Erasable Contracts

Jao-ke Chin-Lee Louis Li
Harvard University
{jchinlee, louisli}@college.harvard.edu

Abstract

Contract programming is a design approach that allows programmers to design formal specifications for software, known as "contracts". These contracts, executed at runtime, allow programmers to make assertions about the behavior of their software and ensure program correctness. In a language with side-effects, however, it is possible for these contracts to modify memory and consequently change the behavior of the program. While contracts provide the specifications, these specifications clearly should not change the behavior of the program that they enforce. Instead, if the contracts were removed ("erased"), then the behavior of the program should be the same as if the contracts remained. This notion is captured by the idea of *erasability*. We present a calculus for erasable contracts, establish properties such as type soundness, and prove a formal definition of erasability for the language.

1. Introduction

Contract programming is a design approach that allows programmers to design formal specifications for software. These contracts are associated with certain portions of the software, such as methods, and allow programmers to:

1. Ensure correctness of program execution
2. Assign blame to the "violator" of a contract
3. Conveniently place assertions near the code

Contracts execute at run-time. Some possible uses for contracts might be to assert that, for a cursor on some data structure, the inserted item is truly in the data structure upon insertion. In order to take advantage of abstraction and code reuse, we would like to be able to call the already existing read functionality for the data structure within the contract code, which would manipulate the cursor. Another example is verifying the insertion of an object into a tree that self-balances according to accesses. In both of these cases, "reads" performed in contract code are not truly and purely reads into memory.

Therefore contract code can in fact read and write to memory, and it is possible for contracts to change the behavior of the program in a language with side-effects. Intuitively, however, a programmer would not want the specifications to change the behavior of the program.

In this work, we explore a formal model for *erasable* contracts, ensuring that contract code does not impact the behavior of the program. In other words, if the contracts were removed or "erased," then the execution of the program would remain unaffected. We discuss this notion more formally later in this paper.

Several implementations of contracts exist. Contracts gained traction through the Eiffel programming language [3]. The .NET Framework contains an implementation named "Code Contracts", allowing for a degree of static verification [2]. The Racket language provides support for contracts in a higher-order language [1].

We look at the behavior of contracts in more detail below.

Flat contracts Flat contracts apply assertions on a flat value, such as an integer or string. Intuitively, a flat contract has type:

$$\tau \to \mathsf{bool}$$

Function contracts *Function contracts* apply assertions on the behavior of a function. More specifically, they assert conditions about the pre- and post-conditions of a function. In other words, they are a function on the domain and range of a function.

Function contracts can be thought of as being composed of two flat contracts corresponding to the pre- and post-conditions. For example, the below could be

a function contract ensuring correct behavior for the `sqrt` function, checking that both the argument and result are non-negative:

$$\begin{aligned} \texttt{sqrt} &\quad \lambda x.\ \sqrt{x} \\ \texttt{preconditions} &\quad \lambda x.\ x \geq 0 \\ \texttt{postconditions} &\quad \lambda r.\ r \geq 0 \end{aligned}$$

Dependent contracts In *dependent contracts*, the argument to a function can be used in checking the post-conditions of a function. In our `sqrt` example, function correctness also depends on the value of the result: we may want to check that the result $r =^{?} \sqrt{x}$ satisfies that $r * r$ is approximately x. Thus, the post-condition takes an extra argument corresponding to the original argument of the function. Extending our previous example:

$$\begin{aligned} \texttt{sqrt} &\quad \ldots \\ \texttt{postconditions} &\quad \lambda x.\ \lambda r.\ (r \geq 0) \wedge (|(r * r) - x| \geq \epsilon) \end{aligned}$$

2. Contributions

This work consists of three main components:

1. Calculus and semantics for erasable contracts
2. Type system and proof of type soundness
3. Proof of erasability

In the following sections, we present our work. Full proofs can be found in the appendix.

3. Grammar

$$\begin{aligned} v ::= & n | \textsf{true} | \textsf{false} | \lambda x.\ e | \ell \\ e ::= & v | x | op(e...) | e\ e | e; e \\ & | \textsf{fix}\ e | \textsf{if}\ e\ \textsf{then}\ e\ \textsf{else}\ e \\ & | e := e | \textsf{ref}\ e |\ !\ell \\ & | \textsf{mon}(c, e) | \textsf{check}(e, v) \\ c ::= & \textsf{flat}(e) | c \to c \end{aligned}$$

$$\begin{aligned} E ::= & [\cdot] | op(v_0, \ldots, E, \ldots e_n) | E\ e | v\ E | E; e \\ & | \textsf{fix}\ E | \textsf{if}\ E\ \textsf{then}\ e\ \textsf{else}\ e \\ & | E := e | v := E | \textsf{ref}\ E \\ & | \textsf{mon}(e, E) | \textsf{mon}(E, v) \\ & | \textsf{flat}(E) \\ \tau ::= & \textsf{int} | \textsf{boolean} | \tau \to \tau | \tau\ \textsf{ref} | \textsf{con}(\tau) \end{aligned}$$

4. Semantics

Notice that all evaluations now occur at some depth δ. This corresponds to the number of nested contracts during the execution. In particularly, we see that the application of a contract by a *monitor*, which takes a contract and a value to be checked, enters a checked state through the rule MON-ENTER. Evaluation then occurs at a higher depth through the rule MON-CHECK. Finally, when the application of the contract evaluates to either *true* or *false*, then the monitor finishes through MON-EXIT.

A store is defined as:

$$\sigma : \textsf{Depth} \to \textsf{Var} \to \textsf{Value}$$

$$\textsc{Context}\ \frac{\delta \vdash \langle e, \sigma \rangle \longrightarrow \langle e', \sigma' \rangle}{\delta \vdash \langle E[e], \sigma \rangle \longrightarrow \langle E[e'], \sigma' \rangle}$$

$$\beta\textsc{-Reduction}\ \frac{}{\delta \vdash \langle (\lambda x.\ e) v, \sigma \rangle \longrightarrow \langle e\{v/x\}, \sigma \rangle}$$

$$\textsc{If-Then}\ \frac{}{\delta \vdash \langle \textsf{if true then}\ e_1\ \textsf{else}\ e_2, \sigma \rangle \longrightarrow \langle e_1, \sigma \rangle}$$

$$\textsc{If-Else}\ \frac{}{\delta \vdash \langle \textsf{if false then}\ e_1\ \textsf{else}\ e_2, \sigma \rangle \longrightarrow \langle e_2, \sigma \rangle}$$

$$\textsc{Seq}\ \frac{}{\delta \vdash \langle v; e, \sigma \rangle \longrightarrow \langle e, \sigma \rangle}$$

