\frac{\Sigma; \cdot \vdash E' \div_{o'} C \quad o' \preceq o}{\Sigma; \cdot \vdash E' \div_o C}\ (34)$$

8. If $\vdash C \nearrow r'$ and $(\bot, w') \preceq o$

$$\frac{\dfrac{\Sigma; \cdot \vdash E' \div_{o'} C \quad \vdash C \nearrow r'}{\Sigma; \cdot \vdash E' \div_{(\bot,w')} C}\ (35) \qquad (\bot, w') \preceq o}{\Sigma; \cdot \vdash E' \div_o C}\ (34)$$

9. By rule (36), $\Sigma; \cdot \vdash E' \div_o A$
10. By rule (38), $\vdash S' \div_o A$

- Case Ref1: similar to Ret1
- Case Bang1: similar to Ret1
- Case Assn1: similar to Ret1
- Case Assn2: similar to Ret1
- Case Ref: $E = \mathsf{ref}_a\ (V : B)$, $H' = H\{\ell \mapsto V\}$, $\Sigma' = \Sigma\{\ell : B\}$, $E' = [\ell]$, where $\ell \notin \mathrm{dom}(H)$, $\mathsf{Level}(\ell) = a$

1. By Inversion, $\Sigma; \cdot \vdash V : B$, $\vdash \mathsf{ref}_a\, B \leq A$
2. By Allocation Safety, $\vdash H' : \Sigma'$
3. $$\dfrac{\dfrac{\dfrac{\dfrac{}{\Sigma'; \cdot \vdash \ell : \mathsf{ref}_a\, B}\ (24) \quad \vdash \mathsf{ref}_a\, B \leq A}{\Sigma'; \cdot \vdash \ell : A}\ (28)}{\Sigma'; \cdot \vdash [\ell] \div_{(\bot,\top)} A}\ (29) \qquad (\bot,\top) \preceq o}{\Sigma'; \cdot \vdash E' \div_o A}\ (34)$$
4. By rule (38), $\vdash S' \div_o A$

- Case Bang: $E = !\ell$, $H' = H$, $\Sigma' = \Sigma$, $E' = [H(\ell)]$
    1. By Inversion, $\Sigma; \cdot \vdash \ell : \mathsf{refr}_a\, B$, and $\vdash B \leq C$, $\vdash C \leq A$, with either $(a, \top) \preceq o$ or $\vdash C \nearrow a$
    2. By Inversion, $\vdash \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell) \leq \mathsf{refr}_a\, B$
    3. By Subtyping Inversion, $\mathsf{Level}(\ell) \sqsubseteq a$, $\vdash \Sigma(\ell) \leq B$
    4. By transitivity, $\vdash \Sigma(\ell) \leq C$
    5. By Store Typing, $\Sigma; \cdot \vdash H(\ell) : \Sigma(\ell)$
    6. $$\dfrac{\Sigma; \cdot \vdash H(\ell) : \Sigma(\ell) \quad \vdash \Sigma(\ell) \leq C}{\Sigma; \cdot \vdash E' \div_{(a,\top)} C}\ (32)$$
    7. If $(a, \top) \preceq o$, then by rule (34), $\Sigma; \cdot \vdash E' \div_o C$
    8. If $\vdash C \nearrow a$, then
    $$\dfrac{\dfrac{\Sigma; \cdot \vdash E' \div_{(a,\top)} C \quad \vdash C \nearrow a}{\Sigma; \cdot \vdash E' \div_{(\bot,\top)} C}\ (35)}{\Sigma; \cdot \vdash E' \div_o C}\ (34)$$
    9. By rule (36), $\Sigma; \cdot \vdash E' \div_o A$
    10. By rule (38), $\vdash S' \div_o A$

- Case Assn: $E = \ell := V$, $H' = H\{\ell \mapsto V\}$, $\Sigma' = \Sigma$, $E' = [*]$, $\ell \in \mathrm{dom}(H)$
    1. By Inversion, $\Sigma; \cdot \vdash \ell : \mathsf{refw}_b\, B$, $\Sigma; \cdot \vdash V : B$, $\vdash 1 \leq A$, $(\bot, a) \preceq o$
    2. By Inversion, $\vdash \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell) \leq \mathsf{refw}_b\, B$
    3. By Subtyping Inversion, $\vdash B \leq \Sigma(\ell)$, $b \sqsubseteq \mathsf{Level}(\ell)$
    4. By rule (28), $\Sigma; \cdot \vdash V : \Sigma(\ell)$
    5. By Store Update, $\vdash H' : \Sigma$
    6. $$\dfrac{\dfrac{\dfrac{}{\Sigma; \cdot \vdash * : 1}\ (19) \quad \vdash 1 \leq A}{\Sigma; \cdot \vdash E' \div_{(\bot,\top)} 1}\ (36) \qquad (\bot,\top) \preceq o}{\Sigma; \cdot \vdash E' \div_o A}\ (34)$$

7. By rule (38), $\vdash S' \div_o A$

$\square$

Lemma C.11 (Term Progress). *If $\Sigma; \cdot \vdash M : A$ then either M is a value, or $\exists M'$ such that $M \to M'$*

Proof. by induction on the given derivation. By cases on the last rule used.

- Case
$$\frac{\Sigma; \cdot \vdash M : B \quad \vdash B \leq A}{\Sigma; \cdot \vdash M : A} \ (28)$$
By IH.
- Case rules (19), (21), (22), (24), (25), (27): Immediate, evidently M is a value.
- Case rule (20). Vacuous, context is empty.
- Case
$$\frac{\Sigma; \cdot \vdash N_1 : \mathsf{bool} \quad \Sigma; \cdot \vdash N_2 : A \quad \Sigma; \cdot \vdash N_3 : A}{\Sigma; \cdot \vdash \mathsf{if}\ N_1\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3 : A} \ (23)$$
By pattern matching, $M = \mathsf{if}\ N_1\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$.
By IH, either N_1 is a value, or $N_1 \to N_1'$.
  - Subcase N_1 is a value
    1. By Canonical Forms, either $N_1 = \mathsf{true}$ or $N_1 = \mathsf{false}$.
    2. In the former case, $M \to N_2$ by IfTrue
    3. In the latter case, $M \to N_3$ by IfFalse
  - Subcase $N_1 \to N_1'$
  Evidently $M \to M'$ where $M' = \mathsf{if}\ N_1'\ \mathsf{then}\ N_2\ \mathsf{else}\ N_3$ by If.
- Case
$$\frac{\Sigma; \cdot \vdash N : B \to A \quad \Sigma; \cdot \vdash P : B}{\Sigma; \cdot \vdash NP : A} \ (26)$$
By pattern matching, $M = NP$.
By IH, either N is a value, or $N \to N'$
  - Subcase N is a value By IH, either P is a value, or $P \to P'$
    * Subsubcase P is a value
      1. By Canonical Forms, $N = \lambda x : B'.N'$
      2. Let $M' = N'[P/x]$. Evidently $M \to M'$ by App.
    * Subsubcase $P \to P'$
    Let $M' = NP'$. Evidently $M \to M'$ by App2.
  - Subcase $N \to N'$
  Let $M' = N'P$. Evidently $M \to M'$ by App1.

$\square$

Progress If $\vdash S \div_o A$ then either S is terminal, or $\exists S'$ such that $S \to S'$

Proof. By pattern matching, $S = (H, \Sigma, E)$.
By Inversion, $\vdash H : \Sigma$, and $\Sigma; \cdot \vdash E \div_o A$.
Proceed by induction on the typing derivation, by cases on the last rule used:

- Case rules (36), (34), (35): Immediate by IH.
- Case

$$\frac{\Sigma; \cdot \vdash M : A}{\Sigma; \cdot \vdash [M] \div_{(\bot,\top)} A}\ (29)$$

  1. By pattern matching, $E = [M]$, $o = (\bot, \top)$
  2. By Term Progress, either M is a value or $M \to M'$
  3. If M is a value, then S is terminal
  4. If $M \to M'$, let $S' = (H, \Sigma, [M'])$; $S \to S'$ by Ret1

- Case

$$\frac{\Sigma; \cdot \vdash M : B}{\Sigma; \cdot \vdash \mathsf{ref}_a\,(M : B) \div_{(\bot,\top)} \mathsf{ref}_a\,B}\ (31)$$

  1. By pattern matching, $E = \mathsf{ref}_a\,(M : B)$, $o = (\bot, \top)$, $A = \mathsf{ref}_a\,B$
  2. By Term Progress, either M is a value or $M \to M'$
  3. If M is a value, let $S' = (H\{\ell \mapsto M\}, \Sigma\{\ell : B\}, [\ell]$ for $\ell \notin \mathrm{dom}(H)$; $S \to S'$ by Ref
  4. If $M \to M'$, let $S' = (H, \Sigma, \mathsf{ref}_a\,(M' : B))$; $S \to S'$ by Ref1

- Case

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{refr}_a\,A}{\Sigma; \Gamma \vdash !M \div_{(a,\top)} A}\ (32)$$

  1. By pattern matching, $E = !M$, $o = (a, \top)$
  2. By Term Progress, either M is a value or $M \to M'$
     - If M is a value
       (a) By Canonical Forms, $M = \ell$, $\ell \in \mathrm{dom}(\Sigma)$
       (b) By store typing, $\mathrm{dom}(H) = \mathrm{dom}(\Sigma)$, so $\ell \in \mathrm{dom}(H)$
       (c) Let $S' = (H, \Sigma, [H(\ell)]$; $S \to S'$ by Bang
     - If $M \to M'$, let $S' = (H, \Sigma, !M')$; $S \to S'$ by Bang1

- Case

$$\frac{\Sigma; \Gamma \vdash M : \mathsf{refw}_a\,A \quad \Sigma; \Gamma \vdash N : A}{\Sigma; \Gamma \vdash M := N \div_{(\bot,a)} 1}\ (33)$$

  1. By pattern matching, $E = M := N$, $o = (\bot, a)$, $A = 1$
  2. By Term Progress, either M is a value, or $M \to M'$

– If M is a value

(a) By Term Progress, either N is a value or $N \to N'$

  ∗ If N is a value

   i. By Canonical Forms, $M = \ell$, $\ell \in \mathrm{dom}(\Sigma)$

   ii. By store typing, $\mathrm{dom}(H) = \mathrm{dom}(\Sigma)$, so $\ell \in \mathrm{dom}(H)$

   iii. Let $S' = (H\{\ell \mapsto N\}, \Sigma, [*])$; $S \to S'$ by Assn

  ∗ If $N \to N'$, let $S' = (H, \Sigma, M := N')$; $S \to S'$ by Assn2

– If $M \to M'$, let $S' = (H, \Sigma, M' := N)$; $S \to S'$ by Assn1

- Case

$$\frac{\Sigma; \Gamma \vdash M : \bigcirc_o B \quad \Sigma; \Gamma, x : A \vdash F \div_o A}{\Sigma; \Gamma \vdash \mathsf{let\ val}\ x = M\ \mathsf{in}\ F \div_o A} \ (30)$$

1. By pattern matching, $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$
2. By Term Progress, either M is a value, or $M \to M'$

– If M is a value

(a) By Canonical Forms, $M = \mathsf{val}\ E_1$

(b) By Inversion, $\Sigma; \cdot \vdash E_1 \div_{o'} C$, $\vdash \bigcirc_{o'} C \leq \bigcirc_o B$

(c) By IH, (H, Σ, E_1) is either terminal or $(H, \Sigma, E_1) \to (H', \Sigma', E_1')$

  ∗ If (H, Σ, E_1) is terminal

   i. By pattern matching, $E_1 = [V]$

   ii. Let $S' = (H, \Sigma, F[V/x])$; $S \to S'$ by Letval

  ∗ If $(H, \Sigma, E_1) \to (H', \Sigma', E_1')$, let $S' = (H', \Sigma', \mathsf{let\ val}\ x = \mathsf{val}\ E_1'\ \mathsf{in}\ F)$; $S \to S'$ by Letvalval

– If $M \to M'$, let $S' = \mathsf{let\ val}\ x = M'\ \mathsf{in}\ F$, $S \to S'$ by Letval1

□

C.2 Structural properties of equivalence

We show that the judgments for $\approx_\zeta$ admit reflexivity (for well-typed computations), symmetry, and transitivity rules, that is they are equivalence relations on well-typed computation states.

Lemma C.12 (Reflexivity). *1. If $\Sigma; \Gamma \vdash M : A$ then $\Sigma; \Sigma; \Gamma \vdash M \approx_\zeta M : A$.*

2. If $\Sigma; \Sigma; \Gamma \vdash E \div_o C$ then $\Sigma; \Sigma; \Gamma \vdash E \approx_\zeta E \div_o C$

3. If $\vdash H : \Sigma$ then $\vdash (H : \Sigma) \approx_\zeta^U (H : \Sigma)$ for all $U \subseteq \mathrm{dom}(H)$

4. If $\vdash S \div_o C$ then $\vdash S \approx_\zeta S \div_o C$

Proof. Parts (1) and (2) simultaneously by induction on the given derivation, by cases on the last rule used. Parts (3) and (4) follow by inversion on the single rule for the given derivation, and then using parts (1) and (2).

