Formal Properties of Linear Memory Types
Dartmouth Technical Report TR2003-468

Heng Huang, Lea Wittie, and Chris Hawblitzel

Aug 8, 2003

Abstract

Efficient low-level systems need more control over memory than safe high-level languages usually provide. As a result, run-time systems are typically written in unsafe languages such as C. This report describes an abstract machine designed to give type-safe code more control over memory. It includes complete definitions and proofs of preservation, progress, strong normalization, erasure, and translation correctness.

Introduction

This report presents the complete syntax and rules for the abstract machine, called λ^{low}, described in [1]. It then presents a proof of soundness (type safety), which says that well-typed programs will never get stuck. The proof consists of a proof of preservation and a proof of progress, in the syntactic style of [5] (also see [4] for an general introduction to syntactic approaches to semantics and types). It also presents a proof of strong normalization for the "proof" portion of the abstract machine (i.e. for the expressions that type-check in a "limited" environment), and a proof that the types and the "proof" portion of the abstract machine can be erased without upsetting the run-time behavior. Finally, it proves the well-typedness of a translation from λ^C, a CPS- and closure-converted polymorphic lambda calculus [3], to λ^{low}. This report supersedes an earlier report[2].

Formally stated, the theorems proved in this report are:

- **Type preservation:** If $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau$, $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$ and $(M, e) \rightarrow (M', e')$, then $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$ and $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$, where $(\Phi'_e \dot{\supseteq} \Phi_e)$.
- **Type progress:** If (M, e) is closed and well-typed ($C_{Me} \vdash (M, e : \tau)$ for some τ and $C_{Me} = \Psi_{Me}; \Phi_{Me}; \varnothing; \varnothing; B; \text{limit}$), then either e is a value or else there is some (M', e') so that $(M, e) \rightarrow (M', e')$.
- **Strong normalization:** If $\Psi_{Me}; \Phi_{Me}; \varnothing; \varnothing; \text{true}; i \vdash (M, e : \tau)$, then e must step to a value in a finite sequence of zero or more steps, without changing memory: $(M, e) \rightarrow (M, e_1) \rightarrow \cdots \rightarrow (M, e_n) \rightarrow (M, v)$.
- **Type erasure:**
  - $\dfrac{C \vdash (M, e:\tau) \quad (M, e) \mapsto (M', e')}{\text{erase}((M, e)) \overset{?}{\mapsto} \text{erase}((M', e'))}$
  - $\dfrac{C \vdash (M, e:\tau) \text{ where } C \text{ has an empty } \Delta \text{ and } \Gamma \quad \text{erase}((M, e)) \mapsto (L', d')}{(M, e) \overset{+}{\mapsto} (M', e'), \text{erase}((M', e')) = (L', d')}$
  - $\dfrac{C \vdash (M, e:\tau) \text{ where } C \text{ has an empty } \Delta \text{ and } \Gamma \quad \text{erase}(e) \text{ is a value}}{(M, e) \overset{*}{\mapsto} (M, v), \text{erase}((M, e)) = \text{erase}((M, v))}$
- **Translation type correctness:**
  - If $\Delta; \Gamma \vdash_C e$, then $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(e)$
  - If $\Delta \vdash_C \Gamma$, then $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\Gamma)$
  - If $\Delta; \Gamma \vdash_{C_2} e$, then $\Delta; \Gamma \vdash_F \mathcal{F}(e)$

– If $\Delta \vdash_{C_2} \Gamma$, then $\Delta \vdash_F \Gamma$

– If $\Delta \vdash_F \Gamma$ and $\Delta; \Gamma \vdash_F e$, then $\emptyset; \emptyset; \mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \mathrm{int}; \mathcal{A}(\Gamma)\, R\, E,\ \{g : \tau_g(R, E)\}; true; \infty \vdash_L \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$

1 Abstract syntax

This section defines the abstract syntax of λ^{low}. The letter x is used to indicate a value variable, while α is used to indicate a type variable. As usual, we consider expressions and types that differ only in bound variable names to be equivalent.

linearity

$$\phi = \cdot \mid \wedge$$

time limits

$$\mathrm{limit} = I \mid \infty$$

kinds

$$K = J \mid \mathrm{int} \mid \mathrm{bool}$$

$$J = \overset{\phi}{i} \mid K \to J$$

arithmetic

$$i = \cdots \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid \cdots$$

$$b = \mathrm{true} \mid \mathrm{false}$$

$$I = \alpha \mid i \mid I_1\, \mathrm{iop}\, I_2$$

$$B = \alpha \mid b \mid \neg B \mid B_1\, \mathrm{bop}\, B_2 \mid I_1\, \mathrm{cmp}\, I_2$$

$$\mathrm{iop}\ = + \mid - \mid *$$

$$\mathrm{bop}\ = \wedge \mid \vee$$

$$\mathrm{cmp}\ =< \ \mid\ > \ \mid\ \leq \ \mid\ \geq \ \mid\ = \ \mid\ \neq$$

$$\mathrm{op}\ =\ \mathrm{iop}\ \mid\ \mathrm{bop}\ \mid\ \mathrm{cmp}$$

types

$$\tau = \tau_1 \overset{\phi,limit}{\longrightarrow} \tau_2 \mid \phi\langle\overrightarrow{\tau}\rangle \mid \alpha \mid I \mid B \mid \lambda\alpha : K.\tau \mid \forall\alpha : K; B.\tau$$

$$\mid \exists\alpha : K; B.\tau \mid \tau_1\, \tau_2 \mid \mu\alpha : K.\tau \mid \mathrm{Int}(I) \mid \mathrm{Bool}(B) \mid \mathrm{Union}(B, \tau_1, \tau_2)$$

$$\mid \mathrm{Has}(I, \tau) \mid \mathrm{Gen}(\tau, I) \mid \mathrm{Eq}(\tau_1, \tau_2) \mid \mathrm{F}^K \mid \mathrm{InDomain}(I, \tau)$$

expressions

$$e = i \mid b \mid x \mid e_1\, e_2 \mid e\tau \mid \phi\langle \overrightarrow{e} \rangle \mid \lambda x : \tau \xrightarrow{\phi,\mathrm{limit}} e \mid e_1 \,\mathrm{op}\, e_2 \mid \neg e$$

$$\mid \Lambda\alpha : K; B.v \mid \mathrm{let}\, \langle \overrightarrow{x} \rangle = e_1 \,\mathrm{in}\, e_2 \mid \mathrm{if}\, e_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3 \mid \mathrm{if}\, B \,\mathrm{then}\, e_1 \,\mathrm{else}\, e_2$$

$$\mid \mathrm{union}(b, \tau_1, \tau_2, e) \mid \mathrm{pack}[\tau_1, e] \,\mathrm{as}\, \exists\alpha : K; B.\tau_2 \mid \mathrm{unpack}\, \alpha, x = e_1 \,\mathrm{in}\, e_2$$

$$\mid \mathrm{case}(b, e) \mid \mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](e) \mid \mathrm{unroll}(e) \mid \mathrm{fix}\, x : \tau.\mathrm{v} \mid \mathrm{load}(e_{\mathrm{ptr}}, e_{\mathrm{Has}})$$

$$\mid \mathrm{store}(e_{\mathrm{ptr}}\ , e_{\mathrm{Has}}, \mathrm{e_v}) \mid \mathrm{coerce}(e) \mid \mathrm{distinguish}(I_1, I_2, e_1, e_2) \mid \mathrm{make\_eq}(\tau)$$

$$\mid \mathrm{apply\_eq}(\tau, e_1, e_2) \mid \mathrm{new\_fun}(K) \mid \mathrm{discard\_fun}(e)$$

$$\mid \mathrm{define\_fun}(e, \tau) \mid \mathrm{in\_domain}(I_1, I_2, e_1, e_2) \mid \mathrm{fact}$$

values

$$v = i \mid b \mid \Lambda\alpha : K; B.v \mid \mathrm{pack}[\tau_1, v] \,\mathrm{as}\, \exists\alpha : K; B.\tau_2$$

$$\mid \mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) \mid \lambda x : \tau \xrightarrow{\phi,\mathrm{limit}} e \mid \phi\langle \overrightarrow{v} \rangle \mid \mathrm{union}(b, \tau_1, \tau_2, v) \mid \mathrm{fact}$$

untyped expressions

$$d = i \mid b \mid x \mid d_1\, d_2 \mid \langle \overrightarrow{d} \rangle \mid \lambda x \longrightarrow d$$

$$\mid d_1 \,\mathrm{op}\, d_2 \mid \neg d \mid \mathrm{let}\, \langle \overrightarrow{x} \rangle = d_1 \,\mathrm{in}\, d_2$$

$$\mid \mathrm{if}\, d_1 \,\mathrm{then}\, d_2 \,\mathrm{else}\, d_3 \mid \mathrm{fix}\, x.u \mid \mathrm{load}(d_{\mathrm{ptr}}, \langle\rangle) \mid \mathrm{store}(d_{\mathrm{ptr}}\ , \langle\rangle, d_{\mathrm{v}})$$

untyped values

$$u = i \mid b \mid \lambda x \longrightarrow d \mid \langle \overrightarrow{u} \rangle$$

expressions for substitution

$$s = v \mid \mathrm{fix}\, x : \tau.v$$

environments

$$F = F_1^{K_1} \mid F_2^{K_2} \mid F_3^{K_3} \mid \cdots$$

$$M = \{1 \mapsto v_1, \cdots, n \mapsto v_n\}$$

$$\Psi = \{1 \mapsto \tau_1, \cdots, n \mapsto \tau_n\}$$

$$\Phi = \{F_1^{K_1} \overset{\phi_1}{\mapsto} \delta_1, \cdots, F_n^{K_n} \overset{\phi_n}{\mapsto} \delta_n\}$$

$$\delta = \{0 \mapsto \tau_0, \cdots, n \mapsto \tau_n\}$$

$$\Delta = \{\alpha_1 \mapsto K_1, \cdots, \alpha_n \mapsto K_n\}$$

$$\Gamma = \{x_1 \mapsto \tau_1 \cdots, x_n \mapsto \tau_n\}$$

$$C = \Psi; \Phi; \Delta; \Gamma; B; \mathrm{limit}$$

untyped environments

$$L = \{1 \mapsto u_1, \cdots, n \mapsto u_n\}$$

judgments

$$\Phi; \Delta \vdash \tau : K$$

$$B \vdash B_1 \doteq B_2$$

$$B \vdash I_1 \doteq I_2$$

$$\Phi; B \vdash \tau_1 \equiv \tau_2$$

$$C \vdash e : \tau$$

$$\Psi_{spare}; \Phi_{spare}; C \vdash (M, e : \tau)$$

$$(M, e) \to (M', e')$$

abbreviations

$$\forall \alpha : K.\tau \triangleq \forall \alpha : K; \mathrm{true}.\tau$$

$$\exists \alpha : K.\tau \triangleq \exists \alpha : K; \mathrm{true}.\tau$$

$$\mathrm{Know}(B) \triangleq \exists \alpha : \mathrm{bool}; B. \cdot \langle\rangle$$

$$\mathrm{know}(B) \triangleq \mathrm{pack}[\mathrm{true}, \cdot\langle\rangle] \,\mathrm{as}\, \mathrm{Know}(B)$$

$$\mathrm{let}\, x = e_1 \,\mathrm{in}\, e_2 \triangleq \mathrm{let}\, \langle x \rangle =^{\wedge} \langle e_1 \rangle \,\mathrm{in}\, e_2$$

$$\tau_1 \xrightarrow{\phi} \tau_2 \triangleq \tau_1 \xrightarrow{\phi,\infty} \tau_2$$

$$B_1 \vdash B_2 \triangleq B_1 \vdash B_2 \doteq \mathrm{true}$$

1.1 Notes on environments

We treat the environments Ψ, Φ, δ, Δ, Γ as sets, so the order of elements does not matter: $\{x_1 \mapsto \tau_1, x_2 \mapsto \tau_2\} = \{x_2 \mapsto \tau_2, x_1 \mapsto \tau_1\}$.

The environments Ψ, Φ, δ, Δ, Γ must be well-formed functions (and the definitions in this report apply only to well-formed functions):
$(i \mapsto \tau_1 \in \Psi) \wedge (i \mapsto \tau_2 \in \Psi) \Rightarrow \tau_1 = \tau_2$
$(F^K \overset{\phi_1}{\mapsto} \delta_1 \in \Phi) \wedge (F^K \overset{\phi_2}{\mapsto} \delta_2 \in \Phi) \Rightarrow (\delta_1 = \delta_2) \wedge (\phi_1 = \phi_2)$
$(i \mapsto \tau_1 \in \delta) \wedge (i \mapsto \tau_2 \in \delta) \Rightarrow \tau_1 = \tau_2$
$(\alpha \mapsto K_1 \in \Delta) \wedge (\alpha \mapsto K_2 \in \Delta) \Rightarrow K_1 = K_2$
$(x \mapsto \tau_1 \in \Gamma) \wedge (x \mapsto \tau_2 \in \Gamma) \Rightarrow \tau_1 = \tau_2$
For Ψ, δ, Δ, Γ we use the usual function application notation: $\Gamma(x) = \tau \Leftrightarrow x \mapsto \tau \in \Gamma$.

Each $F_i^{K_i}$ in Φ must suffice to type-check both $\mathrm{Gen}(F_i^{K_i}, I)$ expressions, which are linear, and other expression containing the type operator $F_i^{K_i}$, which may be nonlinear. The $\mathrm{Gen}(F_i^{K_i}, I)$ expression is only valid in a context where $\phi_i =\wedge$.

Environment splitting

These definitions split environments into two parts, where linear elements must go into exactly one of the parts:
$\Psi = \Psi_1, \Psi_2 \Leftrightarrow (\Psi = \Psi_1 \cup \Psi_2) \wedge (\mathit{domain}(\Psi_1) \not\cap \mathit{domain}(\Psi_2))$

$$
\begin{aligned}
\Phi = \Phi_1, \Phi_2 \Leftrightarrow \forall F^K.((F^K \overset{\wedge}{\mapsto} \delta \in \Phi) \quad &\Rightarrow \quad (F^K \overset{\wedge}{\mapsto} \delta \in \Phi_1) xor (F^K \overset{\wedge}{\mapsto} \delta \in \Phi_2)) \\
\wedge((F^K \overset{\wedge}{\mapsto} \delta \in \Phi) \quad &\Leftarrow \quad (F^K \overset{\wedge}{\mapsto} \delta \in \Phi_1) \vee (F^K \overset{\wedge}{\mapsto} \delta \in \Phi_2)) \\
\wedge((F^K \overset{\phi}{\mapsto} \delta \in \Phi) \quad &\Leftrightarrow \quad (F^K \overset{\phi_1}{\mapsto} \delta \in \Phi_1)) \\
\wedge((F^K \overset{\phi}{\mapsto} \delta \in \Phi) \quad &\Leftrightarrow \quad (F^K \overset{\phi_2}{\mapsto} \delta \in \Phi_2))
\end{aligned}
$$

$$
\begin{aligned}
\Phi; \Delta \vdash \Gamma = \Gamma_1, \Gamma_2 \Leftrightarrow \forall x, \tau.(\Phi; \Delta \vdash \tau \overset{\cdot}{:} i \quad &\Rightarrow \quad ((\Gamma(x) = \tau) \Leftrightarrow (\Gamma_1(x) = \tau)) \wedge ((\Gamma(x) = \tau) \Leftrightarrow (\Gamma_2(x) = \tau))) \\
\wedge(\Phi; \Delta \vdash \tau \overset{\wedge}{:} i \quad &\Rightarrow \quad ((\Gamma(x) = \tau) \Rightarrow (\Gamma_1(x) = \tau) xor (\Gamma_2(x) = \tau))) \\
\wedge(\Phi; \Delta \vdash \tau \overset{\wedge}{:} i \quad &\Rightarrow \quad ((\Gamma(x) = \tau) \Leftarrow (\Gamma_1(x) = \tau) \vee (\Gamma_2(x) = \tau)))
\end{aligned}
$$

$\Psi; \Phi; \Delta; \Gamma; B; \mathrm{limit} = (\Psi_1; \Phi_1; \Delta; \Gamma_1; B; \mathrm{limit}), (\Psi_2; \Phi_2; \Delta; \Gamma_2; B; \mathrm{limit}) \Leftrightarrow (\Psi = \Psi_1, \Psi_2) \wedge (\Phi = \Phi_1, \Phi_2) \wedge (\Phi; \Delta \vdash \Gamma = \Gamma_1, \Gamma_2)$

Environment extension

These add new elements to environments:
$\Psi, i \mapsto \tau \triangleq \Psi \cup \{i \mapsto \tau\}$, where $i \notin \mathit{domain}(\Psi)$
$\Phi, F^K \overset{\phi}{\mapsto} \delta \triangleq \Phi \cup \{F^K \overset{\phi}{\mapsto} \delta\}$, where $F^K \notin \mathit{domain}(\Phi)$
$\Delta, \alpha \mapsto K \triangleq \Delta \cup \{\alpha \mapsto K\}$, where $\alpha \notin \mathit{domain}(\Delta)$
$\Gamma, x \mapsto \tau \triangleq \Gamma \cup \{x \mapsto \tau\}$, where $x \notin \mathit{domain}(\Gamma)$
$B_1, B_2 \triangleq B_1 \wedge B_2$
$(\Psi; \Phi; \Delta; \Gamma; B; \mathit{limit}), i \mapsto \tau \triangleq (\Psi, i \mapsto \tau; \Phi; \Delta; \Gamma; B; \mathit{limit})$
$(\Psi; \Phi; \Delta; \Gamma; B; \mathit{limit}), F^K \overset{\phi}{\mapsto} \delta \triangleq (\Psi; \Phi, F^K \overset{\phi}{\mapsto} \delta; \Delta; \Gamma; B; \mathit{limit})$
$(\Psi; \Phi; \Delta; \Gamma; B; \mathit{limit}), \alpha \mapsto K_\alpha \triangleq (\Psi; \Phi; \Delta, \alpha \mapsto K_\alpha; \Gamma; B; \mathit{limit})$,
where α does not appear anywhere in $(\Psi; \Phi; \Delta; \Gamma; B; \mathit{limit})$.
$(\Psi; \Phi; \Delta; \Gamma; B; \mathit{limit}), x \mapsto \tau \triangleq (\Psi; \Phi; \Delta; \Gamma, x \mapsto \tau; B; \mathit{limit})$,
where x does not appear anywhere in $(\Psi; \Phi; \Delta; \Gamma; B; \mathit{limit})$ and $\Phi; \Delta \vdash \tau \overset{\phi}{:} i$.
$(\Psi; \Phi; \Delta; \Gamma; B_1; \mathit{limit}), B_2 \triangleq (\Psi; \Phi; \Delta; \Gamma; B_1, B_2; \mathit{limit})$

Nonlinear environments

The $\cdot$ operator removes any linearity from an environment. We use it to prohibit linearity in some of the type checking rules.

$$\dot{\Phi} \triangleq \{F^K \overset{\cdot}{\mapsto} \delta | F^K \overset{\phi}{\mapsto} \delta \in \Phi\}$$

$$\dot{\Gamma}\ (\Phi, \Delta) \triangleq \{x \mapsto \tau | (x \mapsto \tau \in \Gamma) \wedge (\Phi; \Delta \vdash \tau : \dot{i})\}$$

If $C = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}$, then $\dot{C} \triangleq \emptyset; \dot{\Phi}; \Delta; \dot{\Gamma}\ (\Phi, \Delta); B; limit$.