$$\textsc{Op}\ \frac{}{\delta \vdash \langle op(e \ldots), \sigma \rangle \longrightarrow \langle \Omega(op, e \ldots), \sigma \rangle}$$

where Ω is a function defined for each operation

$$\textsc{Deref}\ \frac{}{\delta \vdash \langle\ !\ell, \sigma \rangle \longrightarrow \langle v, \sigma \rangle}$$

where $\sigma[(\Delta, \ell)] = v$ for the first such $\Delta \in \delta, \delta - 1, \ldots, 0$

$$\textsc{Alloc}\ \frac{}{\delta \vdash \langle \textsf{ref}\ v, \sigma \rangle \longrightarrow \langle \ell, \sigma[(\delta, \ell) \mapsto v] \rangle}$$

$$\textsc{Assign}\ \frac{}{\delta \vdash \langle \ell := v, \sigma \rangle \longrightarrow \langle v, \sigma[(\delta, \ell) \mapsto v] \rangle}$$

$$\textsc{Mon}\ \frac{}{\delta \vdash \langle \textsf{mon}(v_1 \to v_2, v), \sigma \rangle \longrightarrow \langle \lambda x.\ \textsf{mon}(v_2, v\ \textsf{mon}(v_1, x)), \sigma \rangle}$$

x is a free variable

$$\textsc{Mon-Enter}\ \frac{}{\delta \vdash \langle \textsf{mon}(\textsf{flat}(v'), v), \sigma \rangle \longrightarrow \langle \textsf{check}(v'\ v, v), \sigma \rangle}$$

$$\textsc{Mon-Check}\ \frac{\delta + 1 \vdash \langle e, \sigma \rangle \longrightarrow \langle e', \sigma' \rangle}{\delta \vdash \langle \textsf{check}(e, v), \sigma \rangle \longrightarrow \langle \textsf{check}(e', v), \sigma' \rangle}$$

$$\textsc{Mon-Exit}\ \frac{}{\delta \vdash \langle \textsf{check}(\textsf{true}, v), \sigma \rangle \longrightarrow \langle v, \sigma[\delta + 1 \mapsto \emptyset] \rangle}$$

5. Type System

5.1 Values

$$\textsc{T-Int}\ \frac{}{\Gamma; \Sigma \vdash n : \textsf{int}}$$

$$\textsc{T-BoolT}\ \frac{}{\Gamma; \Sigma \vdash \textsf{true} : \textsf{boolean}}$$

$$\text{T-BoolF}\ \frac{}{\Gamma;\Sigma \vdash \textbf{false} : \textbf{boolean}}$$

$$\text{T-Loc}\ \frac{}{\Gamma;\Sigma \vdash \ell : \tau\ \textbf{ref}}\ \Sigma(\ell) = \tau$$

$$\text{T-Abs}\ \frac{\Gamma, x:\tau;\Sigma \vdash e:\tau'}{\Gamma;\Sigma \vdash \lambda x{:}\tau.\ e : \tau \to \tau'}$$

5.2 Expressions

$$\text{T-Assign}\ \frac{\Gamma;\Sigma \vdash e_1 : \tau\ \textbf{ref} \qquad \Gamma;\Sigma \vdash e_2 : \tau}{\Gamma;\Sigma \vdash e_1 := e_2 : \tau}$$

$$\text{T-Alloc}\ \frac{\Gamma;\Sigma \vdash e:\tau}{\Gamma;\Sigma \vdash \mathsf{ref}\ e : \tau\ \textbf{ref}}$$

$$\text{T-Deref}\ \frac{\Gamma;\Sigma \vdash e:\tau\ \textbf{ref}}{\Gamma;\Sigma \vdash !e:\tau}$$

$$\text{T-Cond}\ \frac{\Gamma;\Sigma \vdash e_1 : \textbf{boolean} \qquad \Gamma;\Sigma \vdash e_2 : \tau \qquad \Gamma;\Sigma \vdash e_3 : \tau}{\Gamma;\Sigma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : \tau}$$

$$\text{T-Seq}\ \frac{\Gamma;\Sigma \vdash e_1 : \tau_1 \qquad \Gamma;\Sigma \vdash e_2 : \tau_2}{\Gamma;\Sigma \vdash e_1; e_2 : \tau_2}$$

$$\text{T-App}\ \frac{\Gamma;\Sigma \vdash e_1 : \tau \to \tau' \qquad \Gamma;\Sigma \vdash e_2 : \tau}{\Gamma;\Sigma \vdash e_1\ e_2 : \tau'}$$

$$\text{T-Fix}\ \frac{\Gamma;\Sigma \vdash e:\tau \to \tau}{\Gamma;\Sigma \vdash \mathsf{fix}\ e : \tau}$$

$$\text{T-Mon}\ \frac{\Gamma;\Sigma \vdash e:\tau \qquad \Gamma;\Sigma \vdash c : \mathsf{con}(\tau)}{\Gamma;\Sigma \vdash \mathsf{mon}(c,e):\tau}$$

$$\text{T-Flat}\ \frac{\Gamma;\Sigma \vdash e:\tau \to \textbf{boolean}}{\Gamma;\Sigma \vdash \mathsf{flat}(e) : \mathsf{con}(\tau)}$$

$$\text{T-Check}\ \frac{\Gamma;\Sigma \vdash v:\tau \qquad \Gamma;\Sigma \vdash e : \textbf{boolean}}{\Gamma;\Sigma \vdash \mathsf{check}(e,v):\tau}$$

$$\text{T-FunCon}\ \frac{\Gamma;\Sigma \vdash c_1 : \mathsf{con}(\tau_1) \qquad \Gamma;\Sigma \vdash c_2 : \mathsf{con}(\tau_2)}{\Gamma;\Sigma \vdash c_1 \to c_2 : \mathsf{con}(\tau_1 \to \tau_2)}$$

$$\text{T-Op}\ \frac{\Gamma;\Sigma \vdash e_1:\tau_1 \quad \ldots \quad \Gamma;\Sigma \vdash e_n:\tau_n \qquad \Gamma;\Sigma \vdash \Delta(op,\tau_1,\ldots,\tau_n):\tau}{\Gamma;\Sigma \vdash op(e_1,\ldots,e_n):\tau}$$

where Δ is a function defined for each operation

6. Type Soundness

Lemma 1. *(Preservation). If* $\Gamma;\Sigma \vdash e : \tau$ *and* $\delta \vdash \langle e, \sigma\rangle \longrightarrow \langle e'\sigma'\rangle$ *then there exists some* $\Sigma' \supseteq \Sigma$ *such that* $\Sigma', \Gamma' \vdash e' : \tau$ *and* $\Gamma, \Sigma' \vdash \sigma'$.