In part (1), the case of store locations is not immediate:

- Case
$$\frac{}{\Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell)}\ (24)$$

  There are two cases, either $\mathsf{Level}(\ell) \sqsubseteq \zeta$ or $\mathsf{Level}(\ell) \not\sqsubseteq \zeta$:

  - Subcase $\mathsf{Level}(\ell) \sqsubseteq \zeta$:
$$\frac{\mathsf{Level}(\ell) \sqsubseteq \zeta \quad \Sigma(\ell) = \Sigma(\ell)}{\Sigma; \Sigma; \Gamma \vdash \ell \approx_\zeta \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell)}\ (47)$$

  - Subcase $\mathsf{Level}(\ell) \not\sqsubseteq \zeta$:
$$\frac{\begin{array}{ll} \Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell) & \dfrac{}{\vdash \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell) \nearrow \mathsf{Level}(\ell)}\ (5) \\ \Sigma; \Gamma \vdash \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell) & \mathsf{Level}(\ell) \not\sqsubseteq \zeta \end{array}}{\Sigma; \Sigma; \Gamma \vdash \ell \approx_\zeta \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell)}\ (39)$$

The remaining cases follow by induction □

Lemma C.13 (Symmetry). *1. If $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$ then $\Sigma_2; \Sigma_1; \Gamma \vdash M_2 \approx_\zeta M_1 : A$.*

2. If $\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o C$ then $\Sigma_2; \Sigma_1; \Gamma \vdash E_2 \approx_\zeta E_1 \div_o C$

3. If $\vdash (H_1 : \Sigma_1) \approx^U_\zeta (H_2 : \Sigma_2)$ then $\vdash (H_2 : \Sigma_2) \approx^U_\zeta (H_1 : \Sigma_1)$

4. If $\vdash S_1 \approx_\zeta S_2 \div_o C$ then $\vdash S_2 \approx_\zeta S_1 \div_o C$

Proof. by induction on derivations. Evident as all the judgments are symmetric. □

Lemma C.14 (Transitivity). *Four parts:*

1. *If $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$ and $\Sigma_2; \Sigma_3; \Gamma \vdash M_2 \approx_\zeta M_3 : A$ then $\Sigma_1; \Sigma_3; \Gamma \vdash M_1 \approx_\zeta M_3 : A$*
2. *If $\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o C$ and $\Sigma_2; \Sigma_3; \Gamma \vdash E_2 \approx_\zeta E_3 \div_o C$ then $\Sigma_1; \Sigma_1; \Gamma \vdash E_1 \approx_\zeta E_3 \div_o C$*
3. *If $\vdash (H_1 : \Sigma_1) \approx^U_\zeta (H_2 : \Sigma_2)$ and $\vdash (H_2 : \Sigma_2) \approx^U_\zeta (H_3 : \Sigma_3)$ then $\vdash (H_1 : \Sigma_1) \approx^U_\zeta (H_3 : \Sigma_3)$*
4. *If $\vdash S_1 \approx_\zeta S_2 \div_o C$ and $\vdash S_2 \approx_\zeta S_3 \div_o C$ then $\vdash S_1 \approx_\zeta S_3 \div_o C$*

Proof. Parts (1) and (2) follow by simultaneous induction on derivations.

Part (3):

1. By Inversion on each given derivation, $\vdash H_i : \Sigma_i$ for $i = 1, 2, 3$, $\Sigma_1 \upharpoonright U = \Sigma_2 \upharpoonright U = \Sigma_3 \upharpoonright U$, and for each $\ell \in U$, $\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : \Sigma_1(\ell)$ and $\Sigma_2; \Sigma_3; \cdot \vdash H_2(\ell) \approx_\zeta H_3(\ell) : \Sigma_2(\ell)$
2. By Part (1), for each $\ell \in U$, $\Sigma_1; \Sigma_3; \cdot \vdash H_1(\ell) \approx_\zeta H_3(\ell) : \Sigma_1(\ell)$
3. By rule (58), $\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_3 : \Sigma_3)$

Part (4):

1. By pattern matching, $S_i = (H_i, \Sigma_i, E_i)$ for $i = 1, 2, 3$
2. By Inversion, $\vdash (H_1 : \Sigma_1)) \approx_\zeta^{U_{12}} (H_2 : \Sigma_2)$, $\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C$ where $U_{12} = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap {\downarrow}(\zeta)$
3. By Inversion, $\vdash (H_2 : \Sigma_2)) \approx_\zeta^{U_{23}} (H_3 : \Sigma_3)$, $\Sigma_2; \Sigma_3; \cdot \vdash E_2 \approx_\zeta E_3 \div_o C$ where $U_{23} = \mathrm{dom}(\Sigma_2) \cap \mathrm{dom}(\Sigma_3) \cap {\downarrow}(\zeta)$
4. Let $U_{13} = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_3) \cap {\downarrow}(\zeta)$
5. Suppose $\ell \in U_{13} \setminus (\mathrm{dom}(\Sigma_2) \cap {\downarrow}(\zeta))$
   (a) Evidently, $\ell \notin U_{12}$ and $\ell \notin U_{23}$
   (b) Choose $\ell' \notin U_{13} \cup \mathrm{dom}(\Sigma_2)$ such that $\mathsf{Level}(\ell') = \mathsf{Level}(\ell)$
   (c) α-vary (H_3, Σ_3, E_3) with ℓ' for ℓ
6. So for all $\ell \in U_{13}$, $\ell \in \mathrm{dom}(\Sigma_2) \cap {\downarrow}(\zeta)$
7. Evidently, $U_{13} \subseteq U_{12}$ and $U_{13} \subseteq U_{23}$
8. By Store Equivalence Coarsening, $\vdash (H_1 : \Sigma_1) \approx_\zeta^{U_{13}} (H_2 : \Sigma_2)$, and $\vdash (H_2 : \Sigma_2) \approx_\zeta^{U_{13}} (H_3 : \Sigma_3)$
9. By Part (3), $\vdash (H_1 : \Sigma_1) \approx_\zeta^{U_{13}} (H_3 : \Sigma_3)$
10. By Part (2), $\Sigma_1; \Sigma_3; \cdot \vdash E_1 \approx_\zeta E_3 \div_o C$
11. By rule (59), $\vdash S_1 \approx_\zeta S_3 \div_o C$

$\square$

Lemma C.15 (Regularity of Equivalence). *Four parts:*

1. *If* $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$ *then* $\Sigma_i; \Gamma \vdash M_i : A$
2. *If* $\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o C$ *then* $\Sigma_i; \Gamma \vdash E_i \div_o C$
3. *If* $\vdash (H_1 : \Sigma_1) \approx_\zeta^U (H_2 : \Sigma_2)$ *then* $\vdash H_i : \Sigma_i$
4. *If* $\vdash S_1 \approx_\zeta S_2 \div_o C$ *then* $\vdash S_i \div_o C$

Proof. by induction on the derivations. $\square$

Lemma C.16 (Store Equivalence Coarsening). *If* $\vdash (H_1 : \Sigma_1) \approx_{\zeta}^{U'} (H_2 : \Sigma_2)$ *and* $U \subseteq U'$ *then* $\vdash (H_1 : \Sigma_1) \approx_{\zeta}^{U} (H_2 : \Sigma_2)$

Proof.

1. By Inversion, $\vdash H_i : \Sigma_i$ for $i = 1, 2$, $\Sigma_1 \restriction U' = \Sigma_2 \restriction U'$, for each $\ell \in U'$, $\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_{\zeta} H_2(\ell) : \Sigma_1(\ell)$
2. Evidently, $\Sigma_1 \restriction U = \Sigma_2 \restriction U$
3. Evidently, for each $\ell \in U$, $\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_{\zeta} H_2(\ell) : \Sigma_1(\ell)$
4. By rule (58), $\vdash (H_1 : \Sigma_1) \approx_{\zeta}^{U} (H_2 : \Sigma_2)$

□

Lemma C.17 (Equivalent Values). *If* $\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_{\zeta} M_2 : A$ *then* M_1 *is a value if and only if* M_2 *is a value.*

Proof. by induction on the equivalence derivation. By cases on the last rule used.

- Case:rules (39), (40), (42), (43), (45), (47), (48). Evidently both M_1 and M_2 are values.
- Case:rule (41). Vacuous, $\Gamma = \cdot$.
- Case:rule (44), (46). Evidently both M_1 and M_2 are not values.
- Case:rule (49). By IH.

□

With the Equivalent Values lemma in hand, we can establish the Hexagon Lemma for terms.

C.3 Term Hexagon lemma proof

Term Hexagon Lemma For all ζ, if $\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_{\zeta} M_2 : A$ and $M_1 \to M_1'$ and $M_2 \to M_2'$ and $M_1' \downarrow$ and $M_2' \downarrow$, then there exist M_1'', M_2'' such that $M_1' \to^* M_1''$, $M_2' \to^* M_2''$, $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_{\zeta} M_2'' : A$

Proof. by induction on the given derivation. By cases on the last rule used.

- Cases rules (39), (40), (42), (43), (45), (47), (48). Vacuous, M_1, M_2 are values, no applicable evaluation rules.
- Case rule (41). Vacuous, $\Gamma = \cdot$
- Case rule (49). By IH.

- Case rule (44):

$$\frac{\Sigma_1;\Sigma_2;\cdot\vdash N_1 \approx_\zeta N_2 : \mathsf{bool} \quad \Sigma_1;\Sigma_2;\cdot\vdash P_{11} \approx_\zeta P_{21} : A \quad \Sigma_1;\Sigma_2;\cdot\vdash P_{12} \approx_\zeta P_{22} : A}{\Sigma_1;\Sigma_2;\cdot\vdash \mathsf{if}\ N_1\ \mathsf{then}\ P_{11}\ \mathsf{else}\ P_{12} \approx_\zeta \mathsf{if}\ N_2\ \mathsf{then}\ P_{21}\ \mathsf{else}\ P_{22} : A} \quad (44)$$

  By pattern matching, $M_i = \mathsf{if}\ N_i\ \mathsf{then}\ P_{i1}\ \mathsf{else}\ P_{i2}$ for $i = 1, 2$

  There are three possible evaluation rules for $M_1 \to M_1'$

  - Case If1: $M_1' = \mathsf{if}\ N_1'\ \mathsf{then}\ P_{11}\ \mathsf{else}\ P_{12}$, $N_1 \to N_1'$
    1. By Equivalent Values, N_2 is not a value
    2. The only applicable evaluation rule for $M_2 \to M_2'$ is If1: $M_2' = \mathsf{if}\ N_2'\ \mathsf{then}\ P_{21}\ \mathsf{else}\ P_{22}$, $N_2 \to N_2'$
    3. By Subterm Termination, $N_1' \downarrow$, $N_2' \downarrow$
    4. By IH, there exist N_1'', N_2'' such that $N_i' \to^* N_i''$ for $i = 1, 2$, and $\Sigma_1;\Sigma_2;\cdot\vdash N_1'' \approx_\zeta N_2'' : \mathsf{bool}$
    5. By repeated application of If1, $M_i' \to^* M_i''$ for $i = 1, 2$
    6. By rule (44), $\Sigma_1;\Sigma_2;\cdot\vdash M_1'' \approx_\zeta M_2'' : A$
  - Case IfTrue: $N_1 = \mathsf{true}$, $M_1' = P_{11}$
    1. By Equivalent Values, N_2 is a value
    2. By Equivalent Term Inversion, there are two subcases:
       * Either there exists a B such that $\vdash B \leq \mathsf{bool}$, $\vdash B \nearrow a$, $a \not\sqsubseteq \zeta$ and $\Sigma_i;\cdot\vdash N_i : B$
         By subtyping inversion, $B = \mathsf{bool}$. By Informativeness Inversion, $a = \bot$, for a contradiction (since $\bot \sqsubseteq \zeta$)
       * Or $N_2 = \mathsf{true}$
       (a) There is a single applicable evaluation rule for $M_2 \to M_2'$, IfTrue: $M_2' = P_{21}$.
       (b) Let $M_i'' = M_i'$ for $i = 1, 2$.
       (c) Evidently, $\Sigma_1;\Sigma_2;\cdot\vdash M_1'' \approx_\zeta M_2'' : A$
  - Case IfFalse: $N_1 = \mathsf{false}$, $M_2' = P_{12}$
    Similar to previous case.

- Case rule (46):

$$\frac{\Sigma_1;\Sigma_2;\cdot\vdash N_1 \approx_\zeta N_2 : B \to A \quad \Sigma_1;\Sigma_2;\cdot\vdash P_1 \approx_\zeta P_2 : B}{\Sigma_1;\Sigma_2;\cdot\vdash N_1 P_1 \approx_\zeta N_2 P_2 : A} \quad (46)$$

  By pattern matching, $M_i = N_i P_i$ for $i = 1, 2$

  There are three possible evaluation rules for $M_1 \to M_1'$

  - Case App1: $M_1' = N_1' P_1$, $N_1 \to N_1'$
    1. By Equivalent Values, N_2 is not a value

2. The only applicable evaluation rule for $M_2 \to M_2'$ is App1: $M_2' = N_2'P_2$, $N_2 \to N_2'$
3. By Subterm Termination, $N_1 \downarrow$, $N_2 \downarrow$
4. By IH, there exist N_1'', N_2'' such that $N_i' \to^* N_i''$ for $i = 1, 2$, and $\Sigma_1; \Sigma_2; \cdot \vdash N_1'' \approx_\zeta N_2'' : B \to A$
5. Let $M_i'' = N_i'' P_i$ for $i = 1, 2$
6. By repeated application of App1, $M_i' \to^* M_i''$ for $i = 1, 2$
7. By rule (46), $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : A$

– Case App2: N_1 value, $M_1' = N_1 P_1'$, $P_1 \to P_1'$
1. By Equivalent Values, N_2 is a value
2. By Equivalent Values, P_2 is not a value
3. The only applicable evaluation rule for $M_2 \to M_2'$ is App2: $M_2' = N_2 P_2'$, $P_2 \to P_2'$
4. By IH, there exist P_1'', P_2'', such that $P_i' \to^* P_i''$ for $i = 1, 2$, and $\Sigma_1; \Sigma_2; \cdot \vdash P_1'' \approx_\zeta P_2'' : B$
5. Let $M_i'' = N_i P_i''$ for $i = 1, 2$
6. By repeated application of App2, $M_i' \to M_i''$ for $i = 1, 2$
7. By rule (46), $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : A$

– Case App: $N_1 = \lambda x : B_1.M_{11}$, P_1 value, $M_1' = M_{11}[P_1/x]$
1. Evidently, N_1 is a value
2. By Equivalent Values, N_2 is a value, P_2 is a value
3. By Equivalent Term Inversion, there are two subcases:
   * Either $\vdash Q \leq B \to A$ and $\vdash Q \nearrow a$ and $a \not\sqsubseteq \zeta$ and $\Sigma_i; \cdot \vdash N_i : D \to C$ for $i = 1, 2$