Environment subsets

As the abstract machine steps from one state to the next, the Φ environment must grow to accomodate new type sequence allocations. Therefore, we define a subset operator $\dot{\subseteq}$ for Φ to indicate that the Φ_2 after a step is an extension of the Φ_1 before the step. To accomodate the discard_fun(e) expression, this operator must be able to demote a linear mapping $F^K \overset{\wedge}{\mapsto} \delta$ to a nonlinear mapping $F^K \overset{\cdot}{\mapsto} \delta$:

$$\Phi_1 \dot{\subseteq} \Phi_2 \Leftrightarrow \forall F^K.(F^K \overset{\phi_1}{\mapsto} \delta_1 \in \Phi_1) \Rightarrow \exists \delta_2.((F^K \overset{\phi_2}{\mapsto} \delta_2 \in \Phi_2) \wedge (\delta_1 \subseteq \delta_2) \wedge ((\phi_2 = \wedge) \Rightarrow (\phi_1 = \wedge)))$$

$$(\Psi; \Phi_1; \Delta; \Gamma; B; \text{limit}) \dot{\subseteq} (\Psi; \Phi_2; \Delta; \Gamma; B; \text{limit}) \Leftrightarrow (\Phi_1 \dot{\subseteq} \Phi_2)$$

To prove a substitution lemma, we need a notion of environment weakening. Weakening, however, cannot add new linear elements nor change the linearity of an existing mapping, so we need a different subset operator, $\overset{\wedge}{\subseteq}$:

$$\begin{aligned}\Phi_1 \overset{\wedge}{\subseteq} \Phi_2 \quad \Leftrightarrow \quad & (\forall F^K.(F^K \overset{\phi}{\mapsto} \delta_1 \in \Phi_1) \Rightarrow \exists \delta_2.((F^K \overset{\phi}{\mapsto} \delta_2 \in \Phi_2) \wedge (\delta_1 \subseteq \delta_2))) \\ & \wedge (\forall F^K.(F^K \overset{\wedge}{\mapsto} \delta_2 \in \Phi_2) \Rightarrow \exists \delta_1.((F^K \overset{\wedge}{\mapsto} \delta_1 \in \Phi_1) \wedge (\delta_1 \subseteq \delta_2)))\end{aligned}$$

$$(\Psi; \Phi_1; \Delta_1; \Gamma_1; B; \text{limit}) \subseteq (\Psi; \Phi_2; \Delta_2; \Gamma_2; B; \text{limit}) \Leftrightarrow (\Phi_1 \overset{\wedge}{\subseteq} \Phi_2) \wedge (\Delta_1 \subseteq \Delta_2) \wedge (\Phi; \Delta \vdash \Gamma_1 \subseteq \Gamma_2)$$

where we define subset for Γ as:

$$\Phi; \Delta \vdash \Gamma_1 \subseteq \Gamma_2 \Leftrightarrow \Gamma_1, x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n = \Gamma_2$$

where $\Phi; \Delta \vdash \tau_1 : \dot{i}_1, \ldots, \Phi; \Delta \vdash \tau_n : \dot{i}_n$.

1.2 Arithmetic constraints

The notation $B \vdash I_1 \doteq I_2$ means that for all possible substitutions of integer constants for integer variables, B implies that I_1 is equal to I_2. Similarly, $B \vdash B_1 \doteq B_2$ means that for all possible substitutions of integer constants for integer variables and boolean constants for boolean variables, B implies that B_1 is equal to B_2.

2 Type equivalence

Type equivalence $\Phi; B \vdash \tau_1 \equiv \tau_2$ is only well-defined with respect to an environment $\Phi; B$. When this environment is clear, however, we will often just write $\tau_1 \equiv \tau_2$ by itself, for convenience.

$$\Phi; B \vdash \tau \equiv \tau$$

$$\frac{\Phi; B \vdash \tau_1 \equiv \tau_2}{\Phi; B \vdash \tau_2 \equiv \tau_1}$$

$$\frac{\Phi; B \vdash \tau_1 \equiv \tau_2 \quad \Phi; B \vdash \tau_2 \equiv \tau_3}{\Phi; B \vdash \tau_1 \equiv \tau_3}$$

$$\Phi; B \vdash (\lambda\alpha : K.\tau_b)\tau_a \equiv [\alpha \mapsto \tau_a]\tau_b$$

$$\frac{B \vdash I \doteq i}{\Phi, \mathrm{F}^K \dot{\mapsto} \delta; B \vdash \mathrm{F}^K I \equiv \delta(i)}$$

$$\frac{B \vdash I_1 \doteq I_2}{\Phi; B \vdash I_1 \equiv I_2}$$

$$\frac{B \vdash B_1 \doteq B_2}{\Phi; B \vdash B_1 \equiv B_2}$$

$$\Phi; B \vdash \mathrm{Eq}(\tau_1, \tau_2) \equiv \mathrm{Eq}(\tau_2, \tau_1)$$

$$\frac{\forall i.(\Phi; B \vdash \tau_i \equiv \tau_i')}{\Phi; B \vdash T[\tau_1, \ldots, \tau_n] \equiv T[\tau_1', \ldots, \tau_n']}$$

Rather than writing each of the congruence rules separately ($\frac{\tau \equiv \tau'}{\lambda\alpha:K.\tau \equiv \lambda\alpha:K.\tau'}$, etc.), we use T to indicate a type with one or more (shallowly dug) holes in it, and $T[\tau_1, \ldots, \tau_n]$ to indicate the type with the holes replaced by $\tau_1, \ldots, \tau_n$, so that a single type equivalence rule covers all the cases. This is only for notational convenience.

$$T[\tau] = \lambda\alpha : K.\tau \,|\, \mu\alpha : K.\tau \,|\, \phi\langle\tau\rangle \,|\, \mathrm{Int}(\tau) \,|\, \mathrm{Bool}(\tau)$$

$$T[\tau_1, \tau_2] = \tau_1\,\tau_2 \,|\, \forall\alpha : K; \tau_1.\tau_2 \,|\, \exists\alpha : K; \tau_1.\tau_2 \,|\, \tau_1 \longrightarrow \tau_2 \,|\, \phi\langle\tau_1, \tau_2\rangle$$

$$|\, \mathrm{Has}(\tau_1, \tau_2) \,|\, \mathrm{Gen}(\tau_1, \tau_2) \,|\, \mathrm{Eq}(\tau_1, \tau_2) \,|\, \mathrm{InDomain}(\tau_1, \tau_2)$$

$$T[\tau_1, \tau_2, \tau_3] = \mathrm{Union}(\tau_1, \tau_2, \tau_3) \,|\, \phi\langle\tau_1, \tau_2, \tau_3\rangle$$

$$T[\tau_1, \tau_2, \tau_3, \ldots, \tau_n] = \phi\langle\tau_1, \tau_2, \tau_3, \ldots, \tau_n\rangle$$

3 Evaluation rules

3.1 Definitions

- $(M_1, e_1) \overset{?}{\mapsto} (M_2, e_2)$ means (M_1, e_1) progresses in zero or one steps to (M_2, e_2).
- $(M_1, e_1) \overset{*}{\mapsto} (M_2, e_2)$ means (M_1, e_1) progresses in zero or more steps to (M_2, e_2).
- $(M_1, e_1) \overset{+}{\mapsto} (M_2, e_2)$ means (M_1, e_1) progresses in one or more steps to (M_2, e_2).
- $(M_1, e_1) \overset{?,(evaluation-rule)}{\mapsto} (M_2, e_2)$ means (M_1, e_1) progresses to (M_2, e_2) by applying the given evaluation rule zero or one times.
- $(M_1, e_1) \overset{*,(evaluation-rule)}{\mapsto} (M_2, e_2)$ and $(M_1, e_1) \overset{+,(evaluation-rule)}{\mapsto} (M_2, e_2)$ are defined in the same way as the above definition.
- $\mathrm{simplify}(i_1 + i_2) = i_3$, where i_3 is the sum of i_1 and i_2
- $\mathrm{simplify}(i_1 - i_2) = i_3$, where i_3 is i_1 minus i_2

- simplify$(i_1 * i_2) = i_3$, where i_3 is the product of i_1 and i_2
- simplify$(i_1 \,\mathrm{cmp}\, i_2) = b$, where b is true iff cmp (i_1, i_2) is true
- simplify$(b_1 \wedge b_2) = b_3$, where b_3 is true iff b_1 and b_2 are true
- simplify$(b_1 \vee b_2) = b_3$, where b_3 is true iff b_1 or b_2 is true
- simplify$(\neg b_1) = b_2$, where b_2 is true iff b_1 is false

3.2 Rules

$$(E-LOAD)(M, \mathrm{load}(i, fact)) \rightarrow (M,^\wedge \langle M(i), fact\rangle)$$

$$(E-STORE)(M, \mathrm{store}(i, fact, v)) \rightarrow ([i \mapsto v]M, fact)$$

$$(E-ABSAPP1)(\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2 \rightarrow \mathrm{coerce}([x \mapsto v_2]e_1)$$

$$(E-ABSAPP2)(\lambda x : \tau \xrightarrow{\phi, \infty} e_1)v_2 \rightarrow [x \mapsto v_2]e_1$$

$$(E-COERCE)\mathrm{coerce}(v) \rightarrow v$$

$$(E-TAPPTABS)(\Lambda\alpha : K; B.v)\tau \rightarrow [\alpha \mapsto \tau]v$$

$$(E-LET)\mathrm{let}\,\langle x_1, \cdots, x_n\rangle = \phi\langle v_1, \cdots, v_n\rangle\,\mathrm{in}\,e \rightarrow [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e$$

$$(E-SIMPLIFY1)v_1\,\mathrm{op}\,v_2 \rightarrow \mathrm{simplify}(v_1\,\mathrm{op}\,v_2)$$

$$(E-SIMPLIFY2)\neg v \rightarrow \mathrm{simplify}(\neg v)$$

$$(E-IF1)\mathrm{if}\,true\,\mathrm{then}\,e_1\,\mathrm{else}\,e_2 \rightarrow e_1$$

$$(E-IF2)\mathrm{if}\,false\,\mathrm{then}\,e_1\,\mathrm{else}\,e_2 \rightarrow e_2$$

$$(E-IFB1)\frac{\vdash B \doteq true}{\mathrm{if}\,B\,\mathrm{then}\,e_1\,\mathrm{else}\,e_2 \rightarrow e_1}$$

$$(E-IFB2)\frac{\vdash B \doteq false}{\mathrm{if}\,B\,\mathrm{then}\,e_1\,\mathrm{else}\,e_2 \rightarrow e_2}$$

$$(E-UNPACK)\mathrm{unpack}\,\alpha, x = (\mathrm{pack}[\tau_1, v_1]\,\mathrm{as}\,\tau_2)\mathrm{in}\,e_2 \rightarrow [\alpha \mapsto \tau_1, x \mapsto v_1]e_2$$

$$(E-UNROLL)\mathrm{unroll}(\mathrm{roll}[\tau](v)) \rightarrow v$$

$$(E-FIX)\mathrm{fix}\,x : \tau.\mathrm{v} \rightarrow [x \mapsto \mathrm{fix}\,x : \tau.\mathrm{v}]\mathrm{v}$$

$$(E-CASE)\mathrm{case}(b, \mathrm{union}(b, \tau_1, \tau_2, v)) \rightarrow v$$

$$(E-INDOMAIN)\mathrm{in\_domain}(I_1, I_2, fact, fact) \rightarrow^\wedge \langle \mathrm{know}(0 \leq I_1 \,\wedge\, I_1 < I_2), fact\rangle$$

$$(E-DISTINGUISH)\text{distinguish}(I_1, I_2, fact, fact) \to^{\wedge} \langle \text{know}(I_1 \neq I_2), fact, fact\rangle$$

$$(E-MAKEEQ)\text{make\_eq}(\tau) \to fact$$

$$(E-APPLYEQ)\text{apply\_eq}(\tau, fact, v) \to v$$

$$(E-NEWFUN)\text{new\_fun}(K) \to \text{pack}[F^K, fact]\,\text{as}\,\exists\alpha : \text{int} \mapsto K.\text{FunGen}(\alpha, 0),\ \text{where}\ \text{F}^K\ \text{is fresh}$$

$$(E-DISCARDFUN)\text{discard\_fun}(fact) \to \cdot\langle\rangle$$

$$(E-DEFINEFUN)\text{define\_fun}(fact, \tau) \to^{\wedge} \langle fact, fact, fact\rangle$$

Rather than writing each of the congruence rules separately ($\frac{e_2 \to e_2'}{v_1\ e_2 \to v_1\ e_2'}$, etc.), we use E to indicate an expression with one (shallowly dug) hole in it, and $E[e]$ to indicate the expression with the hole replaced by e, so that a single evaluation rule covers all the cases. This is only for notational convenience.

$$E[e] = e\tau \,|\, \text{pack}[\tau_1, e]\,\text{as}\,\exists\alpha : K; B.\tau_2 \,|\, \text{unpack}\,\alpha, x = e\,\text{in}\,e_2 \,|\, \text{roll}[\tau](e) \,|\, \text{unroll}(e)$$

$$|\, \text{coerce}(e) \,|\, ee_2 \,|\, v_1 e \,|\, \phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle \,|\, \text{let}\ \overrightarrow{x} = e\,\text{in}\,e_2$$

$$|\, e\,\text{op}\,e_2 \,|\, v_1\,\text{op}\,e \,|\, \neg e \,|\, \text{if}\,e\,\text{then}\,e_2\,\text{else}\,e_3 \,|\, \text{union}(b, \tau_1, \tau_2, e) \,|\, \text{case}(b, e) \,|\, \text{load}(e, e_{\text{Has}})$$

$$|\, \text{load}(v_{\text{ptr}}, e) \,|\, \text{store}(e, e_{\text{Has}}, e_v) \,|\, \text{store}(\text{v}_{\text{ptr}}\ , e, e_v) \,|\, \text{store}(\text{v}_{\text{ptr}}, \text{v}_{\text{Has}}, e)$$

$$|\, \text{distinguish}(I_1, I_2, e, e_2) \,|\, \text{distinguish}(I_1, I_2, \text{v}_1, e) \,|\, \text{apply\_eq}(\tau, e, e_2) \,|\, \text{apply\_eq}(\tau, \text{v}_1, e)$$

$$|\, \text{discard\_fun}(e) \,|\, \text{define\_fun}(e, \tau) \,|\, \text{in\_domain}(I_1, I_2, e, e_2) \,|\, \text{in\_domain}(I_1, I_2, \text{v}_1, e)$$

$$(congruence\,rule)\frac{(M, e) \to (M', e')}{(M, E[e]) \to (M', E[e'])}$$

$$\frac{e \to e'}{(M, e) \to (M, e')}$$

4 Type well-formedness

$$(K-IVAR)\Phi; \Delta \vdash i : \text{int}$$

$$(K-TVAR)\Phi; \Delta, \alpha : K \vdash \alpha : K$$

$$(K-BOOL)\Phi; \Delta \vdash b : \text{bool}$$

$$(K-IOP)\frac{\Phi; \Delta \vdash I_1 : \text{int} \quad \Phi; \Delta \vdash I_2 : \text{int}}{\Phi; \Delta \vdash I_1\,\text{iop}\,I_2 : \text{int}}$$

$$(K-CMP)\frac{\Phi; \Delta \vdash I_1 : \text{int} \quad \Phi; \Delta \vdash I_2 : \text{int}}{\Phi; \Delta \vdash I_1\,\text{cmp}\,I_2 : \text{bool}}$$

$$(K-BOP)\frac{\Phi;\Delta\vdash B_1:\mathrm{bool}\quad \Phi;\Delta\vdash B_2:\mathrm{bool}}{\Phi;\Delta\vdash B_1\,\mathrm{bop}\,B_2:\mathrm{bool}}$$

$$(K-ANTI)\frac{\Phi;\Delta\vdash B:\mathrm{bool}}{\Phi;\Delta\vdash\neg B:\mathrm{bool}}$$

$$(K-TABS)\frac{\Phi;\Delta,\alpha:K_a\vdash\tau:K_b}{\Phi;\Delta\vdash\lambda\alpha:K_a.\tau:K_a\rightarrow K_b}$$

$$(K-APP)\frac{\Phi;\Delta\vdash\tau_f:K_a\rightarrow K_b\quad \Phi;\Delta\vdash\tau_a:K_a}{\Phi;\Delta\vdash\tau_f\tau_a:K_b}$$

$$(K-ALL)\frac{\Phi;\Delta,\alpha:K\vdash B:\mathrm{bool}\quad \Phi;\Delta,\alpha:K\vdash\tau:\overset{\phi}{i}}{\Phi;\Delta\vdash\forall\alpha:K;B.\tau:\overset{\phi}{i}}$$

$$(K-SOME)\frac{\Phi;\Delta,\alpha:K\vdash B:\mathrm{bool}\quad \Phi;\Delta,\alpha:K\vdash\tau:\overset{\phi}{i}}{\Phi;\Delta\vdash\exists\alpha:K;B.\tau:\overset{\phi}{i}}$$

$$(K-REC)\frac{\Phi;\Delta,\alpha:K\vdash\tau:K}{\Phi;\Delta\vdash\mu\alpha:K.\tau:K}$$

$$(K-ABS)\frac{\begin{array}{c}\Phi;\Delta\vdash\tau_1:\overset{\phi_1}{i_1}\quad \Phi;\Delta\vdash\tau_2:\overset{\phi_2}{i_2}\\ (\Phi;\Delta\vdash\mathrm{limit}:\mathrm{int}\Rightarrow i=0)\ \mathit{or}\ (\mathrm{limit}=\infty\Rightarrow i=1)\end{array}}{\Phi;\Delta\vdash\tau_1\overset{\phi,limit}{\longrightarrow}\tau_2:\overset{\phi}{i}}$$

$$(K-NLTUPLE)\frac{\forall j.(\Phi;\Delta\vdash\tau_j:\dot{i_j})}{\Phi;\Delta\vdash\cdot\langle\overrightarrow{\tau}\rangle:\dot{i}}(i=\sum_j i_j)$$

$$(K-LTUPLE)\frac{\forall j.(\Phi;\Delta\vdash\tau_j:\overset{\phi_j}{i_j})}{\Phi;\Delta\vdash^{\wedge}\langle\overrightarrow{\tau}\rangle:\hat{i}}(i=\sum_j i_j)$$

$$(K-INT)\frac{\Phi;\Delta\vdash I:\mathrm{int}}{\Phi;\Delta\vdash\mathrm{Int}(I):\dot{1}}$$

$$(K-BOOL)\frac{\Phi;\Delta\vdash B:\mathrm{bool}}{\Phi;\Delta\vdash\mathrm{Bool}(B):\dot{1}}$$

$$(K-UNION)\frac{\Phi;\Delta\vdash B:\mathrm{bool}\quad \Phi;\Delta\vdash\tau_1:\overset{\phi}{i}\quad \Phi;\Delta\vdash\tau_2:\overset{\phi}{i}}{\Phi;\Delta\vdash\mathrm{Union}(B,\tau_1,\tau_2):\overset{\phi}{i}}$$

$$(K-HAS)\frac{\Phi;\Delta\vdash I:\mathrm{int}\quad \Phi;\Delta\vdash\tau:\dot{1}}{\Phi;\Delta\vdash\mathrm{Has}(I,\tau):\hat{0}}$$

$$(K-FUNGEN)\frac{\Phi;\Delta\vdash I:\mathrm{int}\quad \Phi;\Delta\vdash\tau:\mathrm{int}\rightarrow J}{\Phi;\Delta\vdash\mathrm{Gen}(\tau,I):\hat{0}}$$

$$(K-INDOMAIN)\frac{\Phi;\Delta\vdash I:\mathrm{int}\quad \Phi;\Delta\vdash\tau:\mathrm{int}\rightarrow J}{\Phi;\Delta\vdash\mathrm{InDomain}(I,\tau):\dot{0}}$$

$$(K-EQ)\frac{\Phi;\Delta\vdash\tau_1:K\quad \Phi;\Delta\vdash\tau_2:K}{\Phi;\Delta\vdash\mathrm{Eq}(\tau_1,\tau_2):\dot{0}}$$

$$(K-FUN)\Phi,\mathrm{F}^K\overset{\phi}{\mapsto}\delta;\Delta\vdash\mathrm{F}^K:\mathrm{int}\rightarrow K$$