Lemma 2. *(Progress). Given* $e, \sigma, \delta, \Gamma, \Sigma, \tau$, *if* $\Gamma;\Sigma \vdash e:\tau$ *then* e *is a value,* $\exists e', \sigma' \ni \delta \vdash \langle e,\sigma\rangle \longrightarrow \langle e',\sigma'\rangle$, *or we throw an error.*

Lemma 3. *(Context well-typedness). If* $\Gamma;\Sigma \vdash e_1 : \tau'$ *and* $\Gamma;\Sigma \vdash E[e_1]:\tau$, *then for any* e_2 *where* $\Gamma;\Sigma \vdash e_2 : \tau'$, *we have* $\Gamma;\Sigma \vdash E[e_2]:\tau$.

Lemma 4. *(Type substitution). If* $x : \tau' \vdash e : \tau$ *and* $\vdash v:\tau'$ *then* $\vdash e\{v/x\}:\tau$.

7. Formal Erasability

Definition 1. (η-similarity). *Let* $=_\eta$ *be the smallest equivalence relation such that*

i $e_1 \equiv e_2 \Rightarrow e_1 =_\eta e_2$;

ii $\lambda x.\ (e\ x) =_\eta e$ *for all* $x \notin FV(e)$ and e not a function of the form $\lambda y.\ (e'\ y)$;

iii $e_1 =_\eta e_2 \Rightarrow E[e_1] =_\eta E[e_2]$

For notational convenience, we denote $\langle e,\sigma\rangle =_\eta \langle e',\sigma\rangle$ if $e =_\eta e'$.

Definition 2 (Erasability)**.** Let $erase$ be defined recursively and separately for expressions and stores as follows:

$$erase(e) = \begin{cases} erase(e'), & e \equiv \mathsf{mon}(c,e') \\ erase(v), & e \equiv \mathsf{check}(e,v) \\ \text{homomorphic} & \text{otherwise} \end{cases}$$

$$erase(\sigma(\delta,x)) = \begin{cases} \emptyset & \delta \geq 1 \\ \sigma(\delta, erase(x)) & \delta = 0 \end{cases}$$

Then we say e is *erasable* if and only if given

$$0 \vdash \langle e,\sigma\rangle \longrightarrow^* \langle e',\sigma'\rangle$$

we have, for some $\hat{e}' =_\eta erase(e')$,

$$0 \vdash \langle erase(e), erase(\sigma)\rangle \longrightarrow^* \langle \hat{e}', erase(\sigma')\rangle$$

Lemma 5. *(Erasure substitution).* $\forall e, v, x,\ erase(e\{v/x\}) \equiv (erase(e))\{erase(v)/x\}$.

Lemma 6. *(Single-step erasability). Suppose* $0 \vdash \langle e,\sigma\rangle \longrightarrow \langle e',\sigma'\rangle$. *Then, for any* $\hat{e} =_\eta e$, $0 \vdash \langle erase(\hat{e}), erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$, *where* $e'' =_\eta erase(e')$.

Theorem 1. *(Erasability).* $0 \vdash \langle e,\sigma\rangle \longrightarrow^* \langle e',\sigma'\rangle \Rightarrow \forall \hat{e} =_\eta erase(e)\ \exists e'' =_\eta erase(e')$ *s.t.* $0 \vdash \langle \hat{e}, erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$

8. Future Work

Our work with erasable contracts can be extended in a number of ways. The calculus can be extended to include dependent contracts. Additionally, there may be times when having side-effecting contracts can be useful, allowing for "communicating contracts"– for example, a contract that asserts that function g only runs if function f has run previously, or a contract that only allows function f to run twice at most.

Acknowledgments

We would like to thank Stephen Chong and Christos Dimoulas for their invaluable guidance in our work.

References

[1] Christos Dimoulas, Robert Bruce Findler, Cormac Flanagan, and Matthias Felleisen. Correct blame for contracts: no more scapegoating. In *ACM SIGPLAN Notices*, volume 46, pages 215–226. ACM, 2011.

[2] Manuel Fähndrich. Static verification for code contracts. In *Static Analysis*, pages 2–5. Springer, 2011.

[3] Bertrand Meyer. *Eiffel: the language*. Prentice-Hall, Inc., 1992.

Appendix

1 Type Soundness

Lemma 1. *(Context well-typedness). If* $\Gamma; \Sigma \vdash e_1 : \tau'$ *and* $\Gamma; \Sigma \vdash E[e_1] : \tau$*, then for any* e_2 *where* $\Gamma; \Sigma \vdash e_2 : \tau'$*, we have* $\Gamma; \Sigma \vdash E[e_2] : \tau$.

Proof. Let e_1, e_2 be given such that $\Gamma; \Sigma \vdash e_1 : \tau'$ and $\Gamma; \Sigma \vdash e_2 : \tau'$, and suppose $\Gamma; \Sigma \vdash E[e_1] : \tau$. We consider cases of E.

- $op(v_0, \ldots, E, \ldots e_n)$: Consider the function Δ defined for op. If $\Gamma; \Sigma \vdash e_1 : \tau'$, then $\Delta(op, \tau_1, \ldots, \tau_n)\tau$ by T-OP. If $\Gamma; \Sigma \vdash e_2 : \tau'$, then $\Delta(op, \tau_1, \ldots, \tau_n)$ still evaluates to the same result, and thus $\Gamma; \Sigma \vdash op(e_1, \ldots, e_n) : \tau$.
- $E\ e'$: By T-APP, we must have that for some τ'', $\tau' = \tau'' \to \tau$, and $\Gamma; \Sigma \vdash e' : \tau''$. Applying T-APP for $e_2\ e'$, we see that $\Gamma; \Sigma \vdash e_2\ e' : \tau$, the same as $e_1\ e'$.
- $v\ E$: By T-APP, we must have that $\Gamma; \Sigma \vdash v : \tau' \to \tau$. Applying T-APP for $v\ e_2$, we see that $\Gamma; \Sigma \vdash v\ e_2 : \tau$, the same as $v\ e_1$.
- $E; e'$: By T-SEQ, we have that $\Gamma; \Sigma \vdash e' : \tau$. Consider any e_2 such that $\Gamma; \Sigma \vdash e_2 : \tau'$, and by T-SEQ, it follows that $\Gamma; \Sigma \vdash E[e_2] : \tau$ as desired.
- fix E By T-FIX, we have for some τ'', $\Gamma; \Sigma \vdash e_1 : \tau'' \to \tau''$ and $\Gamma; \Sigma \vdash E[e_1] : \tau''$.