   (a) By Subtyping Inversion, $Q = D \to C$, $\vdash B \leq D$ and $\vdash C \leq A$
   (b) By Canonical Forms, $N_2 = \lambda x : B_2.M_{21}$
   (c) There is a single applicable evaluation rule for $M_2 \to M_2'$, App: $M_2' = M_{21}[P_2/x]$
   (d) By Inversion, $\Sigma_1; x : B_i \vdash M_{i1} : A_i$ and $\vdash B_i \to A_i \leq D \to C$ for $i = 1, 2$
   (e) By Subtyping Inversion, $\vdash D \leq B_i$ and $\vdash A_i \leq C$ for $i = 1, 2$
   (f) By transitivity, $\vdash B \leq B_i$ for $i = 1, 2$
   (g) By Regularity, $\Sigma_i; \cdot \vdash P_i : B$ for $i = 1, 2$
   (h) By rule (28), $\Sigma_i; \cdot \vdash P_i : B_i$ for $i = 1, 2$
   (i) By Substitution, $\Sigma_i; \cdot \vdash M_i' : A_i$ for $i = 1, 2$
   (j) By rule (28), $\Sigma_i; \cdot \vdash M_i : C$ for $i = 1, 2$
   (k) Since $M_i' \downarrow$, there is some V_i, such that $M_i' \to^* V_i$ for $i = 1, 2$
   (l) Let $M_i'' = V_i$ for $i = 1, 2$

(m) By Transitivity of Preservation, $\Sigma_i; \cdot \vdash V_i : C$
(n) By Informativeness Inversion, $\vdash C \nearrow a$
(o) By rule (39), $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : C$
(p) By rule (49), $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : A$

∗ Or $N_2 = \lambda x : C.M_{21}$, $\Sigma_1; \Sigma_2; x : C \vdash M_{11} \approx_\zeta M_{21} : D$, $\vdash C \to D \leq B \to A$

(a) There is a single applicable evaluation rule for $M_2 \to M_2'$, App: $M_2' = M_{21}[P_2/x]$
(b) Let $M_i'' = M_i$
(c) By Subtyping Inversion, $\vdash B \leq C$, $\vdash D \leq A$
(d) By rule (49), $\Sigma_1; \Sigma_2; \cdot \vdash P_1 \approx_\zeta P_2 : C$
(e) By Functionality, $\Sigma_1; \Sigma_2; \cdot \vdash M_{11}[P_1/x] \approx_\zeta M_{21}[P_2/x] : D$
(f) By rule (49), $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : A$

□

C.4 High Security Step proof

Single High Security Step If $\vdash (H, \Sigma, E) \div_o A$, $o = (r, w)$ and $w \not\sqsubseteq \zeta$, and $(H, \Sigma, E) \to (H', \Sigma', E')$ then $\vdash (H : \Sigma) \approx_\zeta^{\mathrm{dom}(\Sigma) \cap \downarrow(\zeta)} (H' : \Sigma')$.

Proof. by induction on $(H, \Sigma, E) \to (H', \Sigma', E')$

By inversion on $\vdash (H, \Sigma, E) \div_o A$, we have

- $\vdash H : \Sigma$
- $\Sigma; \cdot \vdash E \div_o A$

Now consider cases on the evaluation rule used:

- Case Ret1: $E = [M]$, $H' = H$, $\Sigma' = \Sigma$, $E' = [N]$, $M \to N$

  By Reflexivity, $\vdash (H : \Sigma) \approx_\zeta^U (H' : \Sigma')$ where $U = \mathrm{dom}(\Sigma) \cap \downarrow(\zeta)$

- Case Letval1: $E = \mathsf{let\ val}\ x = M\ \mathsf{in}\ F$, $H' = H$, $\Sigma' = \Sigma$, $E' = \mathsf{let\ val}\ x = N\ \mathsf{in}\ F$, $M \to N$

  Identical to preceding case

- Case Ref1: $E = \mathsf{ref}_a (M : B)$, $H' = H$, $\Sigma' = \Sigma$, $E' = \mathsf{ref}_a (N : B)$, $M \to N$

  Identical to Ret1 case

- Case Bang1: $E =!M$, $H'' = H$, $\Sigma' = \Sigma$, $E' =!N$, $M \to N$

  Identical to Ret1 case

- Case Assn1: $E = M := N$, $H' = H$, $\Sigma' = \Sigma$, $E' = M' := N$, $M \to M'$

  Identical to Ret1 case

- Case Assn2: $E = M := N$, M value, $H' = H$, $\Sigma' = \Sigma$, $E' = M := N'$, $N \to N'$

  Identical to Ret1 case

- Case LetValVal: $E = \mathsf{let\ val}\ x = \mathsf{val}\, E_1\ \mathsf{in}\ F$, $E' = \mathsf{let\ val}\ x = \mathsf{val}\, E_2\ \mathsf{in}\ F$, $(H, \Sigma, E_1) \to (H', \Sigma', E_2)$

  1. By Inversion, for some $o' = (r', w')$, $\Sigma; \cdot \vdash \mathsf{val}\ E_1 : \bigcirc_{o'} B$, $\Sigma; x : B \vdash F \div_{o'} C$, $\vdash C \leq A$, where either $o' \preceq o$ or both $\vdash C \nearrow r'$ and $(\perp, w') \preceq o$
  2. Either way, $w \sqsubseteq w'$, so $w' \not\sqsubseteq \zeta$
  3. By Inversion, $\Sigma; \cdot \vdash E_1 \div_{o''} B'$ and $\vdash \bigcirc_{o''} B' \leq \bigcirc_{o'} B$
  4. By rule (38), $\vdash (H, \Sigma, E_1) \div_{o''} B'$
  5. By IH, $\vdash (H : \Sigma) \approx^U_\zeta (H' : \Sigma')$ where $U = \mathrm{dom}(\Sigma) \cap {\downarrow}(\zeta)$

- Case LetVal: $E = \mathsf{let\ val}\ x = \mathsf{val}\ [V]\ \mathsf{in}\ F$, $H' = H$, $\Sigma' = \Sigma$, $E' = F[V/x]$

  Identical to the Ret1 case

- Case Bang: $E = !\ell$, $H' = H$, $\Sigma' = \Sigma$, $E' = H(\ell)$

  Identical to Ret1 case

- Case Ref: $E = \mathsf{ref}_a\,(V : B)$, $H' = H\{\ell \mapsto V\}$, $\Sigma' = \Sigma\{\ell : B\}$, $E' = \ell$, $\ell \notin \mathrm{dom}(H)$, $\mathsf{Level}(\ell) = a$

  1. By Inversion, $\Sigma; \cdot \vdash V : B$, $\vdash \mathsf{ref}_a\, B \leq A$
  2. By rule (37), $\vdash H' : \Sigma'$
  3. Consider $\ell' \in U$, by construction, $H'(\ell') = H(\ell')$ and $\Sigma'(\ell') = \Sigma(\ell')$
  4. By rule (58) $\vdash (H : \Sigma) \approx^U_\zeta (H' : \Sigma')$

- Case Assn: $E = \ell := V$, $H' = H\{\ell \mapsto V\}$, $\Sigma' = \Sigma$, $E' = [*]$

  1. By Inversion, $\Sigma; \cdot \vdash \ell : \mathsf{refw}_a\, B$, $\Sigma; \cdot \vdash V : B$, $(\perp, a) \preceq o$, $\vdash 1 \leq A$
  2. By Inversion, $\vdash \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma(\ell) \leq \mathsf{refw}_a\, B$
  3. By Subtyping Inversion, $\vdash B \leq \Sigma(\ell)$, $a \sqsubseteq \mathsf{Level}(\ell)$
  4. Since $(\perp, a) \preceq o$, $w \sqsubseteq a \sqsubseteq \mathsf{Level}(\ell)$
  5. Since $w \not\sqsubseteq \zeta$, $\mathsf{Level}(\ell) \not\sqsubseteq \zeta$, so $\ell \notin U$ where $U = \mathrm{dom}(\Sigma) \cap {\downarrow}(\zeta)$
  6. By rule (37), $\vdash H' : \Sigma'$
  7. By rule (58), $\vdash (H : \Sigma) \approx^U_\zeta (H' : \Sigma')$

$\square$

We showed Equivalent Term Inversion in the body of the paper. Equivalent Expression Inversion is proved below:

Lemma C.18 (Equivalent Expression Inversion). *If* $\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o A$ *then*

1. *if* $E_1 = [M_1]$ *then* $E_2 = [M_2]$ *and* $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$

Proof. by induction on the given derivation. By cases on the last rule used. The following cases are relevant:

- Case
$$\frac{\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A}{\Sigma_1; \Sigma_2; \Gamma \vdash [M_1] \approx_\zeta [M_2] \div_{(\bot,\top)} A} \ (53)$$
Immediate.

- Case rules (50), (51): by IH.

- Case
$$\frac{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o B \quad \vdash B \leq A}{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o A} \ (52)$$
  1. By IH, if $E_1 = [M_1]$ then $E_2 = [M_2]$ and $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : B$
  2. By rule (49), $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$

□

Next we show that any equivalent terms (or expressions) are equivalent under larger store types. This is a technical result necessary for later proofs.

Lemma C.19 (Simultaneous Store Weakening). *If* $\Sigma'_1 \supseteq \Sigma_1$ *and* $\Sigma'_2 \supseteq \Sigma_2$, *and if* Σ'_1, Σ'_2 *are well-formed, then*

- *if* $\Sigma_1; \Sigma_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$ *then* $\Sigma'_1; \Sigma'_2; \Gamma \vdash M_1 \approx_\zeta M_2 : A$
- *if* $\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o C$ *then* $\Sigma'_1; \Sigma'_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_o C$

Proof. by induction on the given derivation, by cases on the last rule used. Most cases are immediate by IH, since they do not pose any restrictions on Σ_i. The following cases of the first part are interesting:

- Case
$$\frac{\Sigma_1; \Gamma \vdash V_1 : A \quad \Sigma_2; \Gamma \vdash V_2 : A \quad \vdash A \nearrow a \quad a \not\sqsubseteq \zeta}{\Sigma_1; \Sigma_2; \Gamma \vdash V_1 \approx_\zeta V_2 : A} \ (39)$$
  1. By Store Weakening, $\Sigma'_i; \Gamma \vdash V_i : A$ for $i = 1, 2$
  2. By rule (39), $\Sigma'_1; \Sigma'_2; \Gamma \vdash V_1 \approx_\zeta V_2 : A$

- Case
$$\frac{\mathsf{Level}(\ell) \sqsubseteq \zeta \quad \Sigma_1(\ell) = \Sigma_2(\ell)}{\Sigma_1; \Sigma_2; \Gamma \vdash \ell \approx_\zeta \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma_1(\ell)} \ (47)$$

1. Since Σ'_i is well-formed, and $\Sigma'_i \supseteq \Sigma_i$, $\Sigma'_i(\ell) = \Sigma_i(\ell)$ for $i = 1, 2$
2. Consequently, $\Sigma'_1(\ell) = \Sigma'_2(\ell)$
3. By rule (47), $\Sigma'_1; \Sigma'_2; \Gamma \vdash \ell \approx_\zeta \ell : \mathsf{ref}_{\mathsf{Level}(\ell)}\, \Sigma'_1(\ell)$

□

C.5 Hexagon lemma proof

Hexagon Lemma For all ζ, if $o = (r, w)$ with $r \sqsubseteq \zeta$, and if

- $\vdash S_1 \approx_\zeta S_2 \div_o C$
- $S_1 \to S'_1$, $S_2 \to S'_2$
- $S'_1 \downarrow$, $S'_2 \downarrow$

then there exist S''_1, S''_2 such that

- $S'_1 \to^* S''_1$, $S'_2 \to^* S''_2$
- $\cdot \vdash S''_1 \approx_\zeta S''_2 \div_o C$

Proof. By Inversion on $\vdash S_1 \approx_\zeta S_2 \div_o C$,

- $S_1 = (H_1, \Sigma_1, E_1)$, $S_2 = (H_2, \Sigma_2, E_2)$
- $\vdash (H_1 : \Sigma_1) \approx^U_\zeta (H_2 : \Sigma_2)$ where $U = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap {\downarrow}(\zeta)$
- $\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C$

We prove, by induction on the derivation of $\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C$ the following statement:

Given

- $\vdash (H_1 : \Sigma_1) \approx^U_\zeta (H_2 : \Sigma_2)$ where $U = \mathrm{dom}(\Sigma_1) \cap \mathrm{dom}(\Sigma_2) \cap {\downarrow}(\zeta)$
- $(H_1, \Sigma_1, E_1) \to S'_1$ and $(H_2, \Sigma_2, E_2) \to S'_2$
- $S'_1 \downarrow$, $S'_2 \downarrow$

there exist $S''_1 = (H''_1, \Sigma''_1, E''_1), S''_2 = (H''_2, \Sigma''_2, E''_2)$ such that

1. $S'_1 \to^* S''_1$, $S'_2 \to^* S''_2$
2. $\vdash (H''_1 : \Sigma''_1) \approx^{U'}_\zeta (H''_2 : \Sigma''_2)$
   where $U' = \mathrm{dom}(\Sigma''_1) \cap \mathrm{dom}(\Sigma''_2) \cap {\downarrow}(\zeta)$
3. $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C$

Note that the conclusion of the induction hypothesis suffices to show the conclusion of the lemma, by rule (59).

Case analyze the last rule used in the derivation.