4.1 Type checking rules

$$(T-MEM)\frac{\begin{array}{c}\Psi = \Psi_{spare}, \Psi_e \qquad \Phi = \dot{\Phi}_{spare}, \Phi_e \\ \forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M(i) : \Psi(i))) \\ \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau\end{array}}{\Psi; \Phi; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)}$$

$$(T-VAR)\ \dot{C}, x : \tau \vdash x : \tau$$

$$(T-TABS)\frac{C, \alpha : K, B \vdash v : \tau \quad C, \alpha : K \vdash B : \text{bool}}{C \vdash \Lambda\alpha : K; B.v : \forall\alpha : K; B.\tau}$$

$$(T-TAPP)\frac{\begin{array}{c}C \vdash e : \forall\alpha : K; B.\tau_1 \\ C \vdash \tau_2 : K \\ C \vdash [\alpha \mapsto \tau_2]B\end{array}}{C \vdash e\tau_2 : [\alpha \mapsto \tau_2]\tau_1}$$

$$(T-COERCE)\frac{\begin{array}{c}\Psi; \Phi; \Delta; \Gamma; B; I_2 \vdash e : \tau \\ (\text{limit}_1 = \infty \wedge \tau \overset{\phi}{:} 0) \text{ or } (\text{limit}_1 = I_1 \wedge B \vdash I_1 > I_2)\end{array}}{\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e) : \tau}$$

$$(T-EQ)\frac{\begin{array}{c}C \vdash e : \tau_1 \quad C \vdash \tau_1 \equiv \tau_2 \\ C \vdash \tau_1 : K_1 \quad C \vdash \tau_2 : K_1\end{array}}{C \vdash e : \tau_2}$$

$$(T-TUPLE)\frac{\begin{array}{c}C = \dot{C}, C_1, \cdots, C_n \\ \forall i.(C_i \vdash e_i : \tau_i) \quad C \vdash \phi\langle\overrightarrow{\tau}\rangle : K\end{array}}{C \vdash \phi\langle\overrightarrow{e}\rangle : \phi\langle\overrightarrow{\tau}\rangle}$$

$$(T-ABS)\frac{\begin{array}{c}\overset{\phi}{\Phi}; \Delta \vdash \tau_1 : K \qquad \overset{\phi}{\Psi}; \overset{\phi}{\Phi}; \Delta; \overset{\phi}{\Gamma}, x : \tau_1; B; \text{limit}_2 \vdash e : \tau_2 \\ (\text{limit}_2 = \infty) \text{ or } (\overset{\phi}{\Phi}; \Delta \vdash \text{limit}_2 : \text{int} \quad B \vdash \text{limit}_2 \geq 0)\end{array}}{\overset{\phi}{\Psi}; \overset{\phi}{\Phi}; \Delta; \overset{\phi}{\Gamma}; B; \text{limit}_1 \vdash \lambda x : \tau_1 \overset{\phi, \text{limit}_2}{\longrightarrow} e : \tau_1 \overset{\phi, \text{limit}_2}{\longrightarrow} \tau_2}$$

$$(T-APP)\frac{\begin{array}{c}C_1, C_2 = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}_C \\ C_1 \vdash e_1 : \tau_a \overset{\phi, \text{limit}_f}{\longrightarrow} \tau_b \quad C_2 \vdash e_2 : \tau_a \\ (\text{limit}_C = \text{limit}_f = \infty) \text{ or } (B \vdash \text{limit}_f < \text{limit}_C) \\ \text{or } (\text{limit}_C = \infty,\ \text{limit}_f = I,\ \Phi; \Delta \vdash \tau_b \overset{\phi_b}{:} 0)\end{array}}{C_1, C_2 \vdash e_1 e_2 : \tau_b}$$

$$(T-LET)\frac{C_a \vdash e_a : \langle\overrightarrow{\tau}\rangle \quad C_b, \overrightarrow{x : \tau} \vdash e_b : \tau_b}{C_a, C_b \vdash \text{let}\langle\overrightarrow{x}\rangle = e_a \,\text{in}\, e_b : \tau_b}$$

$$(T-FIX)\frac{\begin{array}{c}C \vdash \tau \overset{\phi}{:} i \\ C, x : \tau \vdash \text{v} : \tau\end{array}}{C \vdash (\text{fix}\, x : \tau.\text{v}) : \tau}$$

$$(T-PACK)\frac{\begin{array}{c}C \vdash \tau_1 : K \quad C \vdash \exists\alpha : K; B.\tau_2 \overset{\phi}{:} i \\ C \vdash e : [\alpha \mapsto \tau_1]\tau_2 \quad C \vdash [\alpha \mapsto \tau_1]B\end{array}}{C \vdash \text{pack}[\tau_1, e] \,\text{as}\, \exists\alpha : K; B.\tau_2 : \exists\alpha : K; B.\tau_2}$$

$$
(T-UNPACK)\frac{\begin{array}{c}C_1 \vdash e_1 : \exists\alpha : K; B.\tau_1 \\ C_2, \alpha : K, x : \tau_1, B \vdash e_2 : \tau_2\end{array}}{C_1, C_2 \vdash \text{unpack}\,\alpha, x = e_1\,\text{in}\,e_2 : \tau_2}
$$

$$
(T-ROLL)\frac{\begin{array}{c}\tau = (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n \\ C \vdash \tau \overset{\phi}{:} i \\ C \vdash e : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n\end{array}}{C \vdash \text{roll}[\tau](e) : \tau}
$$

$$
(T-UNROLL)\frac{\begin{array}{c}\tau = (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n \\ C \vdash \tau \overset{\phi}{:} i \quad C \vdash e : \tau\end{array}}{C \vdash \text{unroll}(e) : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n}
$$

$$
(T-FACT1)\ \dot{C}, i \mapsto \tau \vdash fact : \text{Has}(i, \tau)
$$

$$
(T-FACT2)\ \dot{C}, F^K \overset{\wedge}{\mapsto} \delta \vdash fact : \text{Gen}(F^K, i)
$$

$$
(T-FACT3)\ \dot{C} \vdash fact : \text{Eq}(\tau, \tau)
$$

$$
(T-FACT4)\frac{i \in dom(\delta)}{\dot{C}\ , F^K \dot{\mapsto} \delta \vdash fact : \text{InDomain}(i, F^K)}
$$

$$
(T-LOAD)\frac{C_1 \vdash e_{\text{ptr}} : \text{Int}(I) \quad C_2 \vdash e_{\text{Has}} : \text{Has}(I, \tau)}{C_1, C_2 \vdash \text{load}(e_{\text{ptr}}, e_{\text{Has}}) :^{\wedge} \langle \tau, \text{Has}(I, \tau) \rangle}
$$

$$
(T-STORE)\frac{\begin{array}{c}C = C_1, C_2, C_3 = \Psi; \Phi; \Delta; \Gamma; B; \infty \\ C_2 \vdash e_{\text{Has}} : \text{Has}(I, \tau_1) \quad C_3 \vdash e_{\text{v}} : \tau_2 \\ C_1 \vdash e_{\text{ptr}} : \text{Int}(I) \quad C \vdash \tau_2 : \dot{1}\end{array}}{C \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \text{Has}(I, \tau_2)}
$$

$$
(T-DISTINGUISH)\frac{\begin{array}{c}C_1, C_2 \vdash I_1 : \text{int} \quad C_1, C_2 \vdash I_2 : \text{int} \\ C_1 \vdash e_1 : \text{Has}(I_1, \tau_1) \quad C_2 \vdash e_2 : \text{Has}(I_2, \tau_2)\end{array}}{C_1, C_2 \vdash \text{distinguish}(I_1, I_2, e_1, e_2) :^{\wedge} \langle \text{Know}(I_1 \neq I_2), \text{Has}(I_1, e_1), \text{Has}(I_2, e_2) \rangle}
$$

$$
(T-MAKEEQ)\frac{\dot{C} \vdash \tau : K}{\dot{C} \vdash \text{make\_eq}(\tau) : \text{Eq}(\tau, \tau)}
$$

$$
(T-APPLYEQ)\frac{\begin{array}{c}C \vdash \tau_f : K \to J \quad C \vdash e_1 : \text{Eq}(\tau_a, \tau_b) \\ C \vdash \tau_a : K \quad C \vdash \tau_b : K \quad C \vdash e_2 : \tau_f\tau_a\end{array}}{C \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau_f\tau_b}
$$

$$
(T-NEWFUN)\ \dot{C} \vdash \text{new\_fun}(J) : \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)
$$

$$
(T-DISCARDFUN)\frac{C \vdash e : \text{Gen}(\tau, I)}{C \vdash \text{discard\_fun}(e) : \cdot\langle\rangle}
$$

$$
(T-DEFINEFUN)\frac{C \vdash e : \text{Gen}(\tau_f, I) \quad C \vdash \tau_f : \text{int} \to J \quad C \vdash \tau_a : J}{C \vdash \text{define\_fun}(e, \tau_a) :^{\wedge} \langle \text{Gen}(\tau_f, I+1), \text{Eq}(\tau_f I, \tau_a), \text{InDomain}(I, \tau_f) \rangle}
$$

$$(T-INDOMAIN)\frac{\begin{array}{c}C_1, C_2 \vdash I_1 : \text{int} \quad C_1, C_2 \vdash I_2 : \text{int} \\ C_1 \vdash e_1 : \text{InDomain}(I_1, \tau_f) \quad C_2 \vdash e_2 : \text{Gen}(\tau_f, I_2)\end{array}}{C_1, C_2 \vdash \text{in\_domain}(I_1, I_2, e_1, e_2) :^\wedge \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle}$$

$$(T-INT)\ \dot{C} \vdash i : \text{Int}(i)$$

$$(T-BOOL)\ \dot{C} \vdash b : \text{Bool}(b)$$

$$(T-IOP)\frac{C_1 \vdash e_1 : \text{Int}(I_1) \quad C_2 \vdash e_2 : \text{Int}(I_2)}{C_1, C_2 \vdash e_1 \,\text{iop}\, e_2 : \text{Int}(I_1 \,\text{iop}\, I_2)}$$

$$(T-CMP)\frac{C_1 \vdash e_1 : \text{Int}(I_1) \quad C_2 \vdash e_2 : \text{Int}(I_2)}{C_1, C_2 \vdash e_1 \,\text{cmp}\, e_2 : \text{Bool}(I_1 \,\text{cmp}\, I_2)}$$

$$(T-BOP)\frac{C_1 \vdash e_1 : \text{Bool}(B_1) \quad C_2 \vdash e_2 : \text{Bool}(B_2)}{C_1, C_2 \vdash e_1 \,\text{bop}\, e_2 : \text{Bool}(B_1 \,\text{bop}\, B_2)}$$

$$(T-CBOOL)\frac{C \vdash e : \text{Bool}(B)}{C \vdash \neg e : \text{Bool}(\neg B)}$$

$$(T-IFE)\frac{C_a \vdash e_1 : \text{Bool}(B) \quad C_b, B \vdash e_2 : \tau \quad C_b, \neg B \vdash e_3 : \tau}{C_a, C_b \vdash \text{if}\, e_1 \,\text{then}\, e_2 \,\text{else}\, e_3 : \tau}$$

$$(T-IFB)\frac{\begin{array}{c}C = \Psi; \Phi; \Delta; \Gamma; B_C; I \\ C \vdash B : \text{bool} \quad C, B \vdash e_1 : \tau \quad C, \neg B \vdash e_2 : \tau\end{array}}{C \vdash \text{if}\, B \,\text{then}\, e_1 \,\text{else}\, e_2 : \tau}$$

$$(T-UNION)\frac{C \vdash \tau_1 : K \quad C \vdash \tau_2 : K \quad C \vdash e : \tau_i \quad (i = \begin{array}{c} 1\, if\, b \\ 2\, if\, \neg b \end{array})}{C \vdash \text{union}(b, \tau_1, \tau_2, e) : \text{Union}(b, \tau_1, \tau_2)}$$

$$(T-CASE)\frac{C \vdash e : \text{Union}(b, \tau_1, \tau_2)}{C \vdash \text{case}(b, e) : \tau_i \quad (i = \begin{array}{c} 1\, if\, b \\ 2\, if\, \neg b \end{array})}$$

5 Type erasure rules

The erasure rules are defined only for well-typed terms: to erase types, we require a derivation of some judgment $C \vdash (M, e : \tau)$ to be given, and we use the derivation to annotate subexpressions inside (M, e) with types. Specifically, if $e = ...e_1...$, and the derivation of $C \vdash (M, e : \tau)$ contains the judgment $C_1 \vdash e_1 : \tau_1$, then we annotate e with τ_1 as $e = ...(e_1 : \tau_1)...$ when convenient. We annotate types with kinds in a similar fashion. These annotations guide the erasure rules.

$$(ER-M)\,\text{erase}((M, e)) = (\,\text{erase}(M),\, \text{erase}(e))$$

$$(ER-M2)\,\text{erase}(M(e)) = (M(\,\text{erase}(e)))$$

$$(ER-i)\,\text{erase}(i) = i$$

$$(ER-b)\,\text{erase}(b) = b$$

$$(ER-x)\,\text{erase}(x) = x$$

$$(ER-APP)\,\mathrm{erase}((e_1 : (\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2))\, e_2) = \mathrm{erase}(e_1)\,\mathrm{erase}(e_2)$$

$$(ER-APPI)\,\mathrm{erase}((e_1 : (\tau_1 \overset{\phi,I}{\mapsto} \tau_2))\, e_2) = \mathrm{let}\, x = \langle\, \mathrm{erase}(e_1),\, \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle$$

$$(ER-APPT)\,\mathrm{erase}(e\,\tau) = \mathrm{erase}(e)$$

$$(ER-TUPLE)\,\mathrm{erase}(\langle e_1, \ldots, e_n\rangle : t :\overset{\phi}{i}) = \langle\, \mathrm{erase}(e_1), \ldots,\, \mathrm{erase}(e_n)\rangle\, \mathrm{where}\, i > 0$$

$$(ER-TUPLEv0)\,\mathrm{erase}(\langle v_1, \ldots, v_n\rangle : t :\overset{\phi}{0}) = \langle\rangle$$

$$(ER-TUPLEe0)\,\mathrm{erase}(\langle e_1, \ldots, e_n\rangle : t :\overset{\phi}{0}) = \mathrm{let}\, x = \langle\, \mathrm{erase}(e_1), \ldots,\, \mathrm{erase}(e_n)\rangle\, \mathrm{in}\, \langle\rangle \text{ where some } e_i \text{ is a non value.}$$

$$(ER-FUN)\lambda x : \tau \xrightarrow{\phi,\infty} e = \lambda x \longrightarrow \mathrm{erase}(e)$$

$$(ER-FUNI)\lambda x : \tau \xrightarrow{\phi,I} e = \langle\rangle$$

$$(ER-OP)\,\mathrm{erase}(e_1\, \mathrm{op}\, e_2) = \mathrm{erase}(e_1)\, \mathrm{op}\, \mathrm{erase}(e_2)$$

$$(ER-NOT)\,\mathrm{erase}(\neg e) = \neg\, \mathrm{erase}(e)$$

$$(ER-TFUN)\,\mathrm{erase}(\Lambda\alpha : K; B.v) = \mathrm{erase}(v)$$

$$(ER-LET)\,\mathrm{erase}(\mathrm{let}\, \langle \overrightarrow{x}\rangle = e_1\, \mathrm{in}\, e_2) = \mathrm{let}\, \langle \overrightarrow{x}\rangle = \mathrm{erase}(e_1)\, \mathrm{in}\, \mathrm{erase}(e_2)$$

$$(ER-IF)\,\mathrm{erase}(\mathrm{if}\, e_1\, \mathrm{then}\, e_2\, \mathrm{else}\, e_3) = \mathrm{if}\, \mathrm{erase}(e_1)\, \mathrm{then}\, \mathrm{erase}(e_2)\, \mathrm{else}\, \mathrm{erase}(e_3)$$

$$(ER-UNION)\,\mathrm{erase}(\mathrm{union}(b, \tau_1, \tau_2, e)) = \mathrm{erase}(e)$$

$$(ER-PACK)\,\mathrm{erase}(\mathrm{pack}[\tau_1, e]\, \mathrm{as}\, \exists\alpha : K; B.\tau_2) = \mathrm{erase}(e)$$

$$(ER-UNPACK)\,\mathrm{erase}(\mathrm{unpack}\, \alpha, x = e_1\, \mathrm{in}\, e_2) = \mathrm{let}\, x = \mathrm{erase}(e_1)\, \mathrm{in}\, \mathrm{erase}(e_2)$$

$$(ER-CASE)\,\mathrm{erase}(\mathrm{case}(b, e)) = \mathrm{erase}(e)$$

$$(ER-ROLL)\,\mathrm{erase}(\mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](e)) = \mathrm{erase}(e)$$

$$(ER-UNROLL)\,\mathrm{erase}(\mathrm{unroll}(e)) = \mathrm{erase}(e)$$

$$(ER-FIX)\,\mathrm{erase}(\mathrm{fix}\, x : t :\overset{\phi}{i}\, .v) = \mathrm{fix}\, x.\, \mathrm{erase}(v)\, \mathrm{where}\, i > 0$$

$$(ER-FIX0)\,\mathrm{erase}(\mathrm{fix}\, x : t :\overset{\phi}{0}\, .v) = \langle\rangle$$

$$(ER-LOAD)\,\mathrm{erase}(\mathrm{load}(e_{ptr}, e_{Has})) = \mathrm{load}(\, \mathrm{erase}(e_{ptr}),\, \mathrm{erase}(e_{Has}))$$

$$(ER - STORE)\,\text{erase}(\text{store}(e_{ptr}, e_{Has}, e_v)) = \text{store}(\,\text{erase}(e_{ptr}),\,\text{erase}(e_{Has}),\,\text{erase}(e_v))$$

$$(ER - COERCE)\,\text{erase}(\text{coerce}(e)) = \langle\rangle$$

$$(ER - DISTINGUISH)\,\text{erase}(\text{distinguish}(I_1, I_2, e_1, e_2)) = \text{let}\,\langle x, y\rangle = \langle\,\text{erase}(e_1),\,\text{erase}(e_2)\rangle\,\text{in}\,\langle\rangle$$

$$(ER - MAKEEQ)\,\text{erase}(\text{make\_eq}(\tau)) = \langle\rangle$$

$$(ER - APPLYEQ)\,\text{erase}(\text{apply\_eq}(\tau, e_1, e_2)) = \text{let}\,\langle x, y\rangle = \langle\,\text{erase}(e_1),\,\text{erase}(e_2)\rangle\,\text{in}\,y$$

$$(ER - NEWFUN)\,\text{erase}(\text{new\_fun}(K)) = \langle\rangle$$

$$(ER - DISCARDFUN)\,\text{erase}(\text{discard\_fun}(e)) = \text{let}\,x = \,\text{erase}(e)\,\text{in}\,\langle\rangle$$

$$(ER - DEFINEFUN)\,\text{erase}(\text{define\_fun}(e, \tau)) = \text{let}\,x = \,\text{erase}(e)\,\text{in}\,\langle\rangle$$

$$(ER - INDOMAIN)\,\text{erase}(\text{in\_domain}(I_1, I_2, e_1, e_2)) = \text{let}\,\langle x, y\rangle = \langle\,\text{erase}(e_1),\,\text{erase}(e_2)\rangle\,\text{in}\,\langle\rangle$$

$$(ER - FACT)\,\text{erase}(\text{fact}) = \langle\rangle$$

5.1 Untyped evaluation rules

Some of the evaluation rules will work for both typed and untyped evaluation. These new rules apply to untyped evaluation when the standard evaluation rules cannot.

$$(U - LOAD)(L, \text{load}(i, \langle\rangle)) \to (L,\, \langle L(i), \langle\rangle\rangle)$$

$$(U - STORE)(L, \text{store}(i, \langle\rangle, u)) \to ([i \mapsto u]L, \langle\rangle)$$

$$(U - ABSAPP)(\lambda x \longrightarrow d_1)u_2 \to [x \mapsto u_2]d_1$$

$$(U - FIX)\text{fix}\,x.u \to [x \mapsto \text{fix}\,x.u]u$$

6 Type safety

This section proves the two properties constituting type safety: preservation and progress.