  Therefore, $\tau' \equiv \tau'' \to \tau''$, $\tau \equiv \tau''$. Consider some e_2 such that $\Gamma; \Sigma \vdash e_2 : \tau'$, and it follows from T-FIX that $\Gamma; \Sigma \vdash E[e_2] : \tau$.
- **if** E **then** e' **else** e'': By T-COND, we must have that for some τ'', $\Gamma; \Sigma \vdash e', e'' : \tau'$, that $\tau =$ **boolean**, and that $\Gamma; \Sigma \vdash$ **if** E **then** e' **else** $e'' : \tau''$. Applying T-COND for **if** e_2 **then** e' **else** e'', we see that $\Gamma; \Sigma \vdash$ **if** e_2 **then** e' **else** $e'' : \tau''$, the same as **if** e_1 **then** e' **else** e''.
- $E := e'$: By T-ASSIGN, we must have that for some τ'', $\tau = \tau''$ **ref**, $\Gamma; \Sigma \vdash e' : \tau''$, and $\Gamma; \Sigma \vdash E := e' : \tau''$.
- $v := E$: From the statement of the lemma and T-ASSIGN, we have that $\Gamma; \Sigma \vdash v : \tau'$ **ref** and $\tau \equiv \tau'$.

  $$\Gamma; \Sigma \vdash e_1 : \tau$$

  $$\Gamma; \Sigma \vdash E[e_1] : \tau$$

  Consider some e_2 such that $\Gamma; \Sigma \vdash e_2 : \tau'$. By T-ASSIGN, we must have that $\Gamma; \Sigma \vdash E[e_2] : \tau$ as desired.
- ref E If $\Gamma; \Sigma \vdash e_1 : \tau'$ and $\Gamma; \Sigma \vdash E[e_1] : \tau$, then by T-REF, we have that $\tau \equiv tau'$ **ref**. Now consider any e_2 such that $\Gamma; \Sigma \vdash e_2 : \tau'$ and it follows from T-ALLOC that $\Gamma; \Sigma \vdash E[e_2] : \tau$ where $\tau \equiv \tau'$ **ref**.
- mon(e', E) If $\Gamma; \Sigma \vdash E[e_1] : \tau$, then by T-MON, $\Gamma; \Sigma \vdash e_1 :$ con(τ), $\tau' \equiv$ con(τ), and $\Gamma; \Sigma \vdash e' : \tau$. Consider some e_2 such that $\Gamma; \Sigma \vdash e_2 :$ con(τ), and it follows by T-MON that $\Gamma; \Sigma \vdash E[e_2] : \tau$.
- mon(E, v) If $\Gamma; \Sigma \vdash E[e_1] : \tau$, then by T-MON, $\Gamma; \Sigma \vdash e_1 : \tau$, $\tau' \equiv \tau$, and $\Gamma; \Sigma \vdash v :$ con(τ). Consider some e_2 such that $\Gamma; \Sigma \vdash e_2 : \tau$, and it follows by T-MON that $\Gamma; \Sigma \vdash E[e_2] : \tau$ as desired.
- flat(E) If $\Gamma; \Sigma \vdash e_1 : \tau'$, then by T-FLAT, $\Gamma; \Sigma \vdash E[e_1] :$ con(τ') and $\tau \equiv$ con(τ'). Consider some e_2 such that $\Gamma; \Sigma \vdash e_2 : \tau'$, then by T-FLAT again, $\Gamma; \Sigma \vdash E[e_2] :$ con(τ') as desired.

□

1.1 Preservation

Lemma 2. *(Preservation). If* $\Gamma; \Sigma \vdash e : \tau$ *and* $\delta \vdash \langle e, \sigma \rangle \longrightarrow \langle e'\sigma' \rangle$ *then there exists some* $\Sigma' \supseteq \Sigma$ *such that* $\Sigma', \Gamma' \vdash e' : \tau$ *and* $\Gamma, \Sigma' \vdash \sigma'$.

Proof. We proceed by induction on the derivation of $e \longrightarrow e'$. Assume that $\vdash e : \tau$ and $e \longrightarrow e'$.

- β-REDUCTION $e \equiv (\lambda x.\ e_1)\ v,$ and $e' \equiv e_1\{v/x\}$
  Assume that $(\lambda x.\ e_1)\ v$ is well-typed. The last rule used to derive this must have to been T-APP. It follows that $\vdash (\lambda x.\ e_1) : \tau \to \tau'$ and $\vdash v : \tau$.

  In order for $\vdash (\lambda x.\ e_1) : \tau \to \tau'$ to be true, the last rule used to derive this must have been T-ABS. It follows that $x : \tau \vdash e : \tau'$ by the substitution lemma.

- IF-THEN $e \equiv$ **if true then** e_1 **else** $e_2,$ and $e' \equiv e_1$
  Assume that **if true then** e_1 **else** e_2 is well-typed. The last rule used to derive this must have been T-COND, and we have that $\vdash e_1 : \tau$ as desired.

- IF-ELSE $e \equiv$ **if false then** e_1 **else** $e_2,$ and $e' \equiv e_2$
  Assume that **if false then** e_1 **else** e_2 is well-typed. The last rule used to derive this must have been T-COND, we have that $\vdash e_2 : \tau$ as desired.

- SEQ $e \equiv v; e_1,$ and $e' \equiv e_1$
  Assume that $v; e_1$ is well-typed. The last rule used to derive this must have been T-SEQ. It follows that $\vdash e_1 : \tau$ as desired.

- ASSIGN $e \equiv \ell := v,$ and $e' \equiv v$
  Assume that $\ell := v$ is well-typed. By T-ASSIGN, $\vdash v : \tau$ as desired.

- DEREF $e \equiv !\ell,$ and $e' \equiv v$
  Assume that $!\ell$ is well-typed. The last rule used to derive this must have been T-DEREF. Then ℓ has type τ **ref**. From the axiom T-LOC, we have that $\Sigma(\ell) = \tau$, from which it follows that dereferencing ℓ yields $\vdash v : \tau$.

- ALLOC $e \equiv \mathsf{ref}\ v,$ and $e' \equiv \ell$
  ℓ is well-typed by the axiom T-LOC.

- MON $e \equiv (\mathsf{mon}(v_1 \mapsto v_2, v))\ v',$ and $e' \equiv \mathsf{mon}(v_2, (v(\mathsf{mon}(v_1, v'))))$
  We know that e is well-typed, and the last rule used to derive this must have been T-APP. It follows that the both the mon and v' terms that compose e are well-typed.

  If $\mathsf{mon}(v_1 \mapsto v_2, v)$ is well-typed, it must be the case that T-MON was the last rule in the derivation. Then:

  - $\vdash v_1 \mapsto v_2 : \mathsf{con}(\tau_1 \to \tau)$. From this, we further conclude, using T-FUNCON, that $\vdash v_1 : \mathsf{con}(\tau_1)$ and $\vdash v_2 : \mathsf{con}(\tau)$.
  - Because the above term has type $\mathsf{con}(\tau_1 \to \tau)$, then by the hypothesis of T-MON, $\vdash v : \tau_1 \to \tau$.
  - $\vdash v' : \tau'$ by the hypothesis of T-MON

  Now, apply these conditions fulfill the hypotheses for the T-MON and T-APP rules such that $\vdash e' : \tau$.