- Case
$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash E_2 \approx_\zeta E_2 \div_o B \quad \vdash B \leq C}{\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C} \ (52)$$

  1. By IH, there exists $S''_1 = (H''_1, \Sigma''_1, E''_1), S''_2 = (H''_2, \Sigma''_2, E''_2)$ such that
     - $S'_1 \to^* S''_1$ and $S'_2 \to S''_2$
     - $\vdash (H''_1 : \Sigma''_1) \approx^{U'}_\zeta (H''_2 : \Sigma''_2)$ where $U' = \mathrm{dom}(\Sigma''_1) \cap \mathrm{dom}(\Sigma''_2) \cap {\downarrow}(\zeta)$
     - $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o B$
  2. By rule (52), $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C$

- Case
$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_{o'} C \quad o' \preceq o}{\Sigma_1; \Sigma_2; \cdot \vdash E_1 \approx_\zeta E_2 \div_o C} \ (50)$$

  1. From $o' \preceq o$, $r' \sqsubseteq r \sqsubseteq \zeta$
  2. By IH, there exist $S''_1 = (H''_1, \Sigma''_1, E''_1), S''_2 = (H''_2, \Sigma''_2, E''_2)$ such that
     - $S'_i \to^* S''_i$ for $i = 1, 2$
     - $\vdash (H''_1 : \Sigma''_1) \approx^{U'}_\zeta (H''_2 : \Sigma''_2)$
       where $U' = \mathrm{dom}(E''_1) \cap \mathrm{dom}(E''_2) \cap {\downarrow}(\zeta)$
     - $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_{o'} C$
  3. By rule (50), $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C$

- Case
$$\frac{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_{(r',w)} C \quad \vdash C \nearrow r'}{\Sigma_1; \Sigma_2; \Gamma \vdash E_1 \approx_\zeta E_2 \div_{(\bot,w)} C} \ (51)$$

  By pattern matching, $r = \bot$

  Consider two subcases: either $r' \sqsubseteq \zeta$ or $r' \not\sqsubseteq \zeta$

  - If $r' \sqsubseteq \zeta$
    1. By IH, there exist $S''_1 = (H''_1, \Sigma''_1, E''_1), S''_2 = (H''_2, \Sigma''_2, E''_2)$ such that
       * $S'_i \to^* S''_i$ for $i = 1, 2$
       * $\vdash (H''_1 : \Sigma''_1) \approx^{U'}_\zeta (H''_2 : \Sigma''_2)$
         where $U' = \mathrm{dom}(\Sigma''_1) \cap \mathrm{dom}(\Sigma''_2) \cap {\downarrow}(\zeta)$
       * $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_{(r',w)} C$
    2. By rule (51), $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C$
  - If $r' \not\sqsubseteq \zeta$
    1. Since $r' \sqsubseteq w$, then $w \not\sqsubseteq \zeta$
    2. By Regularity of Equivalence, $\vdash H_i : \Sigma_i$, $\Sigma_i; \cdot \vdash E_i \div_{(r',w)} C$ for $i = 1, 2$

3. Since $S'_i \downarrow$, $(H_i, \Sigma_i, E_i) \to^+ (H''_i, \Sigma''_i, [V_i])$ for some $S''_i = (H''_i, \Sigma''_i, [V_i])$ for $i = 1, 2$
4. By HSS
   - $\ast \vdash (H''_1 : \Sigma''_1) \approx^U_\zeta (H''_2 : \Sigma''_2)$
   - $\ast\ U = \mathrm{dom}(H''_1) \cap \mathrm{dom}(H''_2) \cap \downarrow(\zeta)$
5. By repeatedly applying Preservation, $\Sigma''_i; \cdot \vdash [V_i] \div_{(r',w)} C$ for $i = 1, 2$
6. By Inversion, $\Sigma''_i; \cdot \vdash V_i : C$ for $i = 1, 2$
7. $$\dfrac{\dfrac{\dfrac{\begin{array}{ll}\Sigma''_1; \cdot \vdash V_1 : C & \vdash C \nearrow r' \\ \Sigma''_2; \cdot \vdash V_2 : C & r' \not\sqsubseteq \zeta\end{array}}{\Sigma''_1; \Sigma''_2; \cdot \vdash V_1 \approx_\zeta V_2 : C}\ (39)}{\Sigma''_1; \Sigma''_2; \cdot \vdash [V_1] \approx_\zeta [V_2] \div_{(\bot,\top)} C}\ (53) \qquad (\bot, \top) \preceq o}{\Sigma''_1; \Sigma''_2; \cdot \vdash [V_1] \approx_\zeta [V_2] \div_o C}\ (50)$$

- Case
$$\dfrac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : C}{\Sigma_1; \Sigma_2; \cdot \vdash [M_1] \approx_\zeta [M_2] \div_{(\bot,\top)} C}\ (53)$$

1. By pattern matching, $E_i = [M_i]$, $o = (r, w) = (\bot, \top)$
2. The only applicable evaluation rules are: for $i = 1, 2$
$$\dfrac{M_i \to M'_i}{(H_i, \Sigma_i, [M_i]) \to (H_i, \Sigma_i, [M'_i])}\ \textsc{Ret1}$$
3. By Term Hexagon Lemma, there exist M''_1, M''_2 such that $M'_1 \to^* M''_1$ $M'_2 \to^* M''_2$ $\Sigma_1; \Sigma_2; \cdot \vdash M''_1 \approx_\zeta M''_2 : C$
4. Let $H''_i = H_i$, $E''_i = [M''_i]$, $\Sigma''_i = \Sigma_i$, $U' = U$
5. By repeated application of Ret1, $(H_i, \Sigma_i, [M'_i]) \to^* (H''_i, \Sigma''_i, E''_i)$
6. Evidently, $\vdash (H''_1 : \Sigma''_1) \approx^{U'}_\zeta (H''_2 : \Sigma''_2)$
7. By (53), $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C$

- Case
$$\dfrac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : \bigcirc_o A \quad \Sigma_1; \Sigma_2; x : A \vdash F_1 \approx_\zeta F_2 \div_o C}{\Sigma_1; \Sigma_2; \cdot \vdash \mathsf{let\ val}\ x = M_1\ \mathsf{in}\ F_1 \approx_\zeta \mathsf{let\ val}\ x = M_2\ \mathsf{in}\ F_2 \div_o C}\ (54)$$

By pattern matching, $E_i = \mathsf{let\ val}\ x = M_i\ \mathsf{in}\ F_i$

Consider the evaluation rule for $(H_1, \Sigma_1, E_1) \to (H'_1, \Sigma'_1, E'_1)$:

- Subcase Letval1: $M_1 \to M'_1$, $E'_1 = \mathsf{let\ val}\ x = M'_1\ \mathsf{in}\ F_1$, $H'_1 = H_1$, $\Sigma'_1 = \Sigma_1$
  1. By Equivalent Values, M_2 is not a value

2. The only applicable evaluation rule for (H_2, Σ_2, E_2) is LETVAL1: $M_2 \rightarrow M_2'$, $E_2' = \mathsf{let\ val}\ x = M_2'\ \mathsf{in}\ F_2$, $H_2' = H_2$, $\Sigma_2' = \Sigma_2$
3. By Term Hexagon Lemma, there exist M_1'', M_2'' such that
   * $M_1' \rightarrow^* M_1''$, $M_2' \rightarrow M_2''$
   * $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : \bigcirc_o A$
4. Let $H_i'' = H_i$, $\Sigma_i'' = \Sigma_i$, $E_i'' = \mathsf{let\ val}\ x = M_i''\ \mathsf{in}\ F_i$, $U' = U$
5. By repeated application of LETVAL1, $(H_i', \Sigma_i', E_i') \rightarrow^* (H_i'', \Sigma_i'', E_i'')$
6. Evidently, $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^{U'} (H_2'' : \Sigma_2'')$
7. By rule (54), $\Sigma_1''; \Sigma_2; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$

– Subcase LETVAL: $M_1 = \mathsf{val}\ [V_1]$, $E_1' = F_1[V_1/x]$, $H_1' = H_1$, $\Sigma_1' = \Sigma_1$
  By Equivalent Values, M_2 is a value.
  Two applicable evaluation rules for $(H_2, \Sigma_2, E_2) \rightarrow (H_2', \Sigma_2', E_2')$
  * Subcase LETVALVAL: Symmetric to a case below, omitted here.
  * Subcase LETVAL: $M_2 = \mathsf{val}\ [V_2]$, $E_2' = F_2[V_2/x]$, $H_2' = H_2$, $\Sigma_2' = \Sigma_2$
    By Equivalent Term Inversion, two subcases
    ⋆ $\Sigma_i; \cdot \vdash \mathsf{val}\ [V_i] : B$ and $\vdash B \leq \bigcirc_o A$ and $\vdash B \nearrow a$ and $a \not\sqsubseteq \zeta$
    1. By Subtyping Inversion, $B = \bigcirc_{o'} A'$ and $\vdash A' \leq A$, $o' \preceq o$ where $o' = (r', w')$
    2. By Informativeness Inversion, $a \sqsubseteq w' \sqcap b$ and $\vdash A' \nearrow b$
    3. Since $a \not\sqsubseteq \zeta$, $w' \not\sqsubseteq \zeta$, $b \not\sqsubseteq \zeta$
    4. By Inversion, $\Sigma_i; \cdot \vdash [V_i] \div_{o_i} B_i$, $\vdash \bigcirc_{o_i} B_i \leq \bigcirc_{o'} A'$
    5. By Inversion, $\Sigma_i; \cdot \vdash V_i : B_i$
    6. By Subtyping Inversion, $\vdash B_i \leq A'$ and $o_i \preceq o'$
    7.
    $$\dfrac{\dfrac{\Sigma_i''; \cdot \vdash V_i : B_i \quad \vdash B_i \leq A'}{\Sigma_i; \cdot \vdash V_i : A'}\,(28)\ \text{for } i = 1,2 \qquad \begin{matrix}\vdash A' \nearrow b \\ b \not\sqsubseteq \zeta\end{matrix}}{\Sigma_1; \Sigma_2; \cdot \vdash V_1 \approx_\zeta V_2 : A'}\,(39)$$
    8. By rule (49), $\Sigma_1; \Sigma_2; \cdot \vdash V_1 \approx_\zeta V_2 : A$
    9. By Functionality, $\Sigma_1; \Sigma_2; \cdot \vdash F_1[V_1/x] \approx_\zeta F_2[V_2/x] \div_o C$
    10. Let $H_i'' = H_i' = H_i$, $\Sigma_i'' = \Sigma_i' = \Sigma_i$, $E_i'' = E_i'$, $U' = U$ for $i = 1, 2$.

    ⋆ $\Sigma_1; \Sigma_2; \cdot \vdash [V_1] \approx_\zeta [V_2] \div_{o'} B$, $\vdash \bigcirc_{o'} B \leq \bigcirc_o A$
    1. By Equivalent Expression Inversion, $\Sigma_1; \Sigma_2; \cdot \vdash V_1 \approx_\zeta V_2 : B$
    2. By Subtyping Inversion, $\vdash B \leq A$ and $o' \preceq o$
    3. By (49), $\Sigma_1; \Sigma_2; \cdot \vdash V_1 \approx_\zeta V_2 : A$
    4. By Functionality, $\Sigma_1; \Sigma_2; \cdot \vdash F_1[V_1/x] \approx_\zeta F_2[V_2/x] \div_o C$
    5. Let $H_i'' = H_i' = H_i$, $\Sigma_i'' = \Sigma_i' = \Sigma_i$, $U' = U$, $E_i'' = E_i'$

– Subcase LETVALVAL: $M_1 = \mathsf{val}\ E_{11}$, $E'_1 = \mathsf{let\ val}\ x = \mathsf{val}\ E'_{11}\ \mathsf{in}\ F_1$, $(H_1, \Sigma_1, E_{11}) \to (H'_1, \Sigma'_1, E'_{11})$
By Equivalent Values, M_2 is a value
There are two applicable evaluation rules for $(H_2, \Sigma_2, E_2) \to (H'_2, \Sigma'_2, E'_2)$
  * Subcase LETVALVAL: $M_2 = \mathsf{val}\ E_{21}$, $E'_2 = \mathsf{let\ val}\ x = \mathsf{val}\ E'_{21}\ \mathsf{in}\ F_2$, $(H_2, \Sigma_2, E_{21}) \to (H'_2, \Sigma'_2, E'_{21})$
    By Equivalent Term Inversion on $\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : \bigcirc_o A$, there are two possibilities:
    ⋆ $\Sigma_1; \Sigma_2; \cdot \vdash E_{11} \approx_\zeta E_{21} \div_{o'} B$, $\vdash \bigcirc_{o'} B \leq \bigcirc_o A$
    1. By IH, there exist $H''_1, H''_2, \Sigma''_1, \Sigma''_2, E''_{11}, E''_{21}$ such that

$$(H'_i, \Sigma'_i, E'_{i1}) \to^* (H''_i, \Sigma''_i, E''_{i1}) \text{ for } i = 1, 2$$

$$\vdash (H''_1 : \Sigma''_1) \approx_\zeta^{U'} (H''_2 : \Sigma''_2) \text{ where } U' = \mathrm{dom}(\Sigma''_1) \cap \mathrm{dom}(\Sigma''_2) \cap \downarrow(\zeta)$$

$$\Sigma''_1; \Sigma''_2; \cdot \vdash E''_{11} \approx_\zeta E''_{21} \div_{o'} B$$

    2. Let $E''_i = \mathsf{let\ val}\ x = \mathsf{val}\ E''_{i1}\ \mathsf{in}\ F_i$, $S''_i = (H''_i, \Sigma''_i, E''_i)$ for $i = 1, 2$
    3. By repeated application of LETVALVAL, $S'_i \to^* S''_i$
    4. By Store Size, $\Sigma''_i \supseteq \Sigma_i$, for $i = 1, 2$
    5. By Simultaneous Store Weakening, $\Sigma''_1; \Sigma''_2; x : A \vdash F_1 \approx_\zeta F_2 \div_o C$
    6.

$$\cfrac{\cfrac{\Sigma''_1; \Sigma''_2; \cdot \vdash E''_{11} \approx_\zeta E''_{21} \div_{o'} B}{\Sigma''_1; \Sigma''_2; \cdot \vdash \mathsf{val}\ E''_{11} \approx_\zeta \mathsf{val}\ E''_{21} : \bigcirc_{o'} B}\ (48) \qquad \vdash \bigcirc_{o'} B \leq \bigcirc_o A}{\Sigma''_1; \Sigma''_2; \cdot \vdash \mathsf{val}\ E''_{11} \approx_\zeta \mathsf{val}\ E''_{21} : \bigcirc_o A}\ (49)$$