6.1 Basic Properties

6.1.1 LEMMA [WEAKENING FOR TERMS]

$(C_1 \subseteq C_2) \wedge (C_1 \vdash e : \tau) \Rightarrow (C_2 \vdash e : \tau)$. Proof by induction on the type derivation. The leaves of the tree are the interesting cases; they rely on the fact that C_2 only extends C_1 with nonlinear mappings.

6.1.2 LEMMA [WEAKENING FOR TYPES]

$(C_1 \subseteq C_2) \wedge (C_1 \vdash \tau : K) \Rightarrow (C_2, C_3 \vdash \tau : K)$. Proof by induction on the kinding derivation.

6.1.3 LEMMA [SPLIT SUBSET]

If $C = C_1, C_2$ and $C_1 \stackrel{\cdot}{\subseteq} C_1'$, then there is some C_2' such that $C_2 \stackrel{\wedge}{\subseteq} C_2'$ and $C_1, C_2 \stackrel{\cdot}{\subseteq} C_1', C_2'$.

If $C = C_1, C_2$ and $C_1 \stackrel{\cdot}{\subseteq} C_1'$, then there is some C_2' such that $C_2 \subseteq C_2'$ and $C_1, C_2 \stackrel{\cdot}{\subseteq} C_1', C_2'$. Proof: if $C_1 \stackrel{\cdot}{\subseteq} C_1'$, then C_1' can differ from C_1 only in three ways (the proofs for the three cases can easily be combined into a single proof):

- For some $F^K \stackrel{\phi}{\mapsto} \delta$, it can add new elements to δ. In this case, simply add the same elements to the same δ in C_2 (by the definition of splitting C_1, C_2, we know C_2 contains the same δ) to get C_2'.
- It can change a $F^K \stackrel{\wedge}{\mapsto} \delta$ to $F^K \stackrel{\cdot}{\mapsto} \delta$. In this case, C_2 must already contain $F^K \stackrel{\cdot}{\mapsto} \delta$, so choose $C_2' = C_2$.
- It can add a new $F^K \stackrel{\phi}{\mapsto} \delta$, which appears neither in C_1 nor C_2. In this case, add $F^K \stackrel{\cdot}{\mapsto} \delta$ to C_2 to get C_2'.

6.2 Preservation

6.2.1 LEMMA [INVERSION]

In this lemma, $C = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}_C$

1. (VAR) If $C \vdash x : \tau$, then $x : \tau \in C$.
2. (APP) If $C \vdash e_1 e_2 : \tau$, then there is some type τ_a such that $C_1 \vdash e_1 : \tau_a \stackrel{\text{limit}_f}{\longrightarrow} \tau'$, $\tau \equiv \tau'$ and $C_2 \vdash e_2 : \tau_a$, ((limit$_C$ = limit$_f = \infty$) or (limit$_f <$ limit$_C$) or (limit$_C = \infty$, limit$_f = I$, $\Phi; \Delta \vdash \tau' \stackrel{\phi}{:} 0$)).
3. (ABS) If $C \vdash \lambda x : \tau_x \stackrel{\phi,\text{limit}_f}{\longrightarrow} e : \tau_1 \stackrel{\text{limit}_f}{\longrightarrow} \tau_2$, then $\tau_1 \equiv \tau_x$ and $\Psi; \Phi; \Delta; \Gamma, x : \tau_x; B; \text{limit}_f \vdash e : \tau_2$. ($C \vdash \tau_1 :: K_1$, $C \vdash \tau_x :: K_1$), (limit$_f = \infty$) or ($\Phi; \Delta \vdash \text{limit}_f : \text{int}\ \ B \vdash \text{limit}_f \geq 0$)
4. (TAPP) If $C \vdash e_1 \tau_2 : \tau$, then there is $[\alpha \mapsto \tau_2]\tau_1 \equiv \tau$, and $C \vdash e_1 : \forall \alpha : K; B.\tau_1$, $C \vdash \tau_2 : K$, $C \vdash [\alpha \mapsto \tau_2]B$.
5. (TABS) If $C \vdash \Lambda \alpha : K_\alpha; B_\alpha.v : \forall \alpha : K; B'.\tau'$, then $K_\alpha = K$ and $C, \alpha : K_\alpha, B' \vdash v : \tau'$ and $C, \alpha : K_\alpha \vdash B' : \text{bool}$, $B' \equiv B_\alpha$, $\tau' \equiv \tau$.
6. (TUPLE)If $C \vdash \phi\langle \overrightarrow{e} \rangle : \tau$, then there is $\phi\langle \overrightarrow{\tau'} \rangle \equiv \tau$, and $C \vdash \phi\langle \overrightarrow{e} \rangle : \phi\langle \overrightarrow{\tau'} \rangle$, $\forall i.(C_i \vdash e_i : \tau_i')$, $C \vdash \phi\langle \overrightarrow{\tau'} \rangle : K$.
7. (PROJECT)If $C \vdash \text{let}\,\langle \overrightarrow{x} \rangle = e_a \,\text{in}\, e_b : \tau$, then there is $\tau_b \equiv \tau$, and $C \vdash \text{let}\,\langle \overrightarrow{x} \rangle = e_a \,\text{in}\, e_b : \tau_b$, and $C_{e_a} \vdash e_a : \phi\langle \overrightarrow{\tau'} \rangle$, $C_{e_b}, \overrightarrow{x : \tau'} \vdash e_b : \tau_b$.
8. (LOAD)If $C \vdash \text{load}(e_{\text{ptr}}, e_{\text{Has}}) : \tau$, then there is ${}^\wedge\langle \tau', \text{Has}(I, \tau') \rangle \equiv \tau$ and $C \vdash \text{load}(e_{\text{ptr}}, e_{\text{Has}}) :{}^\wedge \langle \tau', \text{Has}(I, \tau') \rangle$, $C_{e_{\text{ptr}}} \vdash e_{\text{ptr}} : \text{Int}(I)$, $C_{e_{\text{Has}}} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$, $(C = C_{e_{\text{ptr}}}, C_{e_{\text{Has}}})$.
9. (STORE) $C \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \tau$, then there is $\text{Has}(I, \tau_2) \equiv \tau$, and $C_{e_{\text{ptr}}} \vdash e_{\text{ptr}} : \text{Int}(I)$, $C_{e_{\text{has}}} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$, $C_{e_{\text{v}}} \vdash e_{\text{v}} : \tau_2$, $C \vdash \tau_2 \stackrel{\cdot}{:} 1$.
10. (FIX)$C \vdash \text{fix}\, x : \tau_x.v : \tau$, then $\tau \equiv \tau_x$ and $C, x : \tau_x \vdash v : \tau$, $C \vdash \tau \stackrel{\phi}{:} i$
11. (UNROLL)$C \vdash \text{unroll}(e_0) : \tau$, then there is $([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n \equiv \tau$, and $C \vdash \tau \stackrel{\phi}{:} i$, $C \vdash e_0 : (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n$
12. (ROLL)$C \vdash \text{roll}[\tau](e) : \tau'$, then there is $(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n \equiv \tau' \equiv \tau$, and $C \vdash \tau \stackrel{\phi}{:} i$, $C \vdash e : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$
13. (UNPACK)$C_{e_1}, C_{e_2} \vdash \text{unpack}\, \alpha, x = e_1 \,\text{in}\, e_2 : \tau$, then there is $\tau' \equiv \tau$, and $C_{e_1}, C_{e_2} \vdash \text{unpack}\, \alpha, x = e_1 \,\text{in}\, e_2 : \tau'$, $C_{e_1} \vdash e_1 : \exists \alpha : K; B_\alpha.\tau_1$, $C_{e_2}, \alpha : K, x : \tau_1, B_\alpha \vdash e_2 : \tau'$, $C \vdash \tau' : K_2$

14. (PACK)$C \vdash \text{pack}[\tau_1, e]\,\text{as}\, \exists\alpha : K; \tau_B.\tau_2' : \tau$, then $\tau \equiv \exists\alpha : K; B_\alpha.\tau_2$, $\tau_2' \equiv \tau_2$, $\tau_B \equiv B_\alpha$ and $C \vdash \tau_1 : K$, $C \vdash [\alpha \mapsto \tau_1]\tau_B$, $C \vdash e : [\alpha \mapsto \tau_1]\tau_2'$, $C \vdash \exists\alpha : K; \tau_B.\tau_2' \overset{\phi}{:} i$

15. (DISTINGUISH)$C \vdash \text{distinguish}(I_1, I_2, e_1, e_2) : \tau$, then there is $\wedge\langle \text{Know}(I_1 \neq I_2), \text{Has}(I_1, \tau_1), \text{Has}(I_2, \tau_2)\rangle \equiv \tau$, and $C_{e_1} \vdash e_1 : \text{Has}(I_1, \tau_1)$, $C_{e_2} \vdash e_2 : \text{Has}(I_2, \tau_2)$, $C \vdash I_1 : \text{int}$, $C \vdash I_2 : \text{int}$

16. (NEWFUN)$\dot{C} \vdash \text{new\_fun}(J) : \tau$, then $\tau \equiv \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)$

17. (DEFINEFUN)$C \vdash \text{define\_fun}(e, \tau_a) : \tau$, then there is $\wedge\langle \text{Gen}(\tau_f, I+1), \text{Eq}(\tau_f I, \tau_a), \text{InDomain}(I, \tau_f)\rangle \equiv \tau$ and $C \vdash e : \text{Gen}(\tau_f, I)$, $C \vdash \tau_f : \text{int} \to J$, $C \vdash \tau_a : J$

18. (DISCARDFUN)$C \vdash \text{discard\_fun}(e_0) : \tau'$, then $\tau' \equiv \cdot\langle\rangle$ and $C \vdash e_0 : \text{Gen}(\tau, I)$

19. (INDOMAIN)$C \vdash \text{in\_domain}(I_1, I_2, e_1, e_2) : \tau$, then there is $\wedge\langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle \equiv \tau$ and $C \vdash I_1 : \text{int}$, $C \vdash I_2 : \text{int}$, $C_{e_1} \vdash e_1 : \text{InDomain}(I_1, \tau_f)$, $C_{e_2} \vdash e_2 : \text{Gen}(\tau_f, I_2)$

20. (MAKEEQ)$\dot{C} \vdash \text{make\_eq}(\tau_0) : \tau$, then $\text{Eq}(\tau_0, \tau_0) \equiv \tau$ and $\dot{C} \vdash \tau_0 : K$

21. (APPLYEQ)$C \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau$, then there is τ_b, $\tau_f\tau_b \equiv \tau$ and $C \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau_f\tau_b$, $C \vdash \tau_f : K \to J$, $C \vdash \tau_a : K$, $C \vdash \tau_b : K$, $C \vdash e_1 : \text{Eq}(\tau_a, \tau_b)$, $C \vdash e_2 : \tau_f\tau_a$

22. (CASE)$C \vdash \text{case}(b, e) : \tau$, then $\tau \equiv \tau_i$ $\left(i = \begin{matrix} 1\,if\,b \\ 2\,if\,\neg b \end{matrix}\right)$ and $C \vdash e : \text{Union}(b, \tau_1, \tau_2)$

23. (UNION)$C \vdash \text{union}(\tau_b, \tau_1', \tau_2', e) : \text{Union}(b, \tau_1, \tau_2)$, then $\tau_1 \equiv \tau_1'$, $\tau_2 \equiv \tau_2'$, $\tau_b \equiv b$, and $C \vdash \tau_1' : K$, $C \vdash \tau_2' : K$, $C \vdash e : \tau_i$' $\left(i = \begin{matrix} 1\,if\,\tau_b \\ 2\,if\,\neg\tau_b \end{matrix}\right)$

24. (COERCE)$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e) : \tau$, then there is $\tau' \equiv \tau$, and $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e) : \tau$, $\Psi; \Phi; \Delta; \Gamma; B; I_2 \vdash e : \tau'$, $((\text{limit}_1 = \infty \wedge \tau \overset{\phi}{:} 0)$ or $(\text{limit}_1 = I_1 > I_2))$.

Proof :

All the proofs are similiar, so we only prove some cases as example.

1. (VAR) If $C \vdash x : \tau$, then $x : \tau \in C$.
For $C \vdash x : \tau$, we only have two type checking rules to use. (T-EQ) and (T-VAR).
If using (T-EQ), then there will be τ_n. For τ_n, we will use (T-VAR) for $C \vdash x : \tau_n$. $(\tau_1 \equiv \cdots \equiv \tau_n \equiv \tau)$
If using (T-VAR), then τ_n is τ.
By (T-VAR) rule, we obtain $\dot{C}_0, x : \tau_n \vdash x : \tau_n$, then $C = \dot{C}_0, x : \tau_n$
By (T-EQ) rule, thus $x : \tau \in C$.

2. (APP) If $C \vdash e_1e_2 : \tau$, then there is some type τ_a such that $C_1 \vdash e_1 : \tau_a \xrightarrow{\text{limit}_f} \tau'$, $\tau \equiv \tau'$ and $C_2 \vdash e_2 : \tau_a$, $((\text{limit}_C = \text{limit}_f = \infty)$ or $(\text{limit}_f < \text{limit}_C)$ or $(\text{limit}_C = \infty, \text{limit}_f = I, \Phi; \Delta \vdash \tau' \overset{\phi}{:} 0))$.
For $C \vdash e_1e_2 : \tau$, we only have two type checking rules to use. (T-EQ) and (T-APP).
If using (T-EQ), then there will be τ'. For τ', we will use (T-APP) for $C \vdash e_1e_2 : \tau'$. $(\tau \equiv \tau_1 \equiv \cdots \equiv \tau_n \equiv \tau')$
If using (T-APP), then τ' is τ.
By (T-APP) rule, we obtain $C \vdash e_1e_2 : \tau'$, $C_1 \vdash e_1 : \tau_a \xrightarrow{\text{limit}_f} \tau'$, and $C_2 \vdash e_2 : \tau_a$, $((\text{limit}_C = \text{limit}_f = \infty)$ or $(\text{limit}_f < \text{limit}_C)$ or $(\text{limit}_C = \infty, \text{limit}_f = I, \Phi; \Delta \vdash \tau' \overset{\phi}{:} 0))$.

We can use the similar proof to prove the other propositions.

6.2.2 LEMMA [LIMIT-CHANGE]

We define $s = v \,|\, \text{fix}\, x : \tau.v$.

If $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash s : \tau$, then $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash s : \tau$

Proof by induction on the type derivation:

1. $s = i$

then $\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash i : \tau$

By (T-INT) rule,

$\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_2 \vdash i : \tau$

2. $s = b$

then $\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash b : \tau$

By (T-BOOL) rule,

$\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_2 \vdash b : \tau$

3. $s = \lambda x : \tau_x \xrightarrow{\phi, \text{limit}_f} v$

$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \lambda x : \tau \xrightarrow{\phi, \text{limit}_f} v : \tau_x \xrightarrow{f} \tau_v$

By Lemma (6.2.1.(3)), we know $\tau \equiv \tau_x$, and $\Psi; \Phi; \Delta; \Gamma, x : \tau; B; \text{limit}_f \vdash v : \tau_v, \;\; \Phi; \Delta \vdash \tau : K$

By (T-ABS) rule, we obtain $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash \lambda x : \tau \xrightarrow{\phi, \text{limit}_f} v : \tau \xrightarrow{f} \tau_v$

4. $s = \text{fact}$

$\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \tau$

There are five typechecking tules can be used. One is (T-EQ), others are the following rules.

For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \tau'$,

and we will only use one rule of the following rules.

$case1 : \;\; i \mapsto \tau; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \text{Has}(i, \tau)$

By (FACT1) rule, $i \mapsto \tau; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_2 \vdash \text{fact} : \text{Has}(i, \tau)$

$case2 : \;\; \varnothing; \dot{\Phi}, F^K \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \text{Gen}(F^K, i)$

By (FACT2) rule, $\varnothing; \dot{\Phi}, F^K \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit}_2 \vdash \text{fact} : \text{Gen}(F^K, i)$

$case3 : \varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \text{Eq}(\tau, \tau)$

By (FACT3) rule, $\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_2 \vdash \text{fact} : \text{Eq}(\tau, \tau)$

$case4 : \varnothing; \dot{\Phi}, F^K \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \text{InDomain}(i, F^K)$

By (FACT4) rule, $\varnothing; \dot{\Phi}, F^K \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit}_2 \vdash \text{fact} : \text{InDomain}(i, F^K)$

5. $s = \Lambda \alpha : K; B.v$

$\Psi; \Phi; \Delta; \Gamma; B_1; \text{limit}_1 \vdash \Lambda \alpha : K; B.v : \tau$

$\Phi; \Delta \vdash B : \text{bool}$

There are two typechecking tules can be used. One is (T-EQ), the other is (T-TABS) rule.

For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\Psi; \Phi; \Delta; \Gamma; B_1; \text{limit}_1 \vdash \Lambda \alpha : K; B.v : \tau'$,

and we will only use the (T-TABS) rule.