- MON-ENTER $e \equiv \mathsf{mon}(\mathsf{flat}(v'), v),$ and $e' \equiv \mathsf{check}(v'\ v, v)$
  We know that $\vdash e : \tau$, and the last rule used to derive this must have been T-MON. It follows that $\vdash v : \tau$ and $\vdash flat(v') : \mathsf{con}(\tau$. By T-FLAT, this means that $\vdash v' : \tau \to$ **boolean**.

  Consider the rule T-CHECK. By T-APP, $\vdash v'\ v :$ **boolean**. We showed earlier that $\vdash v : \tau$, and it follows by the rule T-CHECK that $\vdash e' : \tau$.

- MON-CHECK $e \equiv \mathsf{check}(e_1, v)$, and $e' \equiv \mathsf{check}(e_2, v)$
  If $\vdash e : \tau$, then by T-CHECK, $\vdash e_1 : \textbf{boolean}$ and $\vdash v : \tau$. We need to show that $\vdash e_2 : \textbf{boolean}$. We can simply apply the preservation lemma, given that $\vdash e_1 : \textbf{boolean}$ and $e_1 \longrightarrow e_2$. This satisfies the hypothesis of T-CHECK and consequently $\vdash e' : \tau$.

- MON-EXIT $e \equiv \mathsf{check}(\textbf{true}, v)$, and $e' \equiv v$
  Assume that $\mathsf{check}(\textbf{true}, v)$ is well-typed. The last rule used to derive this must have been T-MON, and it follows that $\vdash v : \tau$.

□

1.2 Progress

Lemma 3. *(Progress). Given $e, \sigma, \delta, \Gamma, \Sigma, \tau$, if $\Gamma; \Sigma \vdash e : \tau$ then e is a value, $\exists e', \sigma' \ni \delta \vdash \langle e, \sigma \rangle \longrightarrow \langle e', \sigma' \rangle$, or we throw an error.*

Proof. We proceed by induction on the derivation of $\Gamma; \Sigma \vdash e : \tau$.

- T-INT, T-BOOLT, T-BOOLF
  e is trivially a value—n, **true**, or **false**, respectively—and σ is unchanged.

- T-LOC
  e is trivially a value—ℓ—and σ is unchanged.

- T-ABS
  e is trivially a value—$\lambda x.e'$—and σ is unchanged.

- T-ASSIGN
  $e \equiv e_1 := e_2$: if $\langle e_1, \sigma \rangle$ or $\langle e_2, \sigma \rangle$ can step to $\langle e', \sigma \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If both are already values, then since we also have that $\Gamma; \Sigma \vdash e_1 : \tau$ **ref** and is hence a location to which we can assign v_2, we have $e \equiv v_1 := v_2$ and $\delta \vdash \langle e, \sigma \rangle \longrightarrow \langle v_2, \sigma[(\delta, v_1) \mapsto v_2] \rangle$ by ASSIGN.

- T-ALLOC
  $e \equiv \mathsf{ref}\ \hat{e}$: if $\langle \hat{e}, \sigma \rangle$ can step to $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If $\hat{e} = v$ is a value, then we have $\delta \vdash \langle e, \sigma \rangle \equiv \langle \mathsf{ref}\ v, \sigma \rangle \longrightarrow \langle \ell, \sigma[(\delta, \ell) \mapsto v] \rangle$ by ALLOC.

- T-DEREF
  $e \equiv !\hat{e}$: if $\langle \hat{e}, \sigma \rangle$ can step to $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If $\hat{e} = v$ is a value, then since $\Gamma; \Sigma \vdash e : \tau$ **ref**, we know v is a location, and we have $\delta \vdash \langle e, \sigma \rangle \equiv \langle\ !v, \sigma \rangle \longrightarrow \langle v', \sigma \rangle$ where $\sigma[(\Delta), v] = v'$ for the first such $\Delta \in \delta, \delta - 1, \ldots, 0$.

- T-COND
  $e \equiv$ **if** e_1 **then** e_2 **else** e_3: by CONTEXT, we consider e_1. If $\langle e_1, \sigma \rangle$ can step to $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If $e' = v$ is a value, then we know that $\Gamma; \Sigma \vdash v : \textbf{boolean}$, and so is either **true**, in which case $\delta \vdash \langle e, \sigma \rangle \longrightarrow \langle e_2, \sigma' \rangle$, or **false**, in which case $\delta \vdash \langle e, \sigma \rangle \longrightarrow \langle e_3, \sigma' \rangle$.

- T-APP
  $e \equiv e_1\ e_2$: if $\langle e_1, \sigma \rangle$ or $\langle e_2, \sigma \rangle$ can step to an $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If both are values v_1 and v_2, respectively, then we have that $\Gamma; \Sigma \vdash v_1 : \tau \to \tau'$ and $\Gamma; \Sigma \vdash v_2 : \tau$, therefore we in fact can have $\delta \vdash \langle e, \sigma \rangle \equiv \langle \lambda(x.\ e) v, \sigma \rangle \longrightarrow \langle e\{v/x\}, \sigma \rangle$ by β-REDUCTION.

- T-FIX
  $e \equiv \mathsf{fix}\ \hat{e}$: if $\langle \hat{e}, \sigma \rangle$ can step to an $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If $\hat{e} = v$ is a value, then we know that $\Gamma; \Sigma \vdash v : \tau \to \tau$, and by definition of fix we have that $\mathsf{fix}\ \hat{e} \equiv \mathsf{fix}\ v$.

- T-MON
  $e \equiv \mathsf{mon}(\hat{c}, \hat{e})$: if $\langle \hat{e}, \sigma \rangle$ can step to an $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If it is a value v where $\Gamma; \Sigma \vdash v : \tau$, then we consider $\hat{c}$. Note that $\Gamma; \Sigma \vdash v : \tau$ and $\Gamma; \Sigma \vdash \hat{c} : \mathsf{con}(\tau)$.