    7. By rule (54), $\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C$
    ⋆ $\Sigma; \Gamma \vdash M_i : B$, and $\vdash B \leq \bigcirc_o A$ (where $M_i = \mathsf{val}\ E_{i1}$, for $i = 1, 2$) and $\vdash B \nearrow a$ with $a \not\sqsubseteq \zeta$
    1. By Subtyping Inversion, $B = \bigcirc_{o'} A'$ and $\vdash A' \leq A$ and $o' \preceq o$ where $o' = (r', w')$
    2. By Informativeness Inversion, $a \sqsubseteq w' \sqcap b$ and $\vdash A' \nearrow b$
    3. Evidently, $w' \not\sqsubseteq \zeta$, $b \not\sqsubseteq \zeta$
    4. By Subterm Termination, $(H_i, \Sigma_1, E_{i1}) \to^+ (H''_i, \Sigma''_i, [V_i])$
    5. By HSS,

$$\vdash (H''_1 : \Sigma''_1) \approx_\zeta^{U} (H''_2 : \Sigma''_2)$$

$$U = \mathrm{dom}(H''_1) \cap \mathrm{dom}(H''_2) \cap \downarrow(\zeta)$$

    6. By repeated application of LETVALVAL, $S'_i \to^* (H''_i, \Sigma''_i, \mathsf{let\ val}\ x = \mathsf{val}\ [V_i]\ \mathsf{in}\ F_i)$ for $i = 1, 2$
    7. And by an application of LETVAL, $(H''_i, \Sigma''_i, \mathsf{let\ val}\ x = \mathsf{val}\ [V_i]\ \mathsf{in}\ F_i) \to (H''_i, \Sigma''_i, F_i[V_i/x])$ for $i = 1, 2$

8. Let $E_i'' = F_i[V_i/x]$ for $i = 1, 2$
9. By repeatedly applying Preservation, $\Sigma_i''; \cdot \vdash [V_i] \div_o A'$
10. By Inversion, $\Sigma_i''; \cdot \vdash V_i : A'$
11. By rule (39), $\Sigma_1''; \Sigma_2''; \cdot \vdash V_1 \approx_\zeta V_2 : A'$
12. By rule (49), $\Sigma_1''; \Sigma_2''; \cdot \vdash V_1 \approx_\zeta V_2 : A$
13. By Store Size, $\Sigma_i'' \supseteq \Sigma_i$ for $i = 1, 2$
14. By Simultaneous Store Weakening, $\Sigma_1''; \Sigma_2''; x : A \vdash F_1 \approx_\zeta F_2 \div_o C$
15. By Functionality, $\Sigma_1''; \Sigma_2''; \vdash E_1'' \approx_\zeta E_2'' \div_o C$

$*$ Subcase Letval: $M_2 = \mathsf{val}\ [V_2]$, $E_2' = F_2[V_2/x]$, $H_2' = H_2$, $\Sigma_2' = \Sigma_2$

By Equivalent Term Inversion, there are two possibilities

$\star$ $\Sigma_1; \Sigma_2; \cdot \vdash E_{11} \approx_\zeta [V_2] \div_{o'} B$ and $\vdash \bigcirc_{o'} B \leq \bigcirc_o A$

1. By Symmetry and Equivalent Expression Inversion, it follows that $E_{11} = [M_{11}]$ and $\Sigma_2; \Sigma_1; \cdot \vdash V_2 \approx_\zeta M_{11} : B$
2. Recall that $(H_1, \Sigma_1, \mathsf{let\ val}\ x = \mathsf{val}\ E_{11}\ \mathsf{in}\ F_1) \to (H_1', \Sigma_1', \mathsf{let\ val}\ x = \mathsf{val}\ E_{11}'\ \mathsf{in}\ F_1)$, by Letvalval, that is $(H_1, \Sigma_1, E_{11}) \to (H_1', \Sigma_1', E_{11}')$
3. $M_{11} \to M_{11}'$, since the only applicable evaluation rule for $(H_1, \Sigma_1, [M_{11}])$ is Ret1
4. By Equivalent Values, M_{11} is a value
5. But this is a contradiction, since values do not evaluate

$\star$ $\Sigma_1; \cdot \vdash \mathsf{val}\ E_{11} : B$, $\Sigma_2; \cdot \vdash \mathsf{val}\ [V_2] : B$, and $\vdash B \leq \bigcirc_o A$, $\vdash B \nearrow a$ and $a \not\sqsubseteq \zeta$

1. By Subtyping Inversion, $B = \bigcirc_{o'} A'$ and $\vdash A' \leq A$, $o' \preceq o$ where $o' = (r', w')$
2. By Inversion, $\Sigma_1; \cdot \vdash E_{11} \div_{o'} A'$ and $\Sigma_2; \cdot \vdash [V_2] \div_{o'} A'$
3. By Informativeness Inversion, $a \sqsubseteq w' \sqcap b$ and $\vdash A' \nearrow b$
4. Evidently $w' \not\sqsubseteq \zeta$ and $b \not\sqsubseteq \zeta$
5. By Subterm Termination, since $S_1' \downarrow$, $(H_1, \Sigma_1, E_{11}) \to^+ (H_1'', \Sigma_1'', [V_1])$
6. Let $H_2'' = H_2' = H_2$, $\Sigma_2'' = H_2' = \Sigma_2$
7. Trivially, $(H_2, \Sigma_2, [V_2]) \to^0 (H_2'', \Sigma_2'', [V_2])$
8. By HSS,
$$\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$$
$$U = \mathrm{dom}(\Sigma_1'') \cap \mathrm{dom}(\Sigma_2'') \cap {\downarrow}(\zeta)$$
9. By repeated application of Letvalval, $S_1' \to^* (H_1'', \Sigma_1'', \mathsf{let\ val}\ x = \mathsf{val}\ [V_1]\ \mathsf{in}\ F_1)$
10. By an application of Letval, $(H_i'', \Sigma_i'', \mathsf{let\ val}\ x = \mathsf{val}\ [V_i]\ \mathsf{in}\ F_i) \to (H_i'', \Sigma_i'', F_i[V_i/x])$ for $i = 1, 2$

11. Let $E_i'' = F_i[V_i/x]$ for $i = 1, 2$
12. By repeatedly applying Preservation, $\Sigma_i''; \cdot \vdash [V_i] \div_{o'} A'$ for $i = 1, 2$
13. By Inversion, $\Sigma_i''; \cdot \vdash V_i : A'$
14. By rule (39), $\Sigma_1''; \Sigma_2''; \cdot \vdash V_1 \approx_\zeta V_2 : A'$
15. By rule (49), $\Sigma_1''; \Sigma_2''; \cdot \vdash V_1 \approx_\zeta V_2 : A$
16. By Store Size, $\Sigma_i'' \supseteq \Sigma_i$, for $i = 1, 2$
17. By Simultaneous Store Weakening, $\Sigma_1''; \Sigma_2''; x : A \vdash F_1 \approx_\zeta F_2 \div_o C$
18. By Functionality, $\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$

- Case

$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : A}{\Sigma_1; \Sigma_2; \cdot \vdash \mathsf{ref}_a\,(M_1 : A) \approx_\zeta \mathsf{ref}_a\,(M_2 : A) \div_{(\bot,\top)} \mathsf{ref}_a\,A}\ (55)$$

By pattern matching, $E_i = \mathsf{ref}_a\,(M_i : A)$, $o = (\bot, \top)$, $C = \mathsf{ref}_a\,A$

There are two possible evaluation rules for $(H_1, \Sigma_1, E_1) \to (H_1', \Sigma_1', E_1')$

– Subcase Ref1: $H_1' = H_1$, $\Sigma_1' = \Sigma_1$, $E_1' = \mathsf{ref}_a\,(M_1' : A)$ $M_1 \to M_1'$

1. By Equivalent Values, M_2 is not a value.
2. There is a single applicable rule, Ref1, for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$: $H_2' = H_2$, $\Sigma_2' = \Sigma_2$, $E_2' = \mathsf{ref}_a\,(M_2' : A)$, $M_2 \to M_2'$.
3. By Term Hexagon Lemma, there exist M_1'', M_2'' such that $M_i' \to M_i''$ and $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : A$
4. Let $H_i'' = H_i' = H_i$, $\Sigma_i'' = \Sigma_i' = \Sigma_i$, $U' = U$, $E_i'' = \mathsf{ref}_a\,(M_i'' : A)$ for $i = 1, 2$,
5. Evidently $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^{U'} (H_2'' : \Sigma_2'')$
6. By repeated application of Ref1, $(H_i', \Sigma_i', E_i') \to^* (H_i'', \Sigma_i'', E_i'')$ for $i = 1, 2$
7. By rule (55), $\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$

– Subcase Ref: M_1 value, $H_1' = H_1\{\ell_1 \mapsto M_1\}$, $\Sigma_1' = \Sigma_1\{\ell_1 : A\}$, $E_1' = [\ell_1]$, where $\ell_1 \notin \mathrm{dom}(H_1)$, $\mathsf{Level}(\ell_1) = a$

1. By Equivalent Values, M_2 is a value
2. Only Ref rule is applicable to $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$: $H_2' = H_2\{\ell_2 \mapsto M_2\}$, $\Sigma_2' = \Sigma_2\{\ell_2 : A\}$, $E_2' = [\ell_2]$, where $\ell_2 \notin \mathrm{dom}(H_2)$, $\mathsf{Level}(\ell_2) = a$
3. Consider two subcases now, either $a \sqsubseteq \zeta$ or $a \not\sqsubseteq \zeta$:
   - Subcase $a \sqsubseteq \zeta$

   (a) WLOG, we may α-vary S_1', S_2' such that $\ell_1 = \ell_2 = \ell$

   (b) Then $\mathsf{Level}(\ell) = a$, $\ell \notin \mathrm{dom}(H_1) \cup \mathrm{dom}(H_2)$

   (c) Let $H_i'' = H_i'$, $\Sigma_i'' = \Sigma_i'$, $E_i'' = E_i' = [\ell]$

(d)

$$\cfrac{\cfrac{\mathsf{Level}(\ell) = a \sqsubseteq \zeta \quad \Sigma''_1(\ell) = \Sigma''_2(\ell) = A}{\Sigma''_1; \Sigma''_2; \cdot \vdash \ell \approx_\zeta \ell : \mathsf{ref}_a\, A}\ (47)}{\Sigma''_1; \Sigma''_2; \cdot \vdash E''_1 \approx_\zeta E''_2 \div_o C}\ (55)$$

(e)

$$\begin{aligned} U' &= \mathrm{dom}(\Sigma''_1) \cap \mathrm{dom}(\Sigma''_2) \cap {\downarrow}(\zeta) \\ &= (\mathrm{dom}(\Sigma'_1) \cup \{\ell\}) \cap (\mathrm{dom}(\Sigma'_2) \cup \{\ell\}) \cap {\downarrow}(\zeta) \\ &= ((\mathrm{dom}(\Sigma'_1) \cap \mathrm{dom}(\Sigma'_2)) \cup \{\ell\}) \cap {\downarrow}(\zeta) \\ &= U \cup \{\ell\} \end{aligned}$$

(f) Evidently, $\Sigma''_1 \restriction U' = \Sigma''_2 \restriction U'$

(g) By Simultaneous Store Weakening, $\Sigma''_1; \Sigma''_2; \cdot \vdash M_1 \approx_\zeta M_2 : A$

(h) By Simultaneous Store Weakening, $\Sigma''_1; \Sigma''_2; \cdot \vdash H''_1(\ell') \approx_\zeta H''_2(\ell') : \Sigma_1(\ell')$ for $\ell' \in U$

(i) By Regularity of Equivalence, Store Weakening, and rule (37), $\vdash H''_i : \Sigma''_i$ for $i = 1, 2$

(j) By rule (58), $\vdash (H''_1 : \Sigma''_1) \approx_\zeta^{U''} (H''_2 : \Sigma''_2)$

$*$ Subcase $a \not\sqsubseteq \zeta$

(a) Let $H''_i = H'_i$, $\Sigma''_i = \Sigma'_i$, $E''_i = E'_i = [\ell_i]$ for $i = 1, 2$

(b) By Regularity of Equivalence, $\vdash H_i : \Sigma_i$ for $i = 1, 2$

(c) By Store Weakening, $\Sigma''_i; \cdot \vdash \ell_i : A$ for $i = 1, 2$

(d) By rule (37), $\vdash H''_i : \Sigma''_i$ for $i = 1, 2$

(e) Evidently $\mathrm{dom}(\Sigma''_1) \cap \mathrm{dom}(\Sigma''_2) \cap {\downarrow}(\zeta) = U$, since $\ell_i \notin {\downarrow}(\zeta)$ for $i = 1, 2$

(f) So $\Sigma''_i \restriction U = \Sigma_i \restriction U$ for $i = 1, 2$

(g) By Store Weakening, for all $\ell \in U$, $\Sigma''_1; \Sigma''_2; \cdot \vdash H''_1(\ell) \approx_\zeta H''_2(\ell) : \Sigma''_1(\ell)$

(h)

$$\cfrac{\begin{array}{l} \vdash H''_1 : \Sigma''_1 \\ \vdash H''_2 : \Sigma''_2 \\ \Sigma''_1 \restriction U = \Sigma''_2 \restriction U \\ \Sigma''_1; \Sigma''_2; \cdot \vdash H''_1(\ell) \approx_\zeta H''_2(\ell) : \Sigma''_1(\ell) \text{ for all } \ell \in U \end{array}}{\vdash (H''_1 : \Sigma''_1) \approx_\zeta^{U} (H''_2 : \Sigma''_2)}\ (58)$$