By (T-TABS) rule, $\tau' = \forall \alpha : K; B.\tau_v$, and $\Psi; \Phi; \Delta, \alpha : K; \Gamma; B_1, B; \text{limit}_1 \vdash v : \tau_v$

By induction, we get $\Psi; \Phi; \Delta, \alpha : K; \Gamma; B_1, B; \text{limit}_2 \vdash v : \tau_v$

Then using (T-TABS) rule, $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_2 \vdash \Lambda\alpha : K;B.v : \forall\alpha : K;B.\tau_v$
By (T-EQ) rule, $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_2 \vdash \Lambda\alpha : K;B.v : \tau$

6. $s = \text{pack}[\tau_1, v]\,\text{as}\,\exists\alpha : K;B.\tau_2$
$\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_1 \vdash \text{pack}[\tau_1, v]\,\text{as}\,\exists\alpha : K;B.\tau_2 : \tau$
There are two typechecking tules can be used. One is (T-EQ), the other is (T-PACk) rule.
For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_1 \vdash \text{pack}[\tau_1, v]\,\text{as}\,\exists\alpha : K;B.\tau_2 : \tau'$,
and we will only use the (T-PACk) rule.
By (T-PACK) rule, we get:
$\tau' = \exists\alpha : K;B.\tau_2$, and $\Phi;\Delta \vdash \tau_1 : K$, $\Phi;\Delta \vdash \exists\alpha : K;B.\tau_2 \overset{\phi}{:} i$
$B_1 \vdash [\alpha \mapsto \tau_1]B$, $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_1 \vdash v : [\alpha \mapsto \tau_1]\tau_2$
By induction, we obtain $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_2 \vdash v : [\alpha \mapsto \tau_1]\tau_2$
and by (T-PACK) rule, thus $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_2 \vdash \text{pack}[\tau_1, v]\,\text{as}\,\exists\alpha : K;B.\tau_2 : \tau'$
By (T-EQ) rule, $\Psi;\Phi;\Delta;\Gamma;B_1;\text{limit}_2 \vdash \text{pack}[\tau_1, v]\,\text{as}\,\exists\alpha : K;B.\tau_2 : \tau$

7. $s = \text{roll}[\tau](v)$ $\tau = (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n$
By Lemma (6.2.1.(12)), we have $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash \text{roll}[\tau](v) : \tau$
$\Phi;\Delta \vdash \tau \overset{\phi}{:} i$,
$\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$
By induction, we obtain $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$
Then by (T-ROLL) rule, we have
$\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \text{roll}[\tau](v) : \tau$

8. $s = \text{union}(b, \tau_1, \tau_2, v)$
$\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash \text{union}(b, \tau_1, \tau_2, v) : \tau$
There are two typechecking tules can be used. One is (T-EQ), the other is (T-UNION) rule.
For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash \text{union}(b, \tau_1, \tau_2, v) : \tau'$,
and we will only use the (T-UNION) rule.
By (T-UNION) rule, we get:
$\Phi;\Delta \vdash \tau_1 : K$, $\Phi;\Delta \vdash \tau_2 : K$, $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash v : \tau_i$ $\left(i = \begin{array}{c} 1\,if\,b \\ 2\,if\,\neg b \end{array}\right)$
By induction, $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash v : \tau_i$
By (T-UNION) rule, $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \text{union}(b, \tau_1, \tau_2, v) : \tau'$
By (T-EQ) rule, thus $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \text{union}(b, \tau_1, \tau_2, v) : \tau$

9. $s = \phi\langle\overrightarrow{v}\rangle$
$\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash \phi\langle\overrightarrow{v}\rangle : \tau$
By Lemma (6.2.1.(6)), we have $\phi\langle\overrightarrow{\tau'}\rangle \equiv \tau$, and $\forall i.(\Psi_i;\Phi_i;\Delta;\Gamma_i;B;\text{limit}_1 \vdash v_i : \tau_i')$, $\Phi;\Delta \vdash \phi\langle\overrightarrow{\tau'}\rangle : K$
By induction, $\forall i.(\Psi_i;\Phi_i;\Delta;\Gamma_i;B;\text{limit}_2 \vdash v_i : \tau_i')$, then using (T-TUPLE) rule,
$\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \phi\langle\overrightarrow{v}\rangle : \phi\langle\overrightarrow{\tau'}\rangle$
By (T-EQ) rule, $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \phi\langle\overrightarrow{v}\rangle : \tau$

10. $s = \text{fix}\,x : \tau.v$
$\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_1 \vdash \text{fix}\,x : \tau.v : \tau'$
By Lemma (6.2.1.(10)), we know $\tau \equiv \tau'$, and
$\Psi;\Phi;\Delta;\Gamma, x : \tau;B;\text{limit}_1 \vdash v : \tau'$, $\Phi;\Delta \vdash \tau' \overset{\phi}{:} i$
By induction, $\Psi;\Phi;\Delta;\Gamma, x : \tau;B;\text{limit}_2 \vdash v : \tau'$

By (T-FIX) rule, $\Psi;\Phi;\Delta;\Gamma;B;\mathrm{limit}_2 \vdash \mathrm{fix}\, x : \tau.v : \tau'$

6.2.3 LEMMA [ARITHMETIC FORMS]

If $\Phi;\Delta \vdash \tau : \mathrm{bool}$, then $\tau = B$. Proof by induction on the kinding derivation for τ. Crucial case: $\tau = \tau_f\, \tau_a$, where τ_f has kind $K_a \to J_b$ (syntactically, J_b cannot be bool).

6.2.4 LEMMA [TYPE ENVIRONMENT SUBSTITUTION]

If $\Phi;\Delta \vdash \tau : K$, then $([\overrightarrow{\alpha \mapsto \tau_\alpha}]\Phi);\Delta \vdash \tau : K$.

Proof by induction on the kinding derivation. The only kinding rule that uses Φ is (K-FUN), which uses only the domain of Φ, which is unaffected by $[\overrightarrow{\alpha \mapsto \tau_\alpha}]$.

6.2.5 LEMMA [TYPE SUBSTITUTION 1]

If $C, \overrightarrow{\alpha : K_\alpha} \vdash \tau : K$, and $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$, then $C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$.

(Corollary, via the type environment substitution lemma: if $C, \overrightarrow{\alpha : K_\alpha} \vdash \tau : K$, and $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$, then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$.)

Proof by induction on the kinding derivation.

1. $\tau = \beta$
If $\beta = \alpha_i$, then $C, \overrightarrow{\alpha : K_\alpha} \vdash \tau : K$, $K = K_{\alpha_i}$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = \tau_{\alpha_i}$, then $C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K_{\alpha_i}$

If $\beta \neq \alpha_i$, then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = \beta$, and because $C, \overrightarrow{\alpha \mapsto \tau_\alpha} \vdash \beta : K$, we know $C(\beta) = K$.
By (K-TVAR) rule, $C \vdash \beta : K$.

2. $\tau = i$
then by (K-IVAR) rule, $\dot{C}, \overrightarrow{\alpha \mapsto \tau_\alpha} \vdash i : \mathrm{int} \Rightarrow \dot{C} \vdash i : \mathrm{int}$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = i$, thus $\dot{C} \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : \mathrm{int}$

3. $\tau = \lambda\beta : K_a.\tau_0$ (In this case, we can rename β, then $\alpha \neq \beta$)
By (K-TABS) rule, $C, \overrightarrow{\alpha : K_\alpha} \vdash \lambda\beta : K_a.\tau_0 : K_a \to K_b$, and $C, \beta : K_a, \overrightarrow{\alpha : K_\alpha} \vdash \tau_0 : K_b$
By induction, we obtain:
$(C, \beta : K_a) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0 : K_b$
$\Rightarrow C, \beta : K_a \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0 : K_b$
By (K-TABS) rule,
$C \vdash \lambda\beta : K_a.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0 : K_a \to K_b$
$\Rightarrow C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\lambda\beta : K_a.\tau_0 : K_a \to K_b$

4. $\tau = \tau_f \tau_a$
By (K-TAPP) rule, $C, \overrightarrow{\alpha : K_\alpha} \vdash \tau_f\tau_a : K_b$, and $C, \overrightarrow{\alpha : K_\alpha} \vdash \tau_f : K_a \to K_b$, $\quad C, \overrightarrow{\alpha : K_\alpha} \vdash \tau_a : K_a$
By induction, we have
$C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_f : K_a \to K_b$
$C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_a : K_a$
By (K-APP) rule,
$C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_f\tau_a : K_b$

5. $\tau = \forall\beta : K; B.\tau_0$

By (K-ALL) rule, $C, \overrightarrow{\alpha : K_\alpha} \vdash \forall\beta : K; B.\tau_0 \overset{\phi}{:} i$, and
$C, \beta : K, \overrightarrow{\alpha : K_\alpha} \vdash B : \text{bool}, \quad C, \beta : K, \overrightarrow{\alpha : K_\alpha} \vdash \tau_0 \overset{\phi}{:} i$
By induction,
$C, \beta : K \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]B : \text{bool}$
$C, \beta : K \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0 \overset{\phi}{:} i$
By (K-ALL) rule, we obtain
$C \vdash \forall\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0 \overset{\phi}{:} i$
$\Rightarrow C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\forall\beta : K; B.\tau_0 \overset{\phi}{:} i$

6. $\tau = F^K$
then by (K-FUN) rule, $C, F^K \dot{\mapsto} \delta, \overrightarrow{\alpha \mapsto \tau_\alpha} \vdash F^K : \text{int} \to K$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = F^K$,
By (K-FUN) rule,
$C, F^K \dot{\mapsto} \delta \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]F^K : \text{int} \to K$

7. In the same way, we can prove the other cases by induction.

6.2.6 LEMMA [TYPE SUBSTITUTION 2]

If $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B_1$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B_2$ are syntactically well-formed and $B_1 \vdash B_2$, then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B_1 \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]B_2$.

Proof: since $B_1 \vdash B_2$ means that $B_1 \Rightarrow B_2$ for all substitutions of integers for integer variables and booleans for boolean variables, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B_1 \Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]B_2$ for all remaining substitutions of of integers for integer variables and booleans for boolean variables.

6.2.7 LEMMA [TYPE SUBSTITUTION 3]

If $\Phi; \Delta, \overrightarrow{\alpha : K_\alpha} \vdash \tau_1 : K$ and $\Phi; \Delta, \overrightarrow{\alpha : K_\alpha} \vdash \tau_2 : K$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B$ is syntactically well-formed and $\Phi; B \vdash \tau_1 \equiv \tau_2$, and $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$, then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\Phi; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1 \equiv [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$.

Proof by induction on the type equivalence derivation. Example case:

1. $\dfrac{B \vdash I \doteq i}{\Phi, \mathrm{F}^K \dot{\mapsto} \delta; B \vdash \mathrm{F}^K I \equiv \delta(i)}$

By type substitution 1, we know $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\Phi; \Delta \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}](\mathrm{F}^K I) : K$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\Phi; \Delta \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\delta(i) : K$. From this, we know that $[\overrightarrow{\alpha \mapsto \tau_\alpha}]I$ is syntactically well-formed. By type substitution 2, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]I \doteq i$. Knowing this, we can use the type equivalence rule to conclude:

$$\frac{[\overrightarrow{\alpha \mapsto \tau_\alpha}]B \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]I \doteq i}{[\overrightarrow{\alpha \mapsto \tau_\alpha}](\Phi, \mathrm{F}^K \dot{\mapsto} \delta); [\overrightarrow{\alpha \mapsto \tau_\alpha}]B \vdash \mathrm{F}^K [\overrightarrow{\alpha \mapsto \tau_\alpha}]I \equiv ([\overrightarrow{\alpha \mapsto \tau_\alpha}]\delta)(i)}$$

6.2.8 LEMMA [TERM SUBSTITUTION]

If we make the following definitions and assumptions:

- We define a substitution $[s] = [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x}, \overrightarrow{y \mapsto s_y}, \overrightarrow{z \mapsto s_z}]$
- $C = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}_C$
- No α_i appears free in $\Psi; \Phi; \Delta$ (note: typically, this condition can be satisfied by alpha-renaming α_i before invoking this lemma).

- $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C = \Psi; \Phi; \Delta; [\overrightarrow{\alpha \mapsto \tau_\alpha}]\Gamma; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B; [\overrightarrow{\alpha \mapsto \tau_\alpha}]\mathrm{limit}_C$ is syntactically well-formed (this assumption is needed because substitutions of non-integers and non-booleans into integers I and booleans B do not always produce integers and booleans).
- $C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e : \tau$
- $z_i \notin domain(\Gamma)$
- $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C} \vdash s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$ (where τ_{x_i} are nonlinear types)
- $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{y_i}) \vdash s_{y_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{y_i}$ (where τ_{y_i} are linear types)
- $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{z_i}) \vdash s_{z_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{z_i}$ (where τ_{z_i} are linear types)
- $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$

then we can conclude the following:

- $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

Proof by induction on the type derivation. The cases that use induction on an environment with an extended Γ must weaken the s_{x_i}, s_{y_i}, s_{z_i} typings; we write this out explicitly for the $e = \lambda w : \tau \xrightarrow{\phi,\mathrm{limit}} e_0$ case but omit the details in the other cases.

1*a*. $e = x_0$, where $x_0 \neq x_i, y_i, z_i$
$\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash x_0 : \tau$. The $\overrightarrow{y}$ must be empty, and $\dot{C} = C = (C, \overrightarrow{C_{y_i}})$, and $C(x_0) = \tau$.
$[s]e = x_0$
Since $C(x_0) = \tau$, we know $([\overrightarrow{\alpha \mapsto \tau_\alpha}]C)(x_0) = [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$.
Thus, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

1*b*. $e = x_i$
$\tau = \tau_{x_i}$ and $\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash x_0 : \tau_{x_i}$. The $\overrightarrow{y}$ must be empty, and $\dot{C} = C = (C, \overrightarrow{C_{y_i}})$.
$[s]e = s_{x_i}$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\dot{C} \vdash s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$.
So $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$.

2*a*. $e = y_i$
$\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, y_i : \tau_{y_i} \vdash e : \tau_{y_i}$ and $\tau = \tau_{y_i}$. The $\overrightarrow{y}$ must be empty except for y_i, and $\dot{C} = C$ and $\overrightarrow{C_{y_i}} = C_{y_i}$.
$[s]e = s_{y_i}$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{y_i}) \vdash s_{y_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{y_i}$,
Thus, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{y_i}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

2*b*. $e = z_i$: cannot happen, because $C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e : \tau$ and $z_i \notin domain(\Gamma)$

3. $e = e_1 e_2$
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e_1 e_2 : \tau_2$
By Lemma (6.2.1.(2)), we have
$C_1, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y' : \tau_{y'}} \vdash e_1 : \tau_1 \xrightarrow{f} \tau_2 \quad C_2, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y'' : \tau_{y''}} \vdash e_2 : \tau_1$

By induction, we obtain:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \overrightarrow{C_{y'_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x}, \overrightarrow{y' \mapsto s_{y'}}, (\overrightarrow{z \mapsto s_z}, \overrightarrow{y'' \mapsto s_{y''}})]e_1 :$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](\tau_1 \xrightarrow{f} \tau_2) \Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \overrightarrow{C_{y'_i}}) \vdash [s]e_1 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1 \xrightarrow{f} [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_2, \overrightarrow{C_{y''_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x}, \overrightarrow{y'' \mapsto s_{y''}}, (\overrightarrow{z \mapsto s_z}, \overrightarrow{y' \mapsto s_{y'}})]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1$

$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_2, \overrightarrow{C_{y_i''}}) \vdash [s]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1$

By (T-APP) rule,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e_1([s]e_2) : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x},\ \overrightarrow{y \mapsto s_y},\ \overrightarrow{z \mapsto s_z}]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

4. $e = \lambda w : \tau \overset{\phi,\text{limit}}{\longrightarrow} e_0$
By Lemma (6.2.1.(3)) and (T-EQ) rule, we have
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \lambda w : \tau_w \overset{\phi,\text{limit}_2}{\longrightarrow} e_0 : \tau_w \overset{\phi,\text{limit}_2}{\longrightarrow} \tau_0$
and $C', w : \tau_w, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e_0 : \tau_0$, $\Phi; \Delta \vdash \tau_w : K$, (Here, $C' = \Psi, \Phi; \Delta; \Gamma; B; \text{limit}_2$)
Applying the limit change lemma to the typings for s_{x_i}, s_{y_i}, s_{z_i} gives:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C}' \vdash s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}', C'_{y_i}) \vdash s_{y_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{y_i}$and $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}', C'_{z_i}) \vdash s_{z_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{z_i}$.
and in $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$,we need $\Phi; \Delta$, then $C' \vdash \tau_{\alpha_i} : K_{\alpha_i}$. (We also use limit_2 in $\dot{C}', C', C'_{y_i}, C'_{z_i}$)
If τ_w is nonlinear, then by type substitution $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w$ is also nonlinear and so the Weakening Lemma says:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}', w : \tau_w) \vdash s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$ and $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}', C'_{y_i}, w : \tau_w) \vdash s_{y_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{y_i}$and $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}', C'_{z_i}, w : \tau_w) \vdash s_{z_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{z_i}$.
If τ_w is linear, weakening is not needed.
By induction, we have:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C', \overrightarrow{C'_{y_i}}), w : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w \vdash [s]e_0 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0$
By Type Substitution Lemma we know, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\Phi; \Delta) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w : K$
By (T-ABS) rule, we obtain:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w \overset{\phi,\text{limit}_2}{\longrightarrow} [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}](\tau_w \overset{\phi,\text{limit}_2}{\longrightarrow} \tau_0)$

5. $e = \text{pack}[\tau_1, e_0]\,\text{as}\,\exists\beta : K; B'.\tau_2$
By Lemma (6.2.1.(14)) and (T-EQ) rule, we have
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \text{pack}[\tau_1, e_0]\,\text{as}\,\exists\beta : K; B'.\tau_2 : \exists\beta : K; B'.\tau_2$, and
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \tau_1 : K \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash \tau_1 : K$,
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash [\beta \mapsto \tau_1]B' \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash [\beta \mapsto \tau_1]B'$,
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e_0 : [\beta \mapsto \tau_1]\tau_2$,
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \exists\beta : K; B'.\tau_2 : \overset{\phi}{i} \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash \exists\beta : K; B'.\tau_2 : \overset{\phi}{i}$
By induction, we get:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e_0 : [\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]\tau_2$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e_0 : [\beta \mapsto \tau_1]([\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2)$
By Type Substitution Lemma,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1 : K$,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}](\exists\beta : K; B'.\tau_2) : \overset{\phi}{i}$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash \exists\beta : K; ([\overrightarrow{\alpha \mapsto \tau_\alpha}]B').[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 : \overset{\phi}{i}$
$C, \overrightarrow{\alpha : K_\alpha} \vdash \exists\beta : K; B'.\tau_2 : \overset{\phi}{i} \Rightarrow C, \overrightarrow{\alpha : K_\alpha}, \beta : K \vdash B' : \text{bool} \Rightarrow [\beta \mapsto \tau_1]C, \overrightarrow{\alpha : K_\alpha} \vdash [\beta \mapsto \tau_1]B' : \text{bool} \Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]B' : \text{bool}$ (this implies that $[\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]B'$ is syntactically well-formed)
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]B'$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\beta \mapsto [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1]([\overrightarrow{\alpha \mapsto \tau_\alpha}]B')$,
By Weakening Lemma,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1 : K$,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\beta \mapsto [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1]([\overrightarrow{\alpha \mapsto \tau_\alpha}]B')$,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash \exists\beta : K; ([\overrightarrow{\alpha \mapsto \tau_\alpha}]B').[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 :\overset{\phi}{i}$

By (T-PACK) rule,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash \mathrm{pack}[[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1, [s]e_0]\,\mathrm{as}\,\exists\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B'.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 :$
$\exists\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B'.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$

Therefore, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash$
$[s](\mathrm{pack}[\tau_1, e_0]\,\mathrm{as}\,\exists\beta : K; B'.\tau_2) : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\exists\beta : K; B'.\tau_2$

6. $e = \mathrm{unpack}\,\beta, w = e_1\,\mathrm{in}\,e_2$

By Lemma (6.2.1.(13)) and (T-EQ) rule, we get

$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \mathrm{unpack}\,\beta, w = e_1\,\mathrm{in}\,e_2 : \tau_2$, and

$C_1, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y' : \tau_{y'}} \vdash e_1 : \exists\beta : K; B'.\tau_1,\;\; C \vdash \tau_2 : K_2$

$C_2, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y'' : \tau_{y''}}, \beta : K, w : \tau_1, B' \vdash e_2 : \tau_2$

By Type Substitution Lemma, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 : K_2$

By Weakening Lemma, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y'_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 : K_2$

By induction, we obtain:

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \overrightarrow{C_{y'_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x},\, \overrightarrow{y' \mapsto s_{y'}},\, \overrightarrow{z \mapsto s_z}, \,,\, \overrightarrow{y'' \mapsto s_{y''}}]e_1 : [\overrightarrow{\alpha \mapsto \tau_\alpha}](\exists\beta : K; B'.\tau_1)$

$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \overrightarrow{C_{y'_i}}) \vdash [s]e_1 : \exists\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B'.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1$ (this also proves that $[\overrightarrow{\alpha \mapsto \tau_\alpha}]B'$ is syntactically well-formed).