– $\hat{c} \equiv \mathsf{flat}(c')$: By T-FLAT, $\Gamma; \Sigma \vdash c' : \tau \to$ **boolean**, so by T-CHECK, we can take the step $\delta \vdash \langle \mathsf{mon}(\hat{c}, \hat{e}), \sigma \rangle \longrightarrow \langle \mathsf{check}(c'\ v, v), \sigma \rangle$

– $\hat{c} \equiv c_1' \to c_2'$: Note $\Gamma; \Sigma \vdash \hat{c} : \mathsf{con}(\tau_1 \to \tau_2)$ and $\tau = \tau_1 \to \tau_2$. By T-FUNCON, $\Gamma; \Sigma \vdash c_1' : \mathsf{con}(\tau_1)$ and $\Gamma; \Sigma \vdash c_2' : \mathsf{con}(\tau_2)$. Therefore, by T-MON, we can take the step $\delta \vdash \langle \mathsf{mon}(c_1' \to c_2', v) \rangle \longrightarrow \langle \lambda x.\ \mathsf{mon}(c_2, v\ \mathsf{mon}(c_1, x)), \sigma \rangle$.

- T-FLAT
  This is impossible because $\mathsf{flat}(c)$ is not an expression.

- T-CHECK
  $e \equiv \mathsf{check}(\hat{e}, v)$: if $\langle \hat{e}, \sigma \rangle$ can step to an $\langle e', \sigma' \rangle$ by the inductive hypothesis, then by CONTEXT we are done. If it is a value v', then we have $\Gamma; \Sigma \vdash v' :$ **boolean**, so if $v' =$ **true**, then by MON-EXIT, $\delta \vdash \langle e, \sigma \rangle \longrightarrow \langle v, \sigma[\delta + 1 \mapsto \emptyset] \rangle$; if $v' =$ **false**, then we throw an error.

- T-FUNCON
  This is impossible because $c_1 \to c_2$ is not an expression.

□

Lemma 4. *(Type substitution). If $x : \tau' \vdash e : \tau$ and $\vdash v : \tau'$ then $\vdash e\{v/x\} : \tau$.*

2 Erasure

Lemma 5. *(Erasure substitution).* $\forall e, v, x,\ erase(e\{v/x\}) \equiv (erase(e))\{erase(v)/x\}$.

Proof. Let e, v, x be given. We proceed by induction on the structure of e.

Base case

- $n, \mathbf{true}, \mathbf{false}, \ell,\ !\ell$:
  By definition of $erase$, $e \equiv erase(e)$, and for any w, since x does not appear in e, $e \equiv e\{w/x\}$. Note that $erase(e\{v/x\}) \equiv erase(e) \equiv e$ and that $(erase(e))\{erase(v)/x\}) \equiv e\{erase(v)/x\} \equiv e$.

- x: We have $erase(e\{v/x\}) \equiv erase(v)$ and that $(erase(e))\{erase(v)/x\}) \equiv x\{erase(v)/x\} \equiv erase(v)$.

Inductive hypothesis Suppose that for sub-expressions e, we have that $erase(e\{v/x\}) \equiv (erase(e))\{erase(v)/x\}$.

Inductive step

- $\mathsf{mon}(c, e')$:

$$
\begin{aligned}
& erase(e\{v/x\}) \\
\equiv & erase((\mathsf{mon}(c, e'))\{v/x\}) \equiv erase(\mathsf{mon}(c\{v/x\}, e'\{v/x\})) \\
\equiv & erase(e'\{v/x\} \text{ by the definition of } erase \\
\equiv & (erase(e'))\{erase(v)/x\} \text{ by the inductive hypothesis} \\
\equiv & (erase(\mathsf{mon}(c, e')))\{erase(v)/x\} \text{ by the definition of } erase \\
\equiv & (erase(e))\{erase(v)/x\}
\end{aligned}
$$

- $\mathsf{check}(e', v')$:

$$
\begin{aligned}
& erase(e\{v/x\}) \\
\equiv & erase((\mathsf{check}(e', v'))\{v/x\}) \equiv erase(\mathsf{check}(e'\{v/x\}, v'\{v/x\})) \\
\equiv & erase(v'\{v/x\} \text{ by the definition of } erase \\
\equiv & (erase(v'))\{erase(v)/x\} \text{ by the inductive hypothesis} \\
\equiv & (erase(\mathsf{check}(e', v')))\{erase(v)/x\} \text{ by the definition of } erase \\
\equiv & (erase(e))\{erase(v)/x\}
\end{aligned}
$$

- other: $erase$ is defined homomorphically for the remaining cases, and this, combined with the inductive hypothesis, establishes the desired property. We demonstrate application; the other cases are similar.

$$\begin{aligned}
&erase(e\{v/x\})\\
\equiv&erase((e_1\ e_2)\{v/x\}) \equiv erase((e_1\{v/x\})(e_2\{v/x\}))\\
\equiv&(erase(e_1\{v/x\}))(erase(e_2\{v/x\})) \text{ by the homomorphism of } erase\\
\equiv&((erase(e_1))\{erase(v)/x\})((erase(e_2))\{erase(v)/x\}) \text{ by the inductive hypothesis}\\
\equiv&(erase(e_1\ e_2))\{erase(v)/x\} \text{ by the homomorphism of } erase\\
\equiv&(erase(e))\{erase(v)/x\}
\end{aligned}$$

□

Definition 1. (η-similarity). *Let $=_\eta$ be the smallest equivalence relation such that*

i $e_1 \equiv e_2 \Rightarrow e_1 =_\eta e_2$;

ii $\lambda x.\ (e\ x) =_\eta e$ *for all* $x \notin FV(e)$ and e not a function of the form $\lambda y.\ (e'\ y)$;

iii $e_1 =_\eta e_2 \Rightarrow E[e_1] =_\eta E[e_2]$

For notational convenience, we denote $\langle e, \sigma\rangle =_\eta \langle e', \sigma\rangle$ if $e =_\eta e'$.

Lemma 6. *(Single-step erasability). Suppose* $0 \vdash \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$. *Then, for any* $\hat{e} =_\eta e$, $0 \vdash \langle erase(\hat{e}), erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$, *where* $e'' =_\eta erase(e')$.

Proof. We induct on the height of the derivation of $\delta \vdash \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$, proving the following:

If $\delta \vdash \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle$, then $\forall \hat{e} =_\eta erase(e)\ \exists e'' =_\eta erase(e')$ s.t. $\delta \vdash \langle \hat{e}, erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$, where $erase(\sigma) = erase(\sigma')$ if $\delta \geq 1$.

Let $\hat{e} =_\eta e$.