(i)

$$\cfrac{\cfrac{\begin{array}{l} \Sigma''_1; \cdot \vdash \ell_1 : \mathsf{ref}_a\, A \\ \Sigma''_2; \cdot \vdash \ell_2 : \mathsf{ref}_a\, A \end{array} \quad \cfrac{}{\vdash \mathsf{ref}_a\, A \nearrow a}\ (5) \quad a \not\sqsubseteq \zeta}{\Sigma''_1; \Sigma''_2; \cdot \vdash \ell_1 \approx_\zeta \ell_2 : \mathsf{ref}_a\, A}\ (39)}{\Sigma''_1; \Sigma''_2; \cdot \vdash [\ell_1] \approx_\zeta [\ell_2] \div_o \mathsf{ref}_a\, A}\ (53)$$

- Case

$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : \mathsf{refr}_a\, C}{\Sigma_1; \Sigma_2; \cdot \vdash !M_1 \approx_\zeta !M_2 \div_{(a,\top)} C}\ (56)$$

By pattern matching, $E_i = !M_i$, $o = (r, w) = (a, \top)$. Recall that $a = r \sqsubseteq \zeta$

There are two applicable rules for $(H_1, \Sigma_1, E_1) \to (H_1', \Sigma_1', E_1')$

  – Subcase Bang1: $H_1' = H_1$, $\Sigma_1' = \Sigma_1$, $E_1' = !M_1'$, where $M_1 \to M_1'$

    1. By Equivalent Values, M_2 is not a value
    2. There is only a single applicable rule for $S_2 \to S_2'$, Bang1: $H_2' = H_2$, $\Sigma_2' = \Sigma_2$, $E_2' = !M_2'$, $M_2 \to M_2'$
    3. By the Term Hexagon Lemma, there exist M_1'', M_2'' such that $M_1' \to^* M_1''$ and $M_2' \to^* M_2''$ with $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : \mathsf{refr}_a\, C$
    4. Let $S_i'' = (H_i'', \Sigma_i'', E_i'')$, $H_i'' = H_i'$, $\Sigma_i'' = \Sigma_i'$, $E_i'' = !M_i''$ for $i = 1, 2$
    5. By repeated application of Bang1, $S_i' \to^* S_i''$ for $i = 1, 2$
    6. Evidently, $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$
    7. By rule (56), $\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$

  – Subcase Bang: $M_1 = \ell_1$, $H_1' = H_1$, $\Sigma_1' = \Sigma_1$, $E_1' = [H_1(\ell_1)]$

    1. By Equivalent Values, M_2 is a value
    2. The single applicable evaluation rule for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$ is Bang: $M_2 = \ell_2$, $H_2' = H_2$, $\Sigma_2' = \Sigma_2$, $E_2' = [H_2(\ell_2)]$
    3. Let $H_i'' = H_i' = H_i$, $\Sigma_i'' = \Sigma_i' = \Sigma_i$, $U'' = U$, $E_i'' = E_i'$ for $i = 1, 2$
    4. Evidently, $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^{U''} (H_2'' : \Sigma_2'')$
    5. By Equivalent Term Inversion, there are two possibilities:
       * Either $\Sigma_i; \cdot \vdash \ell_i : B$ and $\vdash B \leq \mathsf{refr}_a\, C$ and $\vdash B \nearrow b$ and $b \not\sqsubseteq \zeta$
       (a) By Subtyping Inversion, either $B = \mathsf{refr}_{b'}\, B'$ or $B = \mathsf{ref}_{b'}\, B'$ and in either case $\vdash B' \leq C$, $b' \sqsubseteq a$
       (b) In either case, by Informativeness Inversion, $\vdash B' \nearrow c$ and $b' \sqsubseteq a \sqcup c$
       (c) Since $a \sqsubseteq \zeta$, but $b' \not\sqsubseteq \zeta$, then $c \not\sqsubseteq \zeta$
       (d) By Inversion, $\vdash \mathsf{ref}_{\mathsf{Level}(\ell_i)}\, \Sigma_i(\ell_i) \leq B$
       (e) By Subtyping Inversion, $\mathsf{Level}(\ell_i) \sqsubseteq b'$ and $\vdash \Sigma_i(\ell_i) \leq B'$
       (f) By Regularity of Equivalence , and store typing $\Sigma_i''; \cdot \vdash H_i(\ell_i) : \Sigma_i''(\ell_i)$
       (g) By rule (28), $\Sigma_i''; \cdot \vdash H_i(\ell_i) : B'$
       (h) By rule (39), $\Sigma_1''; \Sigma_2''; \cdot \vdash H_1(\ell_1) \approx_\zeta H_2(\ell_2) : B'$
       (i) By rule (49), $\sigma_1''; \Sigma_2''; \cdot \vdash H_1(\ell_1) \approx_\zeta H_2(\ell_2) : C$
       * Or $\ell_1 = \ell_2 = \ell$ where $\mathsf{Level}(\ell) = b \sqsubseteq \zeta$, and $\vdash \mathsf{ref}_b\, \Sigma_1(\ell) \leq \mathsf{refr}_a\, C$ and $\Sigma_1(\ell) = \Sigma_2(\ell)$

(a) $\ell \in U$

(b) By Subtyping Inversion, $b \sqsubseteq a$ and $\vdash \Sigma_1(\ell) \leq C$

(c) From equivalent store typing,
$\Sigma_1; \Sigma_2; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : \Sigma_1(\ell)$

(d)

$$\frac{\Sigma_1''; \Sigma_2''; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : \Sigma_1(\ell) \quad \vdash \Sigma_1(\ell) \leq C}{\Sigma_1''; \Sigma_2''; \cdot \vdash H_1(\ell) \approx_\zeta H_2(\ell) : C} \; (49)$$

6.

$$\frac{\dfrac{\Sigma_1''; \Sigma_2''; \cdot \vdash H_1(\ell_1) \approx_\zeta H_2(\ell_2) : C}{\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_{(\perp,\top)} C} \; (53) \quad (\perp, \top) \preceq o}{\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C} \; (50)$$

- Case

$$\frac{\Sigma_1; \Sigma_2; \cdot \vdash M_1 \approx_\zeta M_2 : \mathsf{refw}_a\, A \quad \Sigma_1; \Sigma_2; \cdot \vdash N_1 \approx_\zeta N_2 : A}{\Sigma_1; \Sigma_2; \cdot \vdash M_1 := N_1 \approx_\zeta M_2 := N_2 \div_{(\perp,a)} 1} \; (57)$$

By pattern matching, $E_i = M_i := N_i$, $o = (r, w) = (\perp, a)$, $C = 1$

There are three applicable rules for $(H_1, \Sigma_1, E_1) \to (H_1', \Sigma_1', E_1')$

– Subcase Assn1: $H_1' = H_1$, $\Sigma_1' = \Sigma_1$, $E_1' = M_1' := N_1$, $M_1 \to M_1'$

1. By Equivalent Values, M_2 is not a value
2. There is a single applicable evaluation rule for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$, Assn1: $H_2' = H_2$, $\Sigma_2' = \Sigma_2$, $E_2' = M_2' := N_2$, $M_2 \to M_2'$
3. By the Term Hexagon Lemma, there exist M_1'', M_2'' such that $M_1' \to^* M_1''$ and $M_2' \to^* M_2''$ and $\Sigma_1; \Sigma_2; \cdot \vdash M_1'' \approx_\zeta M_2'' : \mathsf{refw}_a\, A$
4. Let $S_i'' = (H_i'', \Sigma_i'', E_i'')$, $H_i'' = H_i'$, $\Sigma_i'' = \Sigma_i'$, $E_i'' = M_i'' := N_i$ for $i = 1, 2$
5. By repeated application of Assn1, $S_i' \to^* S_i''$ for $i = 1, 2$
6. Evidently, $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$
7. By rule (57), $\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$

– Subcase Assn2: M_1 value, $H_1' = H_1$, $\Sigma_1' = \Sigma_1$, $E_1' = M_1 := N_1'$ $N_1 \to N_1'$

1. By Equivalent Values, M_2 is a value
2. By Equivalent Values, N_2 is not a value
3. There is a single applicable evaluation rule for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$, Assn2: $H_2' = H_2$, $\Sigma_2' = \Sigma_2$, $E_2' = M_2 := N_2'$, $N_2 \to N_2'$
4. By the Term Hexagon Lemma, there exist N_1'', N_2'' such that $N_1' \to^* N_1''$ and $N_2' \to^* N_2''$, and $\Sigma_1; \Sigma_2; \cdot \vdash N_1'' \approx_\zeta N_2'' : A$

5. Let $S_i'' = (H_i'', \Sigma_i'', E_i'')$, $H_i'' = H_i'$, $\Sigma_i'' = \Sigma_i'$, $E_i'' = M_i := N_i''$ for $i = 1, 2$
6. By repeated application of Assn1, $S_i' \to^* S_i''$ for $i = 1, 2$
7. Evidently, $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$
8. By rule (57), $\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$

– Subcase Assn: $M_1 = \ell_1$, N_1 value, $H_1' = H_1\{\ell_1 \mapsto N_1\}$, $\Sigma_1' = \Sigma_1$, $E_1' = [*]$

1. By Equivalent Values, M_2 is a value
2. By Equivalent Values, N_2 is a value
3. There is a single applicable evaluation rule for $(H_2, \Sigma_2, E_2) \to (H_2', \Sigma_2', E_2')$, Assn: $M_2 = \ell_2$, $H_2' = H_2\{\ell_2 \mapsto N_2\}$, $\Sigma_2' = \Sigma_2$, $E_2' = [*]$
4. Let $H_i'' = H_i'$, $\Sigma_i'' = \Sigma_i' = \Sigma_i$, $E_i'' = E_i' = [*]$
5. $$\dfrac{\dfrac{\dfrac{}{\Sigma_1''; \Sigma_2''; \cdot \vdash * \approx_\zeta * : 1}\,(40)}{\Sigma_1''; \Sigma_2''; \cdot \vdash [*] \approx_\zeta [*] \div_{(\bot,\top)} 1}\,(53) \qquad (\bot, \top) \preceq o}{\Sigma_1''; \Sigma_2''; \cdot \vdash [*] \approx_\zeta [*] \div_o 1}\,(50)$$
6. By Equivalent Term Inversion, there are two possibilities:
   * Either $\Sigma_i; \cdot \vdash \ell_i : B$ and $\vdash B \leq \mathsf{refw}_a\, A$, $\vdash B \nearrow b$ and $b \not\sqsubseteq \zeta$
     By Regularity of Equivalence, $\Sigma_i; \cdot \vdash \ell_i : B$
     By Inversion, $\vdash \mathsf{ref}_{\mathsf{Level}(\ell_i)}\, \Sigma_i(\ell_i) \leq B$ for $i = 1, 2$
     By Subtyping Inversion, either $B = \mathsf{refw}_{b'}\, B'$ or $B = \mathsf{ref}_{b'}\, B'$ and in either case $\vdash A \leq B'$ and $a \sqsubseteq b'$
     ⋆ If $B = \mathsf{refw}_{b'}\, B'$
     (a) By Subtyping Inversion, $\vdash B' \leq \Sigma_i(\ell_i)$ and $b' \sqsubseteq \mathsf{Level}(\ell_i)$ for $i = 1, 2$
     (b) By Informativeness Inversion, if $B = \mathsf{refw}_{b'}\, B'$ then $b \sqsubseteq b'$
     (c) By Regularity of Equivalence, $\Sigma_i; \cdot \vdash N_i : A$ for $i = 1, 2$
     (d) By rule (28), $\Sigma_i; \cdot \vdash N_i : \Sigma_i(\ell_i)$ for $i = 1, 2$
     (e) By Store Update, $\vdash H_i \ell_i \mapsto N_i : \Sigma_i$ for $i = 1, 2$
     (f) Since $b \sqsubseteq \mathsf{Level}(\ell_i)$ and $b \not\sqsubseteq \zeta$, then $\mathsf{Level}(\ell_i) \not\sqsubseteq \zeta$ so $\ell_i \notin U$
     (g) Therefore, for all $\ell \in U$,
     $\Sigma_1''; \Sigma_2''; \cdot \vdash H_1''(\ell) \approx_\zeta H_2''(\ell) : \Sigma_1''(\ell)$
     (h) So by rule (58), $\vdash (H_1'' : \Sigma_i'') \approx_\zeta^U (H_2'' : \Sigma_2'')$
     ⋆ If $B = \mathsf{ref}_{b'}\, B'$
     (a) By Subtyping Inversion, $B' = \Sigma_i(\ell_i)$ and $b' = \mathsf{Level}(\ell_i)$ for $i = 1, 2$
     (b) By Informativeness Inversion, $b \sqsubseteq b' \sqcup c$ and $\vdash B' \nearrow c$

(c) Since $b \not\sqsubseteq \zeta$, either $b' \not\sqsubseteq \zeta$ or $c \not\sqsubseteq \zeta$

(d) If $b' \not\sqsubseteq \zeta$, we can use the same argument as the previous subcase: $B = \mathsf{refw}_{b'}\ B'$. So suppose instead $c \not\sqsubseteq \zeta$

(e) Note that we can further suppose that $b' \sqsubseteq \zeta$ (if not, same argument as previous subcase)

(f) Consider ℓ_1 (the argument for ℓ_2 is symmetric)

(g) Evidently $\mathsf{Level}(\ell_1) = b' \sqsubseteq \zeta$, so suppose $\ell_1 \in U$ (if not, same argument as previous subcase)

(h) If $\ell_1 = \ell_2$ then note that we're in the next subcase ($\ell_1 = \ell_2 = \ell$, ...) below; so suppose ℓ_1 differs from ℓ_2

(i) So $\ell_1 \in \mathrm{dom}(\Sigma_2) = \mathrm{dom}(H_2)$

(j) By heap typing inversion, $\Sigma_2; \cdot \vdash H_2(\ell_1) : \Sigma_2(\ell_1)$

(k) Since $\ell_1 \in U$, $\Sigma_2(\ell_1) = \Sigma_1(\ell_1) = B'$

(l) By rule (39), $\Sigma_1; \Sigma_2; \cdot \vdash N_1 \approx_\zeta H_2(\ell_1) : \Sigma_1(\ell_1)$