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_2, \overrightarrow{C_{y''_i}}, \beta : K, w : \tau_1, B') \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x},\, \overrightarrow{y'' \mapsto s_{y''}},\, \overrightarrow{z \mapsto s_z}, \,,\, \overrightarrow{y' \mapsto s_{y'}}]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$

$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_2, \overrightarrow{C_{y''_i}}), \beta : K, w : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1, [\overrightarrow{\alpha \mapsto \tau_\alpha}]B' \vdash [s]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$

By (T-UNPACK) rule, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash$
$\mathrm{unpack}\,\beta, w = [s]e_1\,\mathrm{in}\,[s]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s](\mathrm{unpack}\,\beta, w = e_1\,\mathrm{in}\,e_2) : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$

7. $e = \mathrm{fix}\,w : \tau.v$

By Lemma (6.2.1.(10)) and (T-EQ) rule, we get

$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \mathrm{fix}\,w : \tau.v : \tau$, and $C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y}, w : \tau \vdash v : \tau$

$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \tau :\overset{\phi}{i} \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash \tau :\overset{\phi}{i}$

By induction,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}, w : \tau) \vdash [s]v : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}), w : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau \vdash [s]v : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

By Type Substitution Lemma,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau :\overset{\phi}{i}$

By Weakening Lemma,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau :\overset{\phi}{i}$

By (T-FIX) rule,

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash \mathrm{fix}\,w : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau.[\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x},\, \overrightarrow{y \mapsto s_y},\, \overrightarrow{z \mapsto s_z}]v : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x},\, \overrightarrow{y \mapsto s_y},\, \overrightarrow{z \mapsto s_z}](\mathrm{fix}\,w : \tau.v) : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

8. In the same way, we can prove the other cases by induction.

6.2.9 THEOREM [PRESERVATION]

If $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau$, $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$ and $(M, e) \rightarrow (M', e')$, then $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$ and $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$, where $(\Phi'_e \dot{\supseteq} \Phi_e)$.

Prove by induction on the type derivation. The cases below omit most of the congruence rule cases, because the proofs for these all look more or less the same. See the tuple, load, and store cases for examples of the proofs for congruence rule cases.

1. $Case\, T-VAR$: $e = x$
There are no evaluation rules for variables

2. $Case\, T-ABS$: $e = \lambda x : \tau_1 \stackrel{\phi,\text{limit}}{\longrightarrow} e_0$
There are no evaluation rules for e (e is value)

3. $Case\, T-TABS$: $e = \Lambda\alpha : K; B.v$
There are no evaluation rules for e (e is value)

4. $Case\, T-APP$: $e = e_1\, e_2$ $\quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash e : \tau$
From lemma 6.2.1.(2), there is τ_1 and
$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit}_C \vdash e_1 : \tau_1 \stackrel{limit_f}{\longrightarrow} \tau'$,
$(\text{limit}_C = \text{limit}_f = \infty)$ or $(\text{limit}_f < \text{limit}_C)$ or $(\text{limit}_C = \infty$, $\text{limit}_f = I$, $\Phi_e; \Delta \vdash \tau' \stackrel{\phi}{:} 0)$
$\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit}_C \vdash e_2 : \tau_1$, $\quad \tau' \equiv \tau$

$(E-APPABS1)$
$e_1 = \lambda x : \tau_x \stackrel{\phi,I}{\longrightarrow} e_{12}$ $\quad e_2 = v_2$ $\quad e' = \text{coerce}([x \mapsto v_2]e_{12})$
By Lemma 6.2.1.(3), $\tau_1 \equiv \tau_x$, and $\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}, x : \tau_x; B; I \vdash e_{12} : \tau'$
By (T-EQ) rule, $\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit}_C \vdash e_2 : \tau_x$
By Limit-Change Lemma, $\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; I \vdash e_2 : \tau_x$
Using Substitution Lemma, we get $\Psi_e; \Phi_e; \Delta; \Gamma; B; I \vdash [x \mapsto v_2]e_{12} : \tau'$
Because $(I < \text{limit}_C)$ or $(\text{limit}_C = \infty$, $\Phi_e; \Delta \vdash \tau' \stackrel{\phi'}{:} 0)$,
by (T-COERCE) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \text{coerce}([x \mapsto v_2]e_{12}) : \tau'$
By (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e$; $\Phi'_e = \Phi_e$; $M' = M$

$(E-APPABS2)$
$e_1 = \lambda x : \tau_x \stackrel{\phi,\infty}{\longrightarrow} e_{12}$ $\quad e_2 = v_2$ $\quad e' = [x \mapsto v_2]e_{12}$
By Lemma 6.2.1.(3), $\tau_1 \equiv \tau_x$, and $\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}, x : \tau_x; B; \infty \vdash e_{12} : \tau'$
By (T-EQ) rule, $\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit}_C \vdash e_2 : \tau_x$
By Limit-Change Lemma, $\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \infty \vdash e_2 : \tau_x$
Using Substitution Lemma, we get $\Psi_e; \Phi_e; \Delta; \Gamma; B; \infty \vdash [x \mapsto v_2]e_{12} : \tau'$
Because $(\text{limit}_C = \text{limit}_f = \infty)$,
By (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M', e' : \tau)$

$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

5. $Case\,T-TAPP$: $e = e_1\tau_2 \qquad \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e_1\tau_2 : \tau$
By Lemma 6.2.1.(4), there is $[\alpha \mapsto \tau'_2]\,\tau'_1 \equiv \tau;\ \tau'_2 \equiv \tau_2,\ \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e_1\tau_2 : [\alpha \mapsto \tau'_2]\,\tau'_1$
and $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e_1 : \forall\alpha : K;B.\tau'_1$,
$\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \tau'_2 : K$ and $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash [\alpha \mapsto \tau'_2]B$

$(E-TAPPTABS)$
$e_1 = \Lambda\alpha : K';B.v, \qquad e' = [\alpha \mapsto \tau_2]v,$
Because $\tau'_2 \equiv \tau_2$, then $e' = [\alpha \mapsto \tau'_2]v$
By Lemma 6.2.1.(5), $K' = K, \quad \Psi_e;\Phi_e;\Delta,\alpha : K';\Gamma;B;\text{limit} \vdash v : \tau'_1$
Thus,$\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \tau'_2 : K'; \quad \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e_1 : \forall\alpha : K';B.\tau'_1$
By Term Substitution lemma, we get $[\alpha \mapsto \tau'_2](\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit}) \vdash [\alpha \mapsto \tau'_2]v : [\alpha \mapsto \tau'_2]\tau'_1$
Because α doesn't appear in $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit}$, and by (T-EQ) rule, $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e' : \tau$
Because $\Psi_{spare},\Psi_e;\Phi_{spare},\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing;\dot{\Phi};\varnothing;\varnothing;true;\infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare},\Psi'_e;\Phi_{spare},\Phi'_e;\Delta;\Gamma;B;\text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

6. $Case\,T-TUPLE$: $e = \phi\langle\overrightarrow{e}\rangle \qquad \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \phi\langle\overrightarrow{e}\rangle : \tau$
By Lemma 6.2.1.(6), $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \phi\langle\overrightarrow{e}\rangle : \phi\langle\overrightarrow{\tau'}\rangle$, and $\phi\langle\overrightarrow{\tau'}\rangle \equiv \tau$,
and $\forall i.(\Psi_{e_i};\Phi_{e_i};\Delta;\Gamma_{e_i};B;\text{limit} \vdash e_i : \tau_i) \quad \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \phi\langle\overrightarrow{\tau'}\rangle : K$
Suppose, $e_k \to e'_k$ now.
Because $\Psi_{spare},\Psi_e;\Phi_{spare},\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash (M, \phi\langle\overrightarrow{e}\rangle : \phi\langle\overrightarrow{\tau'}\rangle)$,
then $\forall i \in dom(\Psi).(\varnothing;\dot{\Phi};\varnothing;\varnothing;true;\infty \vdash M(i) : \Psi(i))$
and $\Psi_{e_k};\Phi_{e_k};\Delta;\Gamma_{e_k};B;\text{limit} \vdash e_k : \tau'_k$
Thus, $\Psi'_{spare},\Psi_{e_k};\Phi'_{spare},\Phi_{e_k};\Delta;\Gamma_{e_k};B;\text{limit} \vdash (M, e_k : \tau'_k)$
By induction, we obtain $\Psi'_{e_k};\Phi'_{e_k};\Delta;\Gamma_{e_k};B;\text{limit} \vdash e'_k : \tau'_k \quad (\Phi'_{e_k} \mathrel{\dot{\supseteq}} \Phi_{e_k})$
and $\Psi'_{spare},\Psi'_{e_k};\Phi'_{spare},\Phi'_{e_k};\Delta;\Gamma_{e_k};B;\text{limit} \vdash (M', e'_k : \tau'_k)$
(Here, $\forall i \in dom(\Psi').(\varnothing;\dot{\Phi}';\varnothing;\varnothing;true;\infty \vdash M'(i) : \Psi'(i))$))
and $\Psi'_{spare} = \Psi_{spare},\Psi_{e_1},\cdots\Psi_{e_{k-1}},\Psi_{e_{k+1}},\cdots\Psi_{e_n};\ \Phi'_{spare} = \Phi_{spare},\Phi_{e_1},\cdots\Phi_{e_{k-1}},\Phi_{e_{k+1}},\cdots\Phi_{e_n}$
For the other e_j,
$\Psi_{e_j};\Phi_{e_j};\Delta;\Gamma_{e_j};B;\text{limit} \vdash e_j : \tau'_j$

Given $C = \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit}$ and $C = C_1,...,C_n$ and $C'_k \mathrel{\dot{\supseteq}} C_k$, by the split subset lemma, we can find a $C'_1,\ldots,C'_n = \Psi'_e;\Phi'_e;\Delta;\Gamma;B;\text{limit} = C' \mathrel{\dot{\supseteq}} C$ so that for all $j \neq k$, $C'_j \supseteq C_j$. We then use weakening to show $C'_j \vdash e_j : \tau'_j$. Then using the (T-TUPLE) rule, we get:

$\Psi'_e;\Phi'_e;\Delta;\Gamma;B;\text{limit} \vdash \phi\langle\overrightarrow{e'}\rangle : \phi\langle\overrightarrow{\tau'}\rangle$
By (T-EQ), we obtain $\Psi'_e;\Phi'_e;\Delta;\Gamma;B;\text{limit} \vdash \phi\langle\overrightarrow{e'}\rangle : \tau$
Using (T-MEM) rule,
$\Psi_{spare},\Psi'_e;\Phi_{spare},\Phi'_e;\Delta;\Gamma;B;\text{limit} \vdash (M', \phi\langle\overrightarrow{e'}\rangle : \tau)$

7. $Case\,T-LET$: $e = \text{let}\,\langle\overrightarrow{x}\rangle = e_a\,\text{in}\,e_b$
$\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \phi\langle\overrightarrow{e}\rangle : \tau$
By Lemma 6.2.1.(7), there is $\tau'_b \equiv \tau. \quad \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \text{let}\,\langle\overrightarrow{x}\rangle = e_a\,\text{in}\,e_b : \tau'_b$
and $\Psi_{e_a};\Phi_{e_a};\Delta;\Gamma_{e_a};B;\text{limit} \vdash e_a : \phi\langle\overrightarrow{\tau'}\rangle, \quad \Psi_{e_b};\Phi_{e_b};\Delta;\Gamma_{e_b},\overrightarrow{x : \tau'};B;\text{limit} \vdash e_b : \tau'_b$

$(E-LET)$
$e_a = \phi\langle v_1, \cdots, v_n\rangle, \quad \Psi_{e_a}; \Phi_{e_a}; \Delta; \Gamma_{e_a}; B; \text{limit} \vdash \phi\langle v_1, \cdots, v_n\rangle : \phi\langle \tau_1', \cdots, \tau_n'\rangle,$
$e' = [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e_b$
By Lemma 6.2.1.(6), there is $\phi\langle \tau_1'', \cdots, \tau_n''\rangle \equiv \phi\langle \tau_1', \cdots, \tau_n'\rangle$
and $\forall i.(\Psi_{e_{a_i}}; \Phi_{e_{a_i}}; \Delta; \Gamma_{e_{a_i}}; B; \text{limit} \vdash v_i : \tau_i'')$
By (T-EQ), $\forall i.(\Psi_{e_{a_i}}; \Phi_{e_{a_i}}; \Delta; \Gamma_{e_{a_i}}; B; \text{limit} \vdash v_i : \tau_i')$
Because $\Psi_{e_b}; \Phi_{e_b}; \Delta; \Gamma_{e_b}, \overrightarrow{x : \tau'}; B; \text{limit} \vdash e_b : \tau_b'$
By substitution lemma, we obtain
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e_b : \tau_b'$
By (T-EQ), $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi_e'; \Phi_{spare}, \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi_e' = \Psi_e;\ \Phi_e' = \Phi_e;\ M' = M$

8. $Case\, T-LOAD$: $\quad e = \text{load}(e_{\text{ptr}}, e_{\text{Has}})$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(e_{\text{ptr}}, e_{\text{Has}}) : \tau$
By Lemma 6.2.1.(8), $^\wedge\langle \tau', \text{Has}(I, \tau')\rangle \equiv \tau$
and $\Psi_{e_{\text{ptr}}}; \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$
$\Psi_{e_{\text{Has}}}; \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$

$(E-LOAD)$
$e_{\text{ptr}} = i$, $e_{\text{Has}} = \text{fact}$, and $e' =^\wedge \langle M(i), \text{fact}\rangle$
$\varnothing; \dot{\Phi}_{e_{\text{ptr}}}; \Delta; \dot{\Gamma}_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$
$\{i \mapsto \tau'\}; \dot{\Phi}_{e_{\text{Has}}}; \Delta; \dot{\Gamma}_{e_{\text{Has}}}; B; \text{limit} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$
$\{i \mapsto \tau'\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e :^\wedge \langle \tau', \text{Has}(I, \tau')\rangle$
By (T-MEM) rule and Weakening Lemma,
$\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; \text{true}; \text{limit} \vdash M(i) : \tau'$
$\{i \mapsto \tau'\}; \dot{\Phi}_{e_{\text{Has}}}; \Delta; \dot{\Gamma}_{e_{\text{Has}}}; B; \text{limit} \vdash \text{fact} : \text{Has}(I, \tau')$
By (T-TUPLE) rule,
$\{i \mapsto \tau'\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash^\wedge \langle M(i), \text{fact}\rangle :^\wedge \langle \tau', \text{Has}(I, \tau')\rangle$
By (T-EQ) rule,
$\{i \mapsto \tau'\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi_e'; \Phi_{spare}, \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi_e' = \Psi_e;\ \Phi_e' = \Phi_e;\ M' = M$

$(E-LOAD1) \quad e_{\text{ptr}} \to e_{\text{ptr}}'$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \text{load}(e_{\text{ptr}}, e_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau')\rangle)$
then $\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M(i) : \Psi(i))$
Thus, by Lemma 6.2.1.(8) and (T-EQ) rule,
$\Psi_{e_{\text{ptr}}}; \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$
$\Rightarrow \Psi_{spare}', \Psi_{e_{\text{ptr}}}; \Phi_{spare}', \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M, e_{\text{ptr}} : \text{Int}(I))$
$(\Psi_{spare}' = \Psi_{spare}, \Psi_{e_{\text{Has}}};\ \Phi_{spare}' = \Phi_{spare}, \Phi_{e_{\text{Has}}})$
By induction, we obtain $\Psi_{e_{\text{ptr}}}'; \Phi_{e_{\text{ptr}}}'; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}}' : \text{Int}(I) \quad (\Phi_{e_{\text{ptr}}} \mathrel{\dot{\supseteq}} \Phi_{e_{\text{ptr}}})$
and $\Psi_{spare}', \Psi_{e_{\text{ptr}}}'; \Phi_{spare}', \dot{\Phi}_{e_{\text{ptr}}}'; \Delta; \dot{\Gamma}_{e_{\text{ptr}}}; B; \text{limit} \vdash (M', e_{\text{ptr}}' : \text{Int}(I))$

$\Rightarrow \Psi_{spare}, \Psi_{e_{\text{Has}}}, \Psi'_{e_{\text{ptr}}}; \Phi_{spare}, \Phi_{e_{\text{Has}}}, \Phi'_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M', e'_{\text{ptr}} : \text{Int}(I))$

(Here, $\forall i \in dom(\Psi').(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi'(i))))$

We still have $\Psi_{e_{\text{Has}}}; \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$

Given $C = \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}$ and $C = C_{ptr}, C_{Has}$ and $C'_{ptr} \dot{\supseteq} C_{ptr}$, by the split subset lemma, we can find a $C'_{ptr}, C'_{Has} = \Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} = C' \dot{\supseteq} C$ so that $C'_{Has} \supseteq C_{Has}$. We then use weakening to show $C'_{Has} \vdash e_{Has} : \text{Has}(I, \tau')$. Then using the (T-LOAD) rule, we have:

$\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(e'_{\text{ptr}}, e_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau') \rangle$

By (T-EQ), we obtain $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(e'_{\text{ptr}}, e_{\text{Has}}) : \tau$

BY (T-MEM) rule,

$\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', \text{load}(e'_{\text{ptr}}, e_{\text{Has}}) : \tau)$

$(E-LOAD2)$ $e_{\text{Has}} \rightarrow e'_{\text{Has}}$

Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \text{load}(v_{\text{ptr}}, e'_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau') \rangle)$

then $\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M(i) : \Psi(i))$

Thus, by Lemma 6.2.1.(8) and (T-EQ) rule,

$\Psi_{e_{\text{Has}}}; \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e_{e_{\text{Has}}} : \text{Has}(I, \tau')$

$\Rightarrow \Psi'_{spare}, \Psi_{e_{\text{Has}}}; \Phi'_{spare}, \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash (M, e_{\text{Has}} : \text{Has}(I, \tau'))$

$(\Psi'_{spare} = \Psi_{spare}, \Psi_{v_{\text{ptr}}};\ \Phi'_{spare} = \Phi_{spare}, \Phi_{v_{\text{ptr}}})$

By induction, we obtain $\Psi'_{e_{\text{Has}}}; \Phi'_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e'_{\text{Has}} : \text{Has}(I, \tau')$ $(\Phi_{e_{\text{Has}}} \dot{\supseteq} \Phi_{e_{\text{Has}}})$

and $\Psi'_{spare}, \Psi'_{e_{\text{Has}}}; \Phi'_{spare}, \Phi'_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash (M', e'_{\text{Has}} : \text{Has}(I, \tau'))$