Base case

- β-REDUCTION: $\delta \vdash \langle(\lambda x.\ e')v, \sigma\rangle \longrightarrow \langle e'\{v/x\}, \sigma\rangle$

$$\begin{aligned}
&\delta \vdash \langle erase(\hat{e}), erase(\sigma)\rangle\\
=_\eta&\langle erase((\lambda x.\ e')v), erase(\sigma)\rangle\\
\equiv&\langle(\lambda x.\ erase(e'))(erase(v)), erase(\sigma)\rangle && \text{by definition of } erase\\
\longrightarrow&\langle(erase(e'))\{erase(v)/x\}, erase(\sigma)\rangle\\
\equiv&\langle erase(e'\{v/x\}), erase(\sigma)\rangle && \text{by the Substitution Lemma}
\end{aligned}$$

- IF-THEN: $\delta \vdash \langle \textbf{if true then } e_1 \textbf{ else } e_2, \sigma\rangle \longrightarrow \langle e_1, \sigma\rangle$

$$\begin{aligned}
&\delta \vdash \langle erase(\hat{e}), erase(\sigma)\rangle\\
=_\eta&\langle erase(\textbf{if true then } e_1 \textbf{ else } e_2), erase(\sigma)\rangle\\
\equiv&\langle \textbf{if } erase(\textbf{true}) \textbf{ then } erase(e_1) \textbf{ else } erase(e_2), erase(\sigma)\rangle && \text{by definition of } erase\\
\equiv&\langle \textbf{if true then } erase(e_1) \textbf{ else } erase(e_2), erase(\sigma)\rangle\\
\longrightarrow&\langle erase(e_1), erase(\sigma)\rangle
\end{aligned}$$

- IF-ELSE: $\delta \vdash \langle \textbf{if false then } e_1 \textbf{ else } e_2, \sigma\rangle \longrightarrow \langle e_2, \sigma\rangle$

$$\begin{aligned}
&\delta \vdash \langle erase(\hat{e}), erase(\sigma)\rangle\\
=_\eta&\langle erase(\textbf{if false then } e_1 \textbf{ else } e_2), erase(\sigma)\rangle\\
\equiv&\langle \textbf{if } erase(\textbf{false}) \textbf{ then } erase(e_1) \textbf{ else } erase(e_2), erase(\sigma)\rangle && \text{by definition of } erase\\
\equiv&\langle \textbf{if false then } erase(e_1) \textbf{ else } erase(e_2), erase(\sigma)\rangle\\
\longrightarrow&\langle erase(e_2), erase(\sigma)\rangle
\end{aligned}$$

- SEQ We have that $\hat{e} =_\eta v; e_1$ and $e' \equiv e_1$. Note that $\sigma = \sigma'$. We want to show that:

  $$\langle erase(\hat{e}), erase(\sigma)\rangle) \longrightarrow^* \langle erase(e_1), erase(\sigma)\rangle$$

  By definition of $erase$ and application of SEQ:

  $$erase(\hat{e}) =_\eta erase(v); erase(e_1) \longrightarrow erase(e_1)$$

  We have trivially that $erase(\sigma) = erase(\sigma)$.
  Therefore, we have the desired result:

  $$\delta \vdash \langle erase(v; e_1), erase(\sigma)\rangle \longrightarrow \langle erase(e_1), erase(\sigma)\rangle$$

- DEREF We have that $\hat{e} =_\eta !\ell$, $e' \equiv v$, and $\sigma \equiv \sigma'$.

  $$\begin{aligned}
  &\langle erase(\hat{e}), erase(\sigma)\rangle \\
  =_\eta &\langle erase(\ !\ell), erase(\sigma)\rangle \\
  \equiv &\langle\ !\ell, erase(\sigma)\rangle && \text{by definition of erase} \\
  \longrightarrow &\langle (erase(\sigma))\,[(\delta, \ell)], erase(\sigma)\rangle && \text{by DEREF} \\
  \longrightarrow &\langle erase(v), erase(\sigma)\rangle && \text{by definition of erase on } \sigma
  \end{aligned}$$

- ALLOC We have that $\hat{e} =_\eta \mathsf{ref}\ v, e' \equiv \ell$. We need to show that for some σ' , $erase(\sigma)$ becomes $erase(\sigma[(\delta, \ell) \mapsto v])$.
  Recall that $\ell \equiv erase(\ell)$.

  $$\begin{aligned}
  &\langle erase(\hat{e}), erase(\sigma)\rangle \\
  =_\eta &\langle erase(\mathsf{ref}\ v), erase(\sigma)\rangle \\
  \equiv &\langle \mathsf{ref}\ erase(v), erase(\sigma)\rangle && \text{by definition of erase} \\
  \longrightarrow &\langle \ell, erase(\sigma)[(\delta, \ell) \mapsto erase(v)]\rangle && \text{by ALLOC} \\
  \equiv &\langle erase(\ell), erase(\sigma)[(\delta, \ell) \mapsto v]\rangle && \text{by definition of erase on } \ell \text{ and } \sigma \\
  \equiv &\langle erase(\ell), erase(\sigma)[(\delta, \ell) \mapsto v]\rangle
  \end{aligned}$$

- ASSIGN We have that $\hat{e} =_\eta \ell := v$, $e' \equiv v$. We also need to show that $erase(\sigma)$ becomes $erase(\sigma[(\delta, \ell) \mapsto v])$. Similar to the proof for ALLOC:

  $$\begin{aligned}
  &\langle erase(\hat{e}), erase(\sigma)\rangle \\
  =_\eta &\langle erase(\ell := v), erase(\sigma)\rangle \\
  \longrightarrow &\langle erase(\ell) := erase(v), erase(\sigma)\rangle \\
  \equiv &\ell := erase(v), erase(\sigma)\rangle && \text{(by definition of erase on } \ell) \\
  \longrightarrow &\langle erase(v), erase(\sigma)[(0, \ell) \mapsto erase(v)]\rangle && \text{(by ASSIGN)} \\
  \equiv &\langle erase(v), erase(\sigma[(0, \ell) \mapsto v])\rangle && \text{(by definition of erase on } \ell \text{ and } \sigma)
  \end{aligned}$$

- MON: $\delta \vdash \langle \mathsf{mon}(v_1 \to v_2, v), \sigma\rangle \longrightarrow \langle \lambda x.\ \mathsf{mon}(v_2, v\ \mathsf{mon}(v_1, x)), \sigma\rangle$ where x is a free variable
  By definition of $erase$,

  $$\langle erase(\hat{e}), erase(\sigma)\rangle =_\eta \langle erase(\mathsf{mon}(v_1 \to v_2, v)), erase(\sigma)\rangle \equiv \langle erase(v), erase(\sigma)\rangle$$

  Also,

  $$\begin{aligned}
  &\langle erase(\lambda x.\ \mathsf{mon}(v_2, v\ \mathsf{mon}(v_1, x))), erase(\sigma)\rangle \\
  \equiv &\langle \lambda x.\ erase(\mathsf{mon}(v_2, v\ \mathsf{mon}(v_1, x))), erase(\sigma)\rangle \\
  \equiv &\langle \lambda x.\ erase(v\ \mathsf{mon}(v_1, x)), erase(\sigma)\rangle \\
  \equiv &\langle \lambda x.\ erase(v)\ erase(\mathsf{mon}(v_1, x)), erase(\sigma)\rangle \\
  \equiv &\langle \lambda x.\ erase(v)\ erase(x), erase(\sigma)\rangle \\
  =_\eta &\langle erase(v), erase(\sigma)\rangle
  \end{aligned}$$

hence $\delta \vdash \langle erase(\mathsf{mon}(v_1 \to v_2, v)), erase(\sigma)\rangle \longrightarrow^0 \langle erase(\lambda x.\ \mathsf{mon}(v_2, v\ \mathsf{mon}(v_1, x))), erase(\sigma)\rangle$.