(m) Therefore for all $\ell \in U$, $\Sigma_1''; \Sigma_2''; \cdot \vdash H_1''(\ell) \approx_\zeta H_2''(\ell) : \Sigma_1''(\ell)$

(n) So by rule (58), $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$

* Or $\ell_1 = \ell_2 = \ell$ and $\mathsf{Level}(\ell) \sqsubseteq \zeta$ and $\Sigma_1(\ell) = \Sigma_2(\ell)$, and $\vdash \mathsf{ref}_{\mathsf{Level}(\ell)}\ \Sigma_1(\ell) \leq \mathsf{refw}_a\ A$

(a) By Subtyping Inversion, $\vdash A \leq \Sigma_1(\ell)$, and $a \sqsubseteq \mathsf{Level}(\ell)$

(b) Evidently $\ell \in U$

(c) By rule (49), $\Sigma_1''; \Sigma_2''; \cdot \vdash N_1 \approx_\zeta N_2 : \Sigma_1(\ell)$

(d) By Regularity, $\Sigma_i''; \cdot \vdash N_i : \Sigma_i(\ell)$ for $i = 1, 2$

(e) By Store Update, $\vdash H_i \ell_i \mapsto N_i : \Sigma_i''$ for $i = 1, 2$

(f) By rule (58), $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^U (H_2'' : \Sigma_2'')$

Given $\vdash (H_1'' : \Sigma_1'') \approx_\zeta^{U'} (H_2'' : \Sigma_2'')$ where $U' = \mathrm{dom}(\Sigma_1'') \cap \mathrm{dom}(\Sigma_2'') \cap \downarrow(\zeta)$ and $\Sigma_1''; \Sigma_2''; \cdot \vdash E_1'' \approx_\zeta E_2'' \div_o C$, by rule (59), $\vdash S_1'' \approx_\zeta S_2'' \div_o C$ which proves the lemma. $\square$

D The SLam Calculus

D.1 Operational semantics

$\boxed{e \to e'}$

$$\frac{e_1 \to e_1'}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \to \mathsf{if}\ e_1'\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3}$$

$$\frac{}{\mathsf{if}\ \mathsf{true}_a\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \to e_2}$$

$$\frac{}{\mathsf{if}\ \mathsf{false}_a\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \to e_3}$$

$$\frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \qquad \frac{e_2 \to e_2'}{v\ e_2 \to v\ e_2'}$$

$$\frac{}{(\lambda x : s.e)_a\ v \to e[v/x]}$$

D.2 Typing rules

$\boxed{s \leq s'}$

$$\frac{s_1 \leq s_2 \quad s_2 \leq s_3}{s_1 \leq s_3}$$

$$\frac{a \sqsubseteq a'}{(1, a) \leq (1, a')} \qquad \frac{a \sqsubseteq a'}{(\mathsf{bool}, a) \leq (\mathsf{bool}, a')}$$

$$\frac{s_1' \leq s_1 \quad s_2 \leq s_2' \quad a \sqsubseteq a'}{(s_1 \to s_2, a) \leq (s_1' \to s_2', a')}$$

$\boxed{\Gamma \vdash e : s}$

We use the notation $(t, a) \sqcup a'$ for $(t, a \sqcup a')$, in the typing rules, below.

$$\frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{}{\Gamma \vdash *_a : (1, a)}$$

$$\frac{}{\Gamma \vdash \mathsf{true}_a : (\mathsf{bool}, a)} \qquad \frac{}{\Gamma \vdash \mathsf{false}_a : (\mathsf{bool}, a)}$$

$$\frac{\Gamma \vdash e_1 : (\mathsf{bool}, a) \quad \Gamma \vdash e_2 : s \quad \Gamma \vdash e_3 : s}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : s \sqcup a}$$

$$\frac{\Gamma, x : s_1 \vdash e : s_2}{\Gamma \vdash (\lambda x : s_1.e)_a \vdash (s_1 \to s_2, a)}$$

$$\frac{\Gamma \vdash e_1 : (s_1 \to s, a) \quad \Gamma \vdash e_2 : s_1}{\Gamma \vdash e_1\ e_2 : s \sqcup a}$$

$$\frac{\Gamma \vdash e : s}{\Gamma \vdash \mathsf{protect}_a e : s \sqcup a} \qquad \frac{\Gamma \vdash e : s \quad s \leq s'}{\Gamma \vdash e : s'}$$

D.3 Type-correct translation proof

Type Correct Translation If $\Gamma \vdash e : s \Rightarrow E$ then $\{\}; \overline{\Gamma} \vdash E \div_{(\bot,\top)} \overline{s}$

Proof. by induction on the given derivation. By cases on the last rule used.

- Case
$$\frac{\Gamma \vdash e : s \Rightarrow E}{\Gamma \vdash \mathsf{protect}_a e : s \sqcup a \Rightarrow E}$$

  1. By pattern matching, $s = (t, a')$, so $s \sqcup a = (t, a' \sqcup a)$
  2. By IH, $\{\}; \overline{\Gamma} \vdash E \div_{(\bot,\top)} \mathsf{refr}_{a'}\, \overline{t}$
  3. By rule (17), $\vdash \mathsf{refr}_a\, \overline{t} \leq \mathsf{refr}_{a' \sqcup a}\, \overline{t}$
  4. By rule (36), $\{\}; \overline{\Gamma} \vdash E \div_{(\bot,\top)} \overline{s \sqcup a}$

- Case
$$\frac{\Gamma \vdash e_1 : (s \to s', a) \Rightarrow E_1 \quad \Gamma \vdash e_2 : s \Rightarrow E_2}{\Gamma \vdash e_1 e_2 : s' \sqcup a \Rightarrow \begin{array}{l}\mathsf{let\ val}\ y_1 = \mathsf{val}\ E_1\ \mathsf{in}\\ \mathsf{let\ val}\ y_2 = \mathsf{val}\ !y_1\ \mathsf{in}\\ \mathsf{let\ val}\ y_3 = \mathsf{val}\ E_2\ \mathsf{in}\\ \mathsf{let\ val}\ y_4 = y_2 y_3\ \mathsf{in}\ [y_4]\end{array}}$$

  1. By pattern matching, $E = \ldots$
  2. By IH, $\{\}; \overline{\Gamma} \vdash E_1 \div_{(\bot,\top)} \mathsf{refr}_a \left(\overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}\right)$
  3. By IH, $\{\}; \overline{\Gamma} \vdash E_2 \div_{(\bot,\top)} \overline{s}$
  4. By rule (27), $\{\}; \overline{\Gamma} \vdash \mathsf{val}\ E_1 : \bigcirc_{(\bot,\top)} \mathsf{refr}_a \left(\overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}\right)$
  5. Let $\Gamma_1 = \overline{\Gamma}, y_1 : \mathsf{refr}_a \left(\overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}\right)$
  6. $$\cfrac{\cfrac{\cfrac{}{\{\}; \Gamma_1 \vdash y_1 : \mathsf{refr}_a\, \overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}}\ (20)}{\{\}; \Gamma_1 \vdash !y_1 \div_{(a,\top)} \overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}}\ (32)}{\{\}; \Gamma_1 \vdash \mathsf{val}\ !y_1 : \bigcirc_{(a,\top)} \overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}}\ (27)$$
  7. Let $\Gamma_2 = \Gamma_1, y_2 : \overline{s} \to \bigcirc_{(\bot,\top)} \overline{s'}$
  8. By Context Weakening, $\{\}; \Gamma_2 \vdash E_2 \div_{(\bot,\top)} \overline{s}$
  9. $$\cfrac{\cfrac{\{\}; \Gamma_2 \vdash E_2 \div_{(\bot,\top)} \overline{s}}{\{\}; \Gamma_2 \vdash E_2 \div_{(a,\top)} \overline{s}}\ (34)}{\{\}; \Gamma_2 \vdash \mathsf{val}\ E_2 : \bigcirc_{(a,\top)} \overline{s}}\ (27)$$
  10. Let $\Gamma_3 = \Gamma_2, y_3 : \overline{s}$
  11. By rule (26), $\{\}; \Gamma_3 \vdash y_2 y_3 : \bigcirc_{(\bot,\top)} \overline{s'}$
  12. By rule (28), $\{\}; \Gamma_3 \vdash y_2 y_3 : \bigcirc_{(a,\top)} \overline{s'}$
  13. Let $\Gamma_4 = \Gamma_3, y_4 : \overline{s'}$

14. By repeated rule (30), $\{\};\overline{\Gamma} \vdash E \div_{(a,\top)} \overline{s'}$
15. By pattern matching, $s' = (t', a')$ and $\overline{s'} = \mathsf{refr}_{a'}\,\overline{t'}$
16. By rule (17), $\vdash \mathsf{refr}_{a'}\,\overline{t'} \leq \mathsf{refr}_{a' \sqcup a}\,\overline{t'}$
17. Note that $\overline{s' \sqcup a} = \overline{(t', a' \sqcup a)} = \mathsf{refr}_{a' \sqcup a}\,\overline{t'}$
18.
$$\dfrac{\dfrac{}{\vdash \mathsf{refr}_{a' \sqcup a}\,\overline{t'} \nearrow a' \sqcup a}\,(8) \qquad a \sqsubseteq a' \sqcup a}{\vdash \mathsf{refr}_{a' \sqcup a}\,\overline{t'} \nearrow a}\,(10)$$
19. By rule (36), $\{\};\overline{\Gamma} \vdash E \div_{(a,\top)} \overline{s' \sqcup a}$
20. By rule (35), $\{\};\overline{\Gamma} \vdash E \div_{(\perp,\top)} \overline{s' \sqcup a}$

- Other cases similarly.

□

E $\lambda^{\mathrm{REF}}_{\mathrm{SEC}}$ well-typed translation proof

Well-typed Translation

1. If $\Sigma;\Gamma \vdash bv : t \Rightarrow M$ then $\overline{\Sigma};\overline{\Gamma} \vdash M : \overline{t}$
2. If $\Sigma;\Gamma[\mathsf{pc}] \vdash e : s \Rightarrow E$ then $\overline{\Sigma};\overline{\Gamma} \vdash E \div_{(\perp,\mathsf{pc})} \overline{s}$

Proof. both parts simultaneously, by induction on the given derivations. By cases on the last rule used.

Part (1)

- The cases for unit and boolean values, and store locations are immediate.
- Case
$$\dfrac{\Sigma;\Gamma, x : s_1[\mathsf{pc}] \vdash e : s_2 \Rightarrow E}{\Sigma;\Gamma \vdash \lambda[\mathsf{pc}]x : s_1.e : s_1 \xrightarrow{\mathsf{pc}} s_2 \Rightarrow \lambda x : \overline{s_1}.\mathsf{val}\ E}$$

  1. By IH,
  $$\overline{\Sigma};\overline{\Gamma}, x : \overline{s_1} \vdash E \div_{(\perp,\mathsf{pc})} \overline{s_2}$$
  2.
  $$\dfrac{\dfrac{\overline{\Sigma};\overline{\Gamma}, x : \overline{s_1} \vdash E \div_{(\perp,\mathsf{pc})} \overline{s_2}}{\overline{\Sigma};\overline{\Gamma}, x : \overline{s_1} \vdash \mathsf{val}\ E : \bigcirc_{(\perp,\mathsf{pc})}\overline{s_2}}}{\overline{\Sigma};\overline{\Gamma} \vdash \lambda x : \overline{s_1}.\mathsf{val}\ E : \overline{s_1 \xrightarrow{\mathsf{pc}} s_2}}$$

- The case for subsumption follows by well-typed type translation

Part (2)

- Case

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{bool}, a) \Rightarrow E_1 \quad \Sigma;\Gamma[\mathsf{pc} \sqcup a] \vdash e_2 : s \Rightarrow E_2 \quad \Sigma;\Gamma[\mathsf{pc} \sqcup a] \vdash e_3 : s \Rightarrow E_3}{\Sigma;\Gamma[\mathsf{pc}] \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : s \Rightarrow \begin{array}{l}\mathsf{let}\ y = E_1\ \mathsf{in} \\ \mathsf{let}\ y' = !y\ \mathsf{in} \\ \ \ \mathsf{run}\quad \begin{array}{l}\mathsf{if}\ y' \\ \mathsf{then\ val}\ E_2 \\ \mathsf{else\ val}\ E_3\end{array}\end{array}}$$

1. By IH, $\overline{\Sigma};\overline{\Gamma} \vdash E_1 \div_{(\bot,\mathsf{pc})} \mathsf{refr}_a\,\mathsf{bool}$, and $\overline{\Sigma};\overline{\Gamma} \vdash E_i \div_{(\bot,\mathsf{pc}\sqcup a)} \overline{s}$ for $i = 2, 3$
2. Let $\Gamma_1 = \overline{\Gamma}, y : \mathsf{refr}_a\,\mathsf{bool}$
3. By rule (32), $\overline{\Sigma};\Gamma_1 \vdash !y \div_{(a,\top)} \mathsf{bool}$
4. Let $\Gamma_2 = \Gamma_1, y' : \mathsf{bool}$
5. $$\dfrac{\dfrac{\dfrac{}{\overline{\Sigma};\Gamma_2 \vdash y' : \mathsf{bool}}\ (20) \qquad \dfrac{}{\overline{\Sigma};\Gamma_2 \vdash \mathsf{val}\ E_i : \bigcirc_{(\bot,\mathsf{pc}\sqcup a)}\overline{s}}\ (27)\ \text{for } i = 2, 3}{\overline{\Sigma};\Gamma_2 \vdash \mathsf{if}\ y'\ \mathsf{then\ val}\ E_2\ \mathsf{else\ val}\ E_3 : \bigcirc_{(\bot,\mathsf{pc}\sqcup a)}\overline{s}}\ (23)}{\overline{\Sigma};\Gamma_2 \vdash \mathsf{run\ if}\ y'\ \mathsf{then\ val}\ E_2\ \mathsf{else\ val}\ E_3 \div_{(\bot,\mathsf{pc}\sqcup a)} \overline{s}}$$
6. We can promote the operation levels of $!y$ and $\mathsf{run}\ \ldots$ to $(a, \mathsf{pc} \sqcup a)$, such that
$$\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in\ run}\ \ldots \div_{(a,\mathsf{pc}\sqcup a)} \overline{s}$$
7. Let $(t, b) = s$, and note that $\overline{s} = \mathsf{refr}_b\,\overline{t}$.
8. By lemma 6.4, $a \sqsubseteq b$. Hence $\vdash \overline{s} \nearrow a$.
9. Therefore,
$$\dfrac{\dfrac{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in\ run}\ \ldots \div_{(a,\mathsf{pc}\sqcup a)} \overline{s} \quad \vdash \overline{s} \nearrow a}{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in\ run}\ \ldots \div_{(\bot,\mathsf{pc}\sqcup a)} \overline{s}}\ (35)}{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in\ run}\ \ldots \div_{(\bot,\mathsf{pc})} \overline{s}}\ (34)$$
10. Therefore,
$$\overline{\Sigma};\overline{\Gamma} \vdash \mathsf{let}\ y = E_1\ \mathsf{in\ let}\ y' = !y\ \mathsf{in\ run}\ \ldots \div_{(\bot,\mathsf{pc})} \overline{s}$$