$\Rightarrow \Psi_{spare}, \Psi_{v_{\text{ptr}}}, \Psi'_{e_{\text{Has}}}; \Phi_{spare}, \Phi_{v_{\text{ptr}}}, \Phi'_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash (M', e'_{\text{Has}} : \text{Has}(I, \tau'))$

(Here, $\forall i \in dom(\Psi').(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi'(i))))$

We still have $\Psi_{v_{\text{ptr}}}; \Phi_{v_{\text{ptr}}}; \Delta; \Gamma_{v_{\text{ptr}}}; B; \text{limit} \vdash v_{\text{ptr}} : \text{Int}(I)$

Given $C = \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}$ and $C = C_{ptr}, C_{Has}$ and $C'_{Has} \dot{\supseteq} C_{Has}$, by the split subset lemma, we can find a $C'_{ptr}, C'_{Has} = \Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} = C' \dot{\supseteq} C$ so that $C'_{ptr} \supseteq C_{ptr}$. We then use weakening to show $C'_{ptr} \vdash v_{ptr} : \text{Int}(I)$. Then using the (T-LOAD) rule, we have:

$\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(v_{\text{ptr}}, e'_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau') \rangle$

By (T-EQ), we obtain $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(v_{\text{ptr}}, e'_{\text{Has}}) : \tau$

BY (T-MEM) rule,

$\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', \text{load}(v_{\text{ptr}}, e'_{\text{Has}}) : \tau)$

9. $Case\,T-STORE$: $e = \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}})$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \tau$

By Lemma 6.2.1.(9), there is $\tau \equiv \text{Has}(I, \tau_2)$, and

$\Psi_{e_{\text{ptr}}}; \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$

$\Psi_{e_{\text{has}}}; \Phi_{e_{\text{has}}}; \Delta; \Gamma_{e_{\text{has}}}; B; \text{limit} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$

$\Psi_{e_{\text{v}}}; \Phi_{e_{\text{v}}}; \Delta; \Gamma_{e_{\text{v}}}; B; \text{limit} \vdash e_{\text{v}} : \tau_2,\ \Phi; \Delta \vdash \tau_2 \dot{:} 1$

$(E-STORE)$

$e = \text{store}(i, \text{fact}, v)$, and

$\varnothing; \dot{\Phi}_{e_{\text{ptr}}}; \Delta; \dot{\Gamma}_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$

$\{i \mapsto \tau_1\}; \dot{\Phi}_{e_{\text{has}}}; \Delta; \dot{\Gamma}_{e_{\text{has}}}; B; \text{limit} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$

$\varnothing; \dot{\Phi}_{e_{\text{v}}}; \Delta; \dot{\Gamma}_{e_{\text{v}}}; B; \text{limit} \vdash e_{\text{v}} : \tau_2,\ (\Phi; \Delta \vdash \tau_2 \dot{:} 1)$

$\{i \mapsto \tau_1\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \text{Has}(I, \tau_2)$

$e' = \text{fact}$, by (T-FACT1) rule,

$\{i \mapsto \tau_2\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Has}(I, \tau_2)$

$\{i \mapsto \tau_2\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau$

Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And $M' = [i \mapsto v]M$, $\Psi' = [i \mapsto \tau_2]\Psi$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi'(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = [i \mapsto \tau_2]\Psi_e;\ \Phi'_e = \Phi_e;\ M' = [i \mapsto v]M$

$(E - STORE1) \quad e_{\text{ptr}} \to e'_{\text{ptr}}$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \text{Has}(I, \tau_2))$
then $\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M(i) : \Psi(i))$
Thus, by Lemma 6.2.1.(9) and (T-EQ) rule,
$\Psi_{e_{\text{ptr}}}; \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$
$\Rightarrow \Psi'_{spare}, \Psi_{e_{\text{ptr}}}; \Phi'_{spare}, \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M, e_{\text{ptr}} : \text{Int}(I))$
$(\Psi'_{spare} = \Psi_{spare}, \Psi_{e_{\text{Has}}}, \Psi_{e_{\text{v}}};\ \Phi'_{spare} = \Phi_{spare}, \Phi_{e_{\text{Has}}}, \Phi_{e_{\text{v}}})$
By induction, we obtain $\Psi'_{e_{\text{ptr}}}; \Phi'_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e'_{\text{ptr}} : \text{Int}(I) \quad (\Phi_{e_{\text{ptr}}} \mathrel{\dot{\supseteq}} \Phi_{e_{\text{ptr}}})$
and $\Psi'_{spare}, \Psi'_{e_{\text{ptr}}}; \Phi'_{spare}, \Phi'_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M', e'_{\text{ptr}} : \text{Int}(I))$
$\Rightarrow \Psi_{spare}, \Psi_{e_{\text{Has}}}, \Psi'_{e_{\text{ptr}}}, \Psi_{e_{\text{v}}}; \Phi_{spare}, \Phi_{e_{\text{Has}}}, \Phi'_{e_{\text{ptr}}}, \Phi_{e_{\text{v}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M', e'_{\text{ptr}} : \text{Int}(I))$
(Here, $\forall i \in dom(\Psi').(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi'(i)))$)
We still have $\Psi_{e_{\text{has}}}; \Phi_{e_{\text{has}}}; \Delta; \Gamma_{e_{\text{has}}}; B; \text{limit} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$
$\Psi_{e_{\text{v}}}; \Phi_{e_{\text{v}}}; \Delta; \Gamma_{e_{\text{v}}}; B; \text{limit} \vdash e_{\text{v}} : \tau_2$

Given $C = \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}$ and $C = C_{ptr}, C_{Has}, C_v$ and $C'_{ptr} \mathrel{\dot{\supseteq}} C_{ptr}$, by the split subset lemma, we can find a $C'_{ptr}, C'_{Has}, C'_v = \Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} = C' \mathrel{\dot{\supseteq}} C$ so that $C'_{Has} \supseteq C_{Has}$ and $C'_v \supseteq C_v$. We then use weakening to show $C'_{Has} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$ and $C'_v \vdash e_v : \tau_2$. Then using the (T-STORE) rule, we have:
$\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \text{Has}(I, \tau_2)$
By (T-EQ), we obtain $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \tau$
BY (T-MEM) rule,
$\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \tau)$

10. $Case\,T - FIX: \quad e = \text{fix}\, x : \tau_x.v$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \text{fix}\, x : \tau_x.v : \tau$
By Lemma 6.2.1.(10), there is $\tau_x \equiv \tau$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \text{fix}\, x : \tau_x.v : \tau$
$\Psi_e; \Phi_e; \Delta; \Gamma, x : \tau_x; B; \text{limit}_C \vdash v : \tau$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \tau \overset{\phi}{:} i$

$(E - FIX)$
$e' = [x \mapsto \text{fix}\, x : \tau_x.v]v$
and $\Psi_e; \Phi_e; \Delta; \Gamma, x : \tau_x; B; \text{limit}_C \vdash v : \tau$
By substitution lemma, we obtain
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash [x \mapsto \text{fix}\, x : \tau_x.v]v : \tau$
Thus $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash e' : \tau$
We don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

11. $Case\,T - UNROLL: \quad e = \text{unroll}(e_0) \quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{unroll}(e_0) : \tau$
By Lemma 6.2.1.(11), there is $([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n \equiv \tau$, $\tau' = (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n$

and $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \tau' : K$, $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e_0 : \tau'$

$(E-UNROLL)$
$e_0 = \text{roll}[\tau''](v) \ \ e' = v$
$\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \text{roll}[\tau''](v) : \tau'$
By Lemma 6.2.1.(12), we know $\tau'' \equiv \tau' \equiv (\mu\beta : K.\tau_{roll})\tau_1 \cdots \tau_n$, (so $\tau_0 \equiv \tau_{roll}$)
and $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \tau_{roll} : K \ \ \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash v : ([\beta \mapsto \mu\beta : K.\tau_{roll}]\tau_{roll})\tau_1 \cdots \tau_n$
By (T-EQ) rule, $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$
By (T-EQ) rule again, we have $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e' : \tau$
Because $\Psi_{spare},\Psi_e;\Phi_{spare},\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing;\dot{\Phi}';\varnothing;\varnothing;true;\infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare},\Psi'_e;\Phi_{spare},\Phi'_e;\Delta;\Gamma;B;\text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

12. $Case\,T-UNPACK$: $\ \ e = \text{unpack}\,\alpha, x = e_1 \,\text{in}\, e_2$
$\Psi_e;\Phi_e;\Delta;\Gamma;B_e;\text{limit} \vdash \text{unpack}\,\alpha, x = e_1 \,\text{in}\, e_2 : \tau$
By Lemma 6.2.1.(13), there is $\tau_2 \equiv \tau$,
$\Psi_{e_1};\Phi_{e_1};\Delta;\Gamma_{e_1};B;\text{limit} \vdash e_1 : \exists\alpha : K;B'.\tau_1$
$\Psi_{e_2};\Phi_{e_2};\Delta,\alpha : K;\Gamma_{e_2},x : \tau_1;B,B';\text{limit} \vdash e_2 : \tau_2$

$(E-UNPACK)$
$e_1 = \text{pack}[\tau_0, v]\,\text{as}\,\exists\beta : K';\tau_B.\tau'_1 \quad e' = [\alpha \mapsto \tau_0, x \mapsto v]e_2$
$\Psi_{e_1};\Phi_{e_1};\Delta;\Gamma_{e_1};B;\text{limit} \vdash \text{pack}[\tau_0, v]\,\text{as}\,\exists\beta : K';\tau_B.\tau'_1 : \exists\alpha : K;B'.\tau_1$
By Lemma 6.2.1.(14), then $\tau'_1 \equiv \tau_1$, $\ \tau_B \equiv B'$ and, $K' \equiv K$,
$\exists\beta : K';\tau_B.\tau'_1 \equiv \exists\alpha : K;B'.\tau_1$
$\Psi_{e_1};\Phi_{e_1};\Delta;\Gamma_{e_1};B;\text{limit} \vdash \tau_0 : K \quad B \vdash [\alpha \mapsto \tau_0]\tau_B$
$\Psi_{e_1};\Phi_{e_1};\Delta;\Gamma_{e_1};B;\text{limit} \vdash v : [\alpha \mapsto \tau_0]\tau'_1$
By (T-EQ), $\Psi_{e_1};\Phi_{e_1};\Delta;\Gamma_{e_1};B;\text{limit} \vdash v : [\alpha \mapsto \tau_0]\tau_1$
Because $\Psi_{e_2};\Phi_{e_2};\Delta,\alpha : K;\Gamma_{e_2},x : \tau_1;B,B';\text{limit} \vdash e_2 : \tau_2$,
by Term Substitution Lemma for α,
$[\alpha \mapsto \tau_0](\Psi_{e_2};\Phi_{e_2};\Delta;\Gamma_{e_2},x : \tau_1;B,B';\text{limit}) \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$
$\Rightarrow \Psi_{e_2};\Phi_{e_2};\Delta;\Gamma_{e_2};B,[\alpha \mapsto \tau_0]B';\text{limit},x : [\alpha \mapsto \tau_0]\tau_1 \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$
$\Rightarrow \Psi_{e_2};\Phi_{e_2};\Delta;\Gamma_{e_2};B;\text{limit},x : [\alpha \mapsto \tau_0]\tau_1 \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$
by Term Substitution Lemma for v,
$\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash [\alpha \mapsto \tau_0, x \mapsto v]e_2 : [\alpha \mapsto \tau_0]\tau_2$
$\Rightarrow \Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash [\alpha \mapsto \tau_0, x \mapsto v]e_2 : \tau_2$
Thus $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e' : \tau_2$
By (T-EQ) tule, $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash e' : \tau$
Because $\Psi_{spare},\Psi_e;\Phi_{spare},\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing;\dot{\Phi};\varnothing;\varnothing;true;\infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare},\Psi'_e;\Phi_{spare},\Phi'_e;\Delta;\Gamma;B;\text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

13. $Case\,T-DISTINGUISH$: $\ \ e = \text{distinguish}(I_1, I_2, e_1, e_2)$
$\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash \text{distinguish}(I_1, I_2, e_1, e_2) : \tau$
By Lemma 6.2.1.(15), we get $\wedge\langle \text{Know}(I_1 \neq I_2), \text{Has}(I_1, \tau_1), \text{Has}(I_2, \tau_2)\rangle \equiv \tau$,
and $\Psi_{e_1};\Phi_{e_1};\Delta;\Gamma_{e_1};B;\text{limit} \vdash e_1 : \text{Has}(I_1, \tau_1)$
$\Psi_{e_2};\Phi_{e_2};\Delta;\Gamma_{e_2};B;\text{limit} \vdash e_2 : \text{Has}(I_2, \tau_2)$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash I_1 : \text{int} \quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash I_2 : \text{int}$

$(E-DISTINGUISH)$
Then $e_1 = fact_1,\ \ e_2 = fact_2,\ e' =^{\wedge} \langle \text{know}(I_1 \neq I_2), fact_1, fact_2 \rangle$
From definition, $\text{know}(I_1 \neq I_2) = \text{pack}[\text{true}, \cdot\langle\rangle]\, \text{as}\, \exists\alpha : \text{bool}; I_1 \neq I_2. \cdot \langle\rangle$
Because by (K-TUPLE) $\Phi_e; \Delta \vdash \text{true} : \text{bool}$,
and by (K-SOME) $\Phi_e; \Delta \vdash \exists\alpha : \text{bool}; I_1 \neq I_2. \cdot \langle\rangle : \dot{0}$
by (T-TUPLE) $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \cdot\langle\rangle : \cdot\langle\rangle$
$\Rightarrow \varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \cdot\langle\rangle : [\alpha \mapsto \cdot\langle\rangle] \cdot \langle\rangle$,
and $\Phi_e; \Delta \vdash [\alpha \mapsto \cdot\langle\rangle](I_1 \neq I_2)$
By (T-PACK) rule,
$\varnothing; \Phi_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{pack}[\text{true}, \cdot\langle\rangle]\, \text{as}\, \exists\alpha : \text{bool}; I_1 \neq I_2. \cdot \langle\rangle : \exists\alpha : \text{bool}; I_1 \neq I_2. \cdot \langle\rangle$
Using abbreviations, $\varnothing; \Phi_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{know}(I_1 \neq I_2) : \text{Know}(I_1 \neq I_2)$
and $\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash e_1 : \text{Has}(I_1, \tau_1)$
$\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit} \vdash e_2 : \text{Has}(I_2, \tau_2)$
By (T-TUPLE) rule , we obtain:
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash^{\wedge} \langle \text{know}(I_1 \neq I_2), fact_1, fact_2 \rangle :^{\wedge} \langle \text{Know}(I_1 \neq I_2), \text{Has}(I_1, \tau_1), \text{Has}(I_2, \tau_2) \rangle$
Thus, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \ \Phi'_e = \Phi_e;\ \ M' = M$

14. $Case\, T-NEWFUN: \quad e = \text{new\_fun}(J)$
$\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{new\_fun}(J) : \tau$
By Lemma 6.2.1.(16), then $\exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0) \equiv \tau$
and $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{new\_fun}(J) : \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)$

$(E-NEWFUN)$
$e' = \text{pack}[F^J, \text{fact}]\, \text{as}\, \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)$
By (T-FACT2) rule, we have $\varnothing; \dot{\Phi}_e, F^J \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Gen}(F^J, 0)$
By (K-FUN) rule, $\Phi_e, F^J \dot{\mapsto} \text{fun}; \Delta \vdash F^J : \text{int} \to J$
Because $\Phi_e; \Delta \vdash 0 : \text{int}$, and $\Phi_e, F^J \dot{\mapsto} \text{fun}; \Delta, \alpha : \text{int} \to J \vdash \alpha : \text{int} \to J$
By (K-FUNGEN) rule, $\Phi_e, F^J \dot{\mapsto} \text{fun}; \Delta \vdash \text{Gen}(\alpha, 0) : \hat{0}$
By (K-SOME) rule, we obtain $\Phi_e, F^J \dot{\mapsto} \text{fun}; \Delta \vdash \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0) : \hat{0}$
By (T-PACK) rule, we get
$\varnothing; \Phi_e, F^J \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{pack}[F^J, \text{fact}]\, \text{as}\, \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0) : \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)$
Thus, $\varnothing; \Phi_e, F^J \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)$
By (T-EQ) rule, $\varnothing; \Phi_e, F^J \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
We don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Now $\Phi'_e \dot{\supseteq} \Phi_e$, $\Psi'_e = \Psi_e$; from (T-NEWFUN) we know F^J is fresh.
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$

15. $Case\, T - DEFINEFUN$: $\quad e = \text{define\_fun}(e_0, \tau_a)$.
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{define\_fun}(e_0, \tau_a) : \tau$
By Lemma 6.2.1.(17), then $^\wedge\langle \text{Gen}(\tau_f, I+1), \text{Eq}(\tau_f I, \tau_a), \text{InDomain}(I, \tau_f)\rangle \equiv \tau$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Gen}(\tau_f, I)$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_f : \text{int} \to J$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_a : J$

$(E - DEFINEFUN)$
$e_0 = \text{fact}$, then $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e_0 : \text{Gen}(\tau_f, I)$
and $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{define\_fun}(e_0, \tau_a) : \tau$
$\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \tau_f : \text{int} \to J$
$\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \tau_a : J$
Now, $e' =^\wedge \langle \text{fact}, \text{fact}, \text{fact}\rangle$, $(\text{fun}' \supseteq \text{fun})$
By (T-FACT2) rule, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Gen}(\tau_f, I+1)$
By (T-FACT3) rule, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\cdot}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_f I, \tau_a)$
By weakening lemma, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\cdot}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_f I, \tau_a)$
By (T-FACT4) rule, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\cdot}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{InDomain}(I, \tau_f)$
By weakening lemma, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\cdot}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{InDomain}(I, \tau_f)$
By (T-TUPLE) rule, we obtain $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash$
$^\wedge\langle \text{fact}, \text{fact}, \text{fact}\rangle :^\wedge \langle \text{Gen}(\tau_f, I+1), \text{Eq}(\tau_f I, \tau_a), \text{InDomain}(I, \tau_f)\rangle$
By (T-EQ) rule, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Now $\Phi'_e \dot{\supseteq} \Phi_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}$; $\Psi'_e = \Psi_e$;
By (T-MEM), $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$

16. $Case\, T - DISCARDFUN$: $\quad e = \text{discard\_fun}(e_0)$.
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{discard\_fun}(e_0) : \tau'$
By Lemma 6.2.1.(18), then $\tau' \equiv \cdot\langle\rangle$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Gen}(\tau, I)$

$(E - DISCARD)$
$e_0 = \text{fact}$, then $\varnothing; \dot{\Phi}_{e_0}, \tau \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e_0 : \text{Gen}(\tau, I)$
$\varnothing; \dot{\Phi}_{e_0}, \tau \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{define\_fun}(e_0, \tau_a) : \tau$
Now $e' = \cdot\langle\rangle$, and by (T-TUPLE) rule $\dot{C} \vdash \cdot\langle\rangle : \cdot\langle\rangle$
Thus, $\varnothing; \dot{\Phi}_{e_0}, \tau \overset{\cdot}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \cdot\langle\rangle : \cdot\langle\rangle$
By (T-EQ) rule, we obtain $\varnothing; \dot{\Phi}_{e_0}, \tau \overset{\cdot}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \cdot\langle\rangle : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
We don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Now $\Phi'_e \dot{\supseteq} \Phi_e$, $\Psi'_e = \Psi_e$;
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$