- MON-ENTER: $\delta \vdash \langle \mathsf{mon}(\mathsf{flat}(v'), v), \sigma\rangle \longrightarrow \langle \mathsf{check}(v'\ v, v), \sigma\rangle$
  By definition of $erase$,

  $$\langle erase(\hat{e}), erase(\sigma)\rangle =_\eta \langle erase(\mathsf{mon}(\mathsf{flat}(v'), v)), erase(\sigma)\rangle \equiv \langle erase(v), erase(\sigma)\rangle$$

  and

  $$\langle erase(\mathsf{check}(v'\ v), v), erase(\sigma)\rangle \equiv \langle erase(v), erase(\sigma)\rangle$$

  hence $\delta \vdash \langle erase(\mathsf{mon}(\mathsf{flat}(v'), v)), erase(\sigma)\rangle \longrightarrow^0 \langle erase(\mathsf{check}(v'\ v), v), erase(\sigma)\rangle$.

- MON-EXIT: $\delta \vdash \langle \mathsf{check}(\mathbf{true}, v), \sigma\rangle \longrightarrow \langle v, \sigma[1 \mapsto \emptyset]\rangle$
  By definition of $erase$,

  $$erase(\hat{e}) =_\eta erase(\mathsf{check}(\mathbf{true}, v)) \equiv erase(v)$$

  Also,

  $$erase(\sigma(\delta, x)) = \begin{cases} \emptyset & \delta \geq 1 \\ \sigma(\delta, erase(x)) & \delta = 0 \end{cases}$$

  $$erase(\sigma[1 \mapsto \emptyset](\delta, x)) = \begin{cases} \emptyset & \delta \geq 1 \\ erase(\sigma(\delta, x)) = \sigma(\delta, erase(x)) & \delta = 0 \end{cases}$$

  hence the erasure of the two stores are also equivalent, as desired.

Inductive hypothesis Suppose that for derivation trees of height h, $\delta \vdash \langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle \Rightarrow \forall \hat{e} =_\eta e\ \exists e'' =_\eta erase(e')$ s.t. $\delta \vdash \langle erase(\hat{e}), erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$, where $erase(\sigma') = erase(\sigma)$ if $\delta \geq 1$.

Case $h + 1$

- CONTEXT

  Consider the case $v\ E$. From CONTEXT, we know that $E[e] \longrightarrow E[e']$. The premise is $e \longrightarrow e'$, and by the inductive hypothesis, we know that $erase(e) \longrightarrow^* erase(e')$.

  By the definition of erase,

  $$erase(E[e]) \equiv erase(v\ e)$$

  Now, we have:

  $$erase(v\ e) \longrightarrow erase(v)\ erase(e)$$

  Using the inductive hypothesis and repeated applications of the CONTEXT rule with the premise $erase(e) \longrightarrow erase(e'')$ for some e'', we yield:

  $$erase(v)\ erase(e) \longrightarrow^* erase(v)\ erase(e') \equiv erase(E[e'])$$

  This is the desired result.

  The other evaluation contexts follow very similarly by straightforward application of the definition of erase and repeated application of the CONTEXT rule.

  By definition of $erase$,

  $$\langle \hat{e}, erase(\sigma)\rangle =_\eta \langle erase(E[e']), erase(\sigma)\rangle$$

- MON-CHECK

  By definition of $erase$,

  $$\langle erase(\hat{e}), erase(\sigma)\rangle =_\eta \langle erase(\mathsf{check}(e, v)), erase(\sigma)\rangle \equiv \langle erase(v), erase(\sigma)\rangle$$

  and

  $$\langle erase(\mathsf{check}(e', v)), erase(\sigma')\rangle \equiv \langle erase(v), erase(\sigma')\rangle$$

  If $\delta \geq 1$, then $\delta + 1 \geq 1$ certainly, and by the inductive hypothesis, $erase(\sigma') = erase(\sigma)$.

□

Theorem 1. *(Erasability).* $0 \vdash \langle e, \sigma\rangle \longrightarrow^* \langle e', \sigma'\rangle \Rightarrow \forall \hat{e} =_\eta erase(e)\ \exists e'' =_\eta erase(e')$ *s.t.* $0 \vdash \langle \hat{e}, erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$

Proof. We prove that

$$0 \vdash \langle e, \sigma\rangle \longrightarrow^n \langle e', \sigma'\rangle \Rightarrow \forall \hat{e} =_\eta erase(e)\ \exists e'' =_\eta erase(e') \quad 0 \vdash \langle \hat{e}, erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$$

and proceed with induction on n.

Base case Suppose that $0 \vdash \langle e, \sigma\rangle \longrightarrow^0 \langle v, \sigma'\rangle$, i.e. e is a value and $e \equiv v$. It follows that:

$$\langle \hat{e}, erase(\sigma)\rangle =_\eta \langle erase(e), erase(\sigma)\rangle \equiv \langle erase(v), erase(\sigma)\rangle$$

As $\sigma = \sigma'$, this yields the desired result:

$$0 \vdash \langle \hat{e}, erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma)\rangle$$

where $e'' =_\eta erase(v)$.

Inductive hypothesis Suppose that $0 \vdash \langle e, \sigma\rangle \longrightarrow^n \langle e', \sigma'\rangle \Rightarrow \forall \hat{e} =_\eta erase(e)\ \exists e'' =_\eta erase(e')$ s.t. $0 \vdash \langle \hat{e}, erase(\sigma)\rangle \longrightarrow^* \langle e'', erase(\sigma')\rangle$.

Case $n+1$ Suppose we have $0 \vdash \langle e, \sigma\rangle \longrightarrow^n \langle e'', \sigma''\rangle \longrightarrow \langle e', \sigma'\rangle$. By the inductive hypothesis, we have that $0 \vdash \langle erase(e), erase(\sigma)\rangle \longrightarrow^* \langle \hat{e}'', erase(\sigma'')\rangle$ where $\hat{e}'' =_\eta erase(e'')$. By the lemma, $0 \vdash \langle \hat{e}'', erase(\sigma'')\rangle \longrightarrow^* \langle \hat{e}', erase(\sigma')\rangle$, where $\hat{e}' =_\eta erase(e')$, finishing the proof.

□