- Case

$$\frac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (s' \xrightarrow{\mathsf{pc}'} s, a) \Rightarrow E_1 \quad \Sigma;\Gamma[\mathsf{pc}] \vdash e_2 : s' \Rightarrow E_2 \quad \mathsf{pc} \sqcup a \sqsubseteq \mathsf{pc}'}{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 e_2 : s \sqcup a \Rightarrow \begin{array}{l}\mathsf{let}\ y_1 = E_1\ \mathsf{in} \\ \mathsf{let}\ y_2 = E_2\ \mathsf{in} \\ \mathsf{let}\ y_1' = !y_1\ \mathsf{in} \\ \mathsf{run}\ (y_1' y_2)\end{array}}$$

1. By IH,
$$\overline{\Sigma}; \overline{\Gamma} \vdash E_1 \div_{(\bot,\mathsf{pc})} \mathsf{refr}_a\, \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s}$$
2. By IH,
$$\overline{\Sigma}; \overline{\Gamma} \vdash E_2 \div_{(\bot,\mathsf{pc})} \overline{s'}$$
3. Let $\Gamma_1 = \overline{\Gamma}, y_1 : \mathsf{refr}_a\, \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s}, y_2 : \overline{s'}$
4. Let $(t, b) = s$. Then $s \sqcup a = (t, a \sqcup b)$ and $\overline{s \sqcup a} = \mathsf{refr}_{a \sqcup b}\, \overline{t}$.
5. Since $\vdash \overline{s} \leq \overline{s \sqcup a}$, $\vdash \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s} \leq \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a}$
6. $$\frac{\overline{\Sigma}; \Gamma_1 \vdash y_1 : \mathsf{refr}_a\, \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a}}{\overline{\Sigma}; \Gamma_1 \vdash !y_1 \div_{(a,\top)} \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a}}\ (32)$$
7. Let $\Gamma_2 = \Gamma_1, y_1' : \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a}$
8. Since $a \sqsubseteq \mathsf{pc}'$, $a \sqsubseteq (a \sqcup b) \sqcap \mathsf{pc}'$, so
$$\frac{\dfrac{\dfrac{\vdash \mathsf{refr}_{a \sqcup b}\, \overline{t} \nearrow a \sqcup b}{\vdash \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a} \nearrow (a \sqcup b) \sqcap \mathsf{pc}'}}{\vdash \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a} \nearrow a} \qquad a \sqsubseteq (a \sqcup b) \sqcap \mathsf{pc}'}{\vdash \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a} \nearrow a}$$
9. Therefore,
$$\overline{\Sigma}; \Gamma_1 \vdash !y_1 \div_{(\bot,\top)} \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a}$$
10. So
$$\frac{\overline{\Sigma}; \Gamma_2 \vdash !y_1 \div_{(\bot,\mathsf{pc}')} \overline{s'} \to \bigcirc_{(\bot,\mathsf{pc}')}\overline{s \sqcup a} \qquad \dfrac{\overline{\Sigma}; \Gamma_2 \vdash \mathsf{run}\ (y_1' y_2) \div_{(\bot,\mathsf{pc}')} \overline{s}}{\overline{\Sigma}; \Gamma_2 \vdash y_1' y_2 : \bigcirc_{(\bot,\mathsf{pc}')}\overline{s}}}{\overline{\Sigma}; \Gamma_1 \vdash \mathsf{let}\ y_1' = !y_1\ \mathsf{in}\ \mathsf{run}\ (y_1' y_2) \div_{(\bot,\mathsf{pc}')} \mathsf{refr}_{a \sqcup b}\, \overline{t}}$$
11. Since $\mathsf{pc} \sqsubseteq \mathsf{pc}'$,
$$\overline{\Sigma}; \overline{\Gamma} \vdash \mathsf{let}\ y_1 = E_1\ \mathsf{in}\ \mathsf{let}\ y_2 = E_2\ \mathsf{in}\ \mathsf{let}\ y_1' = !y_1\ \mathsf{in}\ \mathsf{run}\ (y_1' y_2) \div_{(\bot,\mathsf{pc})} \overline{s \sqcup a}$$

- Case
$$\frac{\Sigma; \Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{ref}\ s, a) \Rightarrow E_1}{\Sigma; \Gamma[\mathsf{pc}] \vdash !e_1 : s \sqcup a \Rightarrow \mathsf{let}\ y = E_1\ \mathsf{in}\ \mathsf{let}\ y' = !y\ \mathsf{in}\ !y'}$$

1. Let $(t, b) = s$. Note $s \sqcup a = (t, a \sqcup b)$, $\overline{\mathsf{ref}\ s} = \mathsf{ref}_b\, \overline{s}$
2. By IH, $\overline{\Sigma}; \overline{\Gamma} \vdash E_1 \div_{(\bot,\mathsf{pc})} \mathsf{refr}_a\, \mathsf{ref}_b\, \overline{s}$
3. Let $\Gamma_1 = \overline{\Gamma}, y : \mathsf{refr}_a\, \mathsf{ref}_b\, \overline{s}$
4. By rule (32),
$$\overline{\Sigma}; \Gamma_1 \vdash !y \div_{(a,\top)} \mathsf{ref}_b\, \overline{s}$$

5.
$$\dfrac{\dfrac{\overline{\Sigma};\Gamma_1 \vdash !y \div_{(a,\top)} \mathsf{ref}_b\,\overline{s} \quad (a,\top) \preceq (a \sqcup b,\top)}{\overline{\Sigma};\Gamma_1 \vdash !y \div_{(a\sqcup b,\top)}} \quad \dfrac{\overline{\Sigma};\Gamma_1, y' : \mathsf{ref}_b\,\overline{s} \vdash !y' \div_{(b,\top)} \overline{s} \quad (b,\top) \preceq (a \sqcup b,\top)}{\overline{\Sigma};\Gamma_1, y' : \mathsf{ref}_b\,\overline{s} \vdash !y' \div_{(a\sqcup b,\top)} \overline{s}}}{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in}\ !y' \div_{(a\sqcup b,\top)} \overline{s}}$$

6. Since $\overline{s} = \overline{(t,b)} = \mathsf{refr}_b\,\overline{t}$,

$$\dfrac{\dfrac{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in}\ !y' \div_{(a\sqcup b,\top)} \overline{s} \quad \vdash \overline{s} \leq \mathsf{refr}_{a\sqcup b}\,\overline{t}}{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in}\ !y' \div_{(a\sqcup b,\top)} \mathsf{refr}_{a\sqcup b}\,\overline{t}} \quad \vdash \mathsf{refr}_{a\sqcup b}\,\overline{t} \nearrow a \sqcup b}{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in}\ !y' \div_{(\perp,\top)} \mathsf{refr}_{a\sqcup b}\,\overline{t}}$$

7.
$$\dfrac{\overline{\Sigma};\overline{\Gamma} \vdash E_1 \div_{(\perp,\mathsf{pc})} \mathsf{refr}_a\,\mathsf{ref}_b\,\overline{s} \quad \dfrac{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in}\ !y' \div_{(\perp,\top)} \mathsf{refr}_{a\sqcup b}\,\overline{t} \quad (\perp,\top) \preceq (\perp,\mathsf{pc})}{\overline{\Sigma};\Gamma_1 \vdash \mathsf{let}\ y' = !y\ \mathsf{in}\ !y' \div_{(\perp,\mathsf{pc})} \mathsf{refr}_{a\sqcup b}\,\overline{t}}}{\overline{\Sigma};\overline{\Gamma} \vdash \mathsf{let}\ y = E_1\ \mathsf{in}\ \mathsf{let}\ y' = !y\ \mathsf{in}\ y' \div_{(\perp,\mathsf{pc})} \mathsf{refr}_{a\sqcup b}\,\overline{t}}$$

•

$$\dfrac{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 : (\mathsf{ref}\ (t,b), a) \Rightarrow E_1 \quad \Sigma;\Gamma[\mathsf{pc}] \vdash e_2 : (t,b) \Rightarrow E_2 \quad a \sqsubseteq b}{\Sigma;\Gamma[\mathsf{pc}] \vdash e_1 := e_2 : (1,\mathsf{pc}) \Rightarrow \begin{array}{l}\mathsf{let}\ y_1 = E_1\ \mathsf{in}\\ \mathsf{let}\ y_2 = E_2\ \mathsf{in}\\ \mathsf{let}\ y_1' = !y_1\ \mathsf{in}\\ \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in}\\ \mathsf{ref}_{\mathsf{pc}}\,(* : 1)\end{array}}$$

1. By IH,
$$\overline{\Sigma};\overline{\Gamma} \vdash E_1 \div_{(\perp,\mathsf{pc})} \mathsf{refr}_a\,\mathsf{ref}_b\,\mathsf{refr}_b\,\overline{t}$$
and
$$\overline{\Sigma};\overline{\Gamma} \vdash E_2 \div_{(\perp,\mathsf{pc})} \mathsf{refr}_b\,\overline{t}$$
2. By Lemma 6.4, $\mathsf{pc} \sqsubseteq b$
3. Let $\Gamma_1 = \overline{\Gamma}, y_1 : \mathsf{refr}_a\,\mathsf{ref}_b\,\mathsf{refr}_b\,\overline{t}, y_2 : \mathsf{refr}_b\,\overline{t}$
4.
$$\dfrac{\overline{\Sigma};\Gamma_1 \vdash !y_1 \div_{(a,\top)} \mathsf{ref}_b\,\mathsf{refr}_b\,\overline{t} \quad (a,\top) \preceq (a,b)}{\overline{\Sigma};\Gamma_1 \vdash !y_1 \div_{(a,b)} \mathsf{ref}_b\,\mathsf{refr}_b\,\overline{t}}$$
Note that (a,b) is a well-formed operation level since $a \sqsubseteq b$
5. Let $\Gamma_2 = \Gamma_1, y_1' : \mathsf{ref}_b\,\mathsf{refr}_b\,\overline{t}$

$$\dfrac{\dfrac{\overline{\Sigma};\Gamma_2 \vdash y_1' := y_2 \div_{(\perp,b)} 1 \quad \overline{\Sigma};\Gamma_2 \vdash \mathsf{ref}_{\mathsf{pc}}\,(* : 1) \div_{(\perp,\top)} \mathsf{refr}_{\mathsf{pc}}\,1}{\overline{\Sigma};\Gamma_2 \vdash \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in}\ \mathsf{ref}_{\mathsf{pc}}\,(* : 1) \div_{(\perp,b)} \mathsf{refr}_{\mathsf{pc}}\,1} \quad (\perp,b) \preceq (a,b)}{\overline{\Sigma};\Gamma_2 \vdash \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in}\ \mathsf{ref}_{\mathsf{pc}}\,(* : 1) \div_{(a,b)} \mathsf{refr}_{\mathsf{pc}}\,1}$$

6.

$$\dfrac{\dfrac{\begin{array}{l}\overline{\Sigma}; \Gamma_1 \vdash !y_1 \div_{(a,b)} \mathsf{ref}_b\, \mathsf{refr}_b\, \overline{t} \\ \overline{\Sigma}; \Gamma_2 \vdash \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in}\ \mathsf{ref}_{\mathsf{pc}}\, (* : 1) \div_{(a,b)} \mathsf{refr}_{\mathsf{pc}}\, 1\end{array}}{\overline{\Sigma}; \Gamma_1 \vdash \begin{array}{l}\mathsf{let}\ y_1' = !y_1\ \mathsf{in} \\ \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in}\ \mathsf{ref}_{\mathsf{pc}}\, (* : 1) \div_{(a,b)} \mathsf{refr}_{\mathsf{pc}}\, 1\end{array}} \quad \dfrac{\vdash 1 \nearrow a}{\vdash \mathsf{refr}_{\mathsf{pc}}\, 1 \nearrow a}}{\overline{\Sigma}; \Gamma_1 \vdash \begin{array}{l}\mathsf{let}\ y_1' = !y_1\ \mathsf{in} \\ \mathsf{let}\ \_ = y_1' := y_2\ \mathsf{in}\ \mathsf{ref}_{\mathsf{pc}}\, (* : 1) \div_{(\bot,b)} \mathsf{refr}_{\mathsf{pc}}\, 1\end{array}}$$

7. Since $\mathsf{pc} \sqsubseteq b$,

$$\overline{\Sigma}; \overline{\Gamma} \vdash \mathsf{let}\ y_1 = E_1\ \mathsf{in}\ \mathsf{let}\ y_2 = E_2\ \mathsf{in}\ \mathsf{let}\ y_1' = !y_1\ \mathsf{in}\ ... \div_{(\bot,\mathsf{pc})} \mathsf{refr}_{\mathsf{pc}}\, 1$$

- Other cases are similar.

$\square$