17. $Case\, T - INDOMAIN$: $\quad e = \text{in\_domain}(I_1, I_2, e_1, e_2)$

$\Psi_e; \Phi_e; \Delta; \Gamma_e; B; \text{limit} \vdash \text{in\_domain}(I_1, I_2, e_1, e_2) : \tau$
By Lemma 6.2.1.(19), then $^\wedge\langle \text{Know}(0 \le I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle \equiv \tau$
and $\Psi_e; \Phi_e; \Delta; \Gamma_e; B; \text{limit} \vdash I_1 : \text{int}, \quad \Psi_e; \Phi_e; \Delta; \Gamma_e; B; \text{limit} \vdash I_2 : \text{int}$
$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash e_1 : \text{InDomain}(I_1, \tau_f)$
$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash e_2 : \text{Gen}(\tau_f, I_2)$

$(E - INDOMAIN)$
$e_1 = \text{fact}_1$, $e_2 = \text{fact}_2$ then
$\varnothing; \dot{\Phi}_{e_1}, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_{e_1}; B; \text{limit} \vdash e_1 : \text{InDomain}(I_1, \tau_f)$
$\varnothing; \dot{\Phi}_{e_2}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_{e_2}; B; \text{limit} \vdash e_2 : \text{Gen}(\tau_f, I_2)$
$\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_{e}; B; \text{limit} \vdash e : \tau$
$e' =^\wedge \langle \text{know}(0 \le I_1 \wedge I_1 < I_2), \text{fact}\rangle$
By (T-FACT2) rule, we get $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{fact} : \text{Gen}(\tau_f, I_2)$
From definition, we $\text{know}(0 \le I_1 \wedge I_1 < I_2) = \text{pack}[\text{true}, \cdot\langle\rangle]\,\text{as}\,\exists\alpha : \text{bool}; 0 \le I_1 \wedge I_1 < I_2. \cdot \langle\rangle$
Because by (K-BOOL) $\dot{\Phi}_{e_0}, \tau_f \dot{\mapsto} \text{fun}; \Delta \vdash \text{true} : \text{bool}$,
and by (K-SOME) $\dot{\Phi}_{e_0}, \tau_f \dot{\mapsto} \text{fun}; \Delta \vdash \exists\alpha : \text{bool}; 0 \le I_1 \wedge I_1 < I_2. \cdot \langle\rangle :\dot{0}$
$\varnothing; \dot{\Phi}_{e_0}, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \cdot\langle\rangle : [\alpha \mapsto \text{true}] \cdot \langle\rangle$,
and $B \vdash [\alpha \mapsto \text{true}](0 \le I_1 \wedge I_1 < I_2)$
By (T-PACK) rule, we obtain:
$\varnothing; \dot{\Phi}_{e_0}, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{pack}[\text{true}, \cdot\langle\rangle]\,\text{as}\,\exists\alpha : \text{bool}; 0 \le I_1 \wedge I_1 < I_2. \cdot \langle\rangle : \exists\alpha : \text{bool}; 0 \le I_1 \wedge I_1 < I_2. \cdot \langle\rangle$
By using abbreviations, $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{know}(0 \le I_1 \wedge I_1 < I_2) : \text{Know}(0 \le I_1 \wedge I_1 < I_2)$

By (T-TUPLE) rule, we obtain:
$\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash^\wedge \langle \text{know}(0 \le I_1 \wedge I_1 < I_2), \text{fact}\rangle :^\wedge \langle \text{Know}(0 \le I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle$
Then, $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e' :^\wedge \langle \text{Know}(0 \le I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle$
By (T-EQ) rule, $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
Here we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

18. $Case\, T - MAKEEQ$: $\quad e = \text{make\_eq}(\tau_0)$
$\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{make\_eq}(\tau_0) : \tau$
By Lemma 6.2.1.(20), then $\text{Eq}(\tau_0, \tau_0) \equiv \tau$ and $\Phi_e; \Delta \vdash \tau_0 : K$

$(E - MAKEEQ)$
$e' = \text{fact}$
By (T-FACT3) rule, $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_0, \tau_0)$
Then $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e' : \text{Eq}(\tau_0, \tau_0)$
Thus, $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash (M', e' : \tau)$

$\Psi_e' = \Psi_e;\ \Phi_e' = \Phi_e;\ M' = M$

19. $Case\, T - APPLYEQ$: $\quad e = \text{apply\_eq}(\tau_f, e_1, e_2)$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau$
By Lemma 6.2.1.(21), then $\tau_f \tau_b \equiv \tau$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau_f \tau_b$
$\Phi_e; \Delta \vdash \tau_f : K \to J \quad \Phi_e; \Delta \vdash \tau_a : K \quad \Phi_e; \Delta \vdash \tau_b : K$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_1 : \text{Eq}(\tau_a, \tau_b) \quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_2 : \tau_f \tau_a$

$(E - APPLYEQ)$
$e_1 = \text{fact},\ e_2 = \text{v}$, and $e' = \text{v}$
By (T-FACT3) rule, $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e_1 : \text{Eq}(\tau_{e_1}, \tau_{e_1})$,
then $\tau_{e_1} \equiv \tau_a \equiv \tau_b$, and $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e : \tau_f \tau_b$,
Thus, $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e_2 : \tau_f \tau_a \Rightarrow \varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{v} : \tau_f \tau_b$
$\Rightarrow \varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau_f \tau_b$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
By (T-MEM) rule, $\Psi_{spare}, \Psi_e'; \Phi_{spare}, \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi_e' = \Psi_e;\ \Phi_e' = \Phi_e;\ M' = M$

20. $Case\, T - CASE$: $\quad e = \text{case}(b, e_0)$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{case}(b, e_0) : \tau$
By Lemma 6.2.1.(22), then $\tau \equiv \tau_i \quad (i = \begin{array}{c} 1\, if\, b \\ 2\, if\, \neg b \end{array})$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Union}(b, \tau_1, \tau_2)$

$(E - CASE)$
$e_0 = \text{union}(\tau_b, \tau_1', \tau_2', v), \quad e' = v$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{union}(\tau_b, \tau_1', \tau_2', v) : \text{Union}(b, \tau_1, \tau_2)$
By Lemma 6.2.1.(23), then $\tau_1 \equiv \tau_1'$, $\tau_2 \equiv \tau_2'$, $\tau_b \equiv b$ and
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_1' : K$, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_2' : K$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash v : \tau_i' \quad (i = \begin{array}{c} 1\, if\, \tau_b \\ 2\, if\, \neg \tau_b \end{array})$
Thus, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau_i' \quad (i = \begin{array}{c} 1\, if\, b \\ 2\, if\, \neg b \end{array})$
By (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
By (T-MEM) rule, $\Psi_{spare}, \Psi_e'; \Phi_{spare}, \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi_e' = \Psi_e;\ \Phi_e' = \Phi_e;\ M' = M$

21. $Case\, T - COERCE$: $\quad e = \text{coerce}(e_0)$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e_0) : \tau$
By Lemma 6.2.1.(24), then there is $\tau' \equiv \tau$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e_0) : \tau'$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; I_2 \vdash e_0 : \tau'$

$(E-COERCE)$
$e_0 = v$, $e' = v$
Because $\Psi_e;\Phi_e;\Delta;\Gamma;B;I_2 \vdash e_0 : \tau'$, then $\Psi_e;\Phi_e;\Delta;\Gamma;B;I_2 \vdash v : \tau'$
By Limit-Change Lemma, $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit}_1 \vdash v : \tau'$
By (T-EQ) rule, $\Psi_e;\Phi_e;\Delta;\Gamma;B;\text{limit}_1 \vdash e' : \tau$
Because $\Psi_{spare},\Psi_e;\Phi_{spare},\Phi_e;\Delta;\Gamma;B;\text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing;\dot{\Phi}';\varnothing;\varnothing;true;\infty \vdash M'(i) : \Psi(i))$
By (T-MEM) rule, $\Psi_{spare},\Psi_e';\Phi_{spare},\Phi_e';\Delta;\Gamma;B;\text{limit} \vdash (M', e' : \tau)$
$\Psi_e' = \Psi_e;\ \Phi_e' = \Phi_e;\ M' = M$

22. All cases under the congruence evaluation lemma are similar.
We already proved some cases in (TUPLE), (LOAD), and (STORE).

6.3 Progress

Rephrase the type equivalence rules as parallel reduction rules:

$$(A) \qquad \Phi;B \vdash \tau \Rightarrow \tau$$

$$(B) \qquad \frac{\Phi;B \vdash \tau_a \Rightarrow \tau_a' \qquad \Phi;B \vdash \tau_b \Rightarrow \tau_b'}{\Phi;B \vdash (\lambda\alpha : K.\tau_b)\tau_a \Rightarrow [\alpha \mapsto \tau_a']\tau_b'}$$

$$(C) \qquad \frac{B \vdash I \doteq i}{\Phi, \mathrm{F}^K \dot{\mapsto} \delta; B \vdash \mathrm{F}^K I \Rightarrow \delta(i)}$$

$$(D) \qquad \frac{B \vdash I_1 \doteq I_2}{\Phi;B \vdash I_1 \Rightarrow I_2}$$

$$(E) \qquad \frac{B \vdash B_1 \doteq B_2}{\Phi;B \vdash B_1 \Rightarrow B_2}$$

$$(F) \qquad \Phi;B \vdash \mathrm{Eq}(\tau_1,\tau_2) \Rightarrow \mathrm{Eq}(\tau_2,\tau_1)$$

$$(G) \qquad \frac{\forall i.(\Phi;B \vdash \tau_i \Rightarrow \tau_i')}{\Phi;B \vdash T[\tau_1,\ldots,\tau_n] \Rightarrow T[\tau_1',\ldots,\tau_n']}$$

6.3.1 LEMMA [SINGLE-PARALLEL-STEP CONFLUENCE FOR TYPES]

If $C \vdash \tau_a \Rightarrow \tau_b$, and $C \vdash \tau_a \Rightarrow \tau_c$, then there is some τ_d so that $C \vdash \tau_b \Rightarrow \tau_d$ and $C \vdash \tau_c \Rightarrow \tau_d$.
Proof by induction on the sum of the sizes of the derivations of $\tau_a \Rightarrow \tau_b$ and $\tau_a \Rightarrow \tau_c$.

1. case $\tau_a \overset{A}{\Rightarrow} \tau_b$. Then $\tau_b = \tau_a$, so choose $\tau_d = \tau_c$.
2. case $\tau_a \overset{B}{\Rightarrow} \tau_b$ and $\tau_a \overset{B}{\Rightarrow} \tau_c$. The proof for this is standard.
3. case $\tau_a \overset{G}{\Rightarrow} \tau_b$ and $\tau_a \overset{B}{\Rightarrow} \tau_c$. The proof for this is standard.
4. case $\tau_a \overset{G}{\Rightarrow} \tau_b$ and $\tau_a \overset{G}{\Rightarrow} \tau_c$. The proof for this is standard.
5. case $\tau_a = \mathrm{Eq}(\tau_{a1},\tau_{a2}) \overset{F}{\Rightarrow} \tau_b = \mathrm{Eq}(\tau_{a2},\tau_{a1})$. If $\tau_a \overset{F}{\Rightarrow} \tau_c$, then choose $\tau_d = \tau_b = \tau_c$. If $\tau_a = \mathrm{Eq}(\tau_{a1},\tau_{a2}) \overset{G}{\Rightarrow} \tau_c = \mathrm{Eq}(\tau_{c1},\tau_{c2})$, then choose $\tau_d = \mathrm{Eq}(\tau_{c2},\tau_{c1})$.

6. case $\tau_a = I_a \overset{D}{\Rightarrow} \tau_b = I_b$ and $\tau_a = I_a \overset{D}{\Rightarrow} \tau_c = I_c$. Then $I_a \doteq I_b \doteq I_c$, so pick $\tau_d = I_a$.

7. case $\tau_a = B_a \overset{E}{\Rightarrow} \tau_b = B_b$ and $\tau_a = B_a \overset{E}{\Rightarrow} \tau_c = B_c$. Then $B_a \doteq B_b \doteq B_c$, so pick $\tau_d = B_a$.

8. case $\tau_a = \mathrm{F}^K I_a \overset{C}{\Rightarrow} \tau_b = \delta(i_b)$. If $\tau_a \overset{C}{\Rightarrow} \tau_c = \delta(i_c)$, then $i_b = i_c$, so choose $\tau_d = \tau_b = \tau_c$. If $\tau_a = \mathrm{F}^K I_a \overset{G}{\Rightarrow} \tau_c = \mathrm{F}^K I_c$, then $i_b \doteq I_a \doteq I_c$, so choose $\tau_d = \tau_b$.

The other cases are either impossible, or are symmetric to one of the cases above.

6.3.2 LEMMA [CONFLUENCE FOR TYPES]

If $C \vdash \tau_a \overset{*}{\Rightarrow} \tau_b$, and $C \vdash \tau_a \overset{*}{\Rightarrow} \tau_c$, then there is some τ_d so that $C \vdash \tau_b \overset{*}{\Rightarrow} \tau_d$ and $C \vdash \tau_c \overset{*}{\Rightarrow} \tau_d$. Standard proof by tiling single parallel steps to fill in the area between $\tau_a \overset{*}{\Rightarrow} \tau_b$ and $\tau_a \overset{*}{\Rightarrow} \tau_c$.

6.3.3 CORROLARY [CONFLUENCE FOR EQUIVALENT TYPES]

If $C \vdash \tau_b \equiv \tau_c$, then there is some τ_d so that $C \vdash \tau_b \overset{*}{\Rightarrow} \tau_d$ and $C \vdash \tau_c \overset{*}{\Rightarrow} \tau_d$. Proof by induction on the derivation of $C \vdash \tau_b \equiv \tau_c$.

6.3.4 LEMMA [SHAPE PRESERVATION]

To prove progress, we must first say something about how values are typed. These are the values:

$$v = i \,|\, b \,|\, \Lambda\alpha : K; B.v \,|\, \mathrm{pack}[\tau_1, v]\,\mathrm{as}\,\exists\alpha : K; B.\tau_2$$

$$|\, \mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) \,|\, \lambda x : \tau \overset{\phi,\mathrm{limit}}{\longrightarrow} e \,|\, \phi\langle\overrightarrow{v}\rangle \,|\, \mathrm{union}(b, \tau_1, \tau_2, v) \,|\, \mathrm{fact}$$

For these values, there are type checking rules that give the values the following types:

$$\tau = \tau_1 \overset{\phi,limit}{\longrightarrow} \tau_2 \,|\, \langle\overrightarrow{\tau}\rangle \,|\, \forall\alpha : K; B.\tau$$

$$|\, \exists\alpha : K; B.\tau \,|\, (\mu\alpha : K.\tau)\tau_1 \cdots \tau_n \,|\, \mathrm{Int}(I) \,|\, \mathrm{Bool}(B) \,|\, \mathrm{Union}(B, \tau_1, \tau_2)$$

$$|\, \mathrm{Has}(I, \tau) \,|\, \mathrm{Gen}(\tau, I) \,|\, \mathrm{Eq}(\tau_1, \tau_2) \,|\, \mathrm{InDomain}(I, \tau)$$

In addition, there is the (T-EQ) rule, which can also give a value a type. Call the 12 different categories of types listed above the 12 *value shapes*. If $\tau \Rightarrow \tau'$, and τ has a value shape, then τ' has the same value shape (proof by case analysis on $\tau \Rightarrow \tau'$). Combining this with confluence, we conclude that if $C \vdash \tau_b \equiv \tau_c$, and τ_b and τ_c have value shapes, there is some τ_d so that $C \vdash \tau_b \overset{*}{\Rightarrow} \tau_d$ and $C \vdash \tau_c \overset{*}{\Rightarrow} \tau_d$, and τ_d has the same shape as τ_b and τ_c, which implies that τ_b and τ_c have the same shape.

6.3.5 LEMMA [CANONICAL FORMS]

Suppose v is a closed, well-typed expression: $C \vdash v : \tau$ for some τ, $C = \Psi; \Phi; \varnothing; \varnothing; B; \mathrm{limit}$.

1. If $C \vdash v : \tau_1 \overset{\phi,limit}{\longrightarrow} \tau_2$, then $v = \lambda x : \tau_1 \overset{\phi,limit}{\longrightarrow} e$.
2. If $C \vdash v : \langle\tau_1, \ldots \tau_n\rangle$, then $v = \langle v_1, \ldots, v_n\rangle$.
3. If $C \vdash v : \forall\alpha : K; B.\tau$, then $v = \Lambda\alpha : K; B.v_0$.
4. If $C \vdash v : \exists\alpha : K; B.\tau_2$, then $v = \mathrm{pack}[\tau_1, v_0]\,\mathrm{as}\,\exists\alpha : K; B.\tau_2$.
5. If $C \vdash v : (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n$, then $v = \mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v_0)$.
6. If $C \vdash v : \mathrm{Int}(I)$, then $v = i$.

7. If $C \vdash v : \mathrm{Bool}(B)$, then $v = b$.

8. If $C \vdash v : \mathrm{Union}(b, \tau_1, \tau_2)$, then $v = \mathrm{union}(b, \tau_1, \tau_2, v_0)$.

9. If $C \vdash v : \mathrm{Has}(I, \tau)$ or $C \vdash v : \mathrm{Gen}(\tau, I)$ or $C \vdash v : \mathrm{Eq}(\tau_1, \tau_2)$ or $C \vdash v : \mathrm{InDomain}(I, \tau)$, then $v = \mathrm{fact}$.

Proof:

1. Only two type checking rules can prove $C \vdash v : \tau_1 \xrightarrow{\phi, limit} \tau_2$: (T-EQ) and (T-ABS).
For (T-EQ) rule, there is some $\tau' \equiv \tau_1 \xrightarrow{\phi, limit} \tau_2$. By shape preservation, $\tau' = \tau'_1 \xrightarrow{\phi, limit} \tau'_2$. Use induction.
By (T-ABS) rule, we know $v = \lambda x : \tau_1 \xrightarrow{\phi, limit} e$ is an abstraction.

2. The similar proof for the other cases.

6.3.6 THEOREM[PROGRESS]

If (M, e) is closed and well-typed ($C_{Me} \vdash (M, e : \tau)$ for some τ and $C_{Me} = \Psi_{Me}; \Phi_{Me}; \varnothing; \varnothing; B; \mathrm{limit}$), then either e is a value or else there is some (M', e') so that $(M, e) \to (M', e')$.

Observe that by the rule for typing (M, e), we know that $\Psi_{Me} = \Psi_{spare}, \Psi$ and $\Phi_{Me} = \Phi_{spare}, \Phi$ and $C \vdash e : \tau$, where $C = \Psi; \Phi; \varnothing; \varnothing; B; \mathrm{limit}$. Now prove that (M, e) steps, by induction on the derivation of $C \vdash e : \tau$ (holding M, Ψ_{Me}, and Φ_{Me} fixed throughout the induction as C, e, and τ vary).

Proof :

First, if $e = E[e_0]$ where e_0 is not a value, then by inspection of the type checking rules, $C_0 \vdash e_0 : \tau_0$, where $C_0 = \Psi_0; \Phi_0; \varnothing; \varnothing; B; \mathrm{limit}_0$ and $\Psi_{Me} = \Psi_{0-spare}, \Psi_0$ $\Phi_{Me} = \Phi_{0-spare}, \Phi_0$, so by induction, $(M, e_0) \to (M', e'_0)$, so $(M, e) = (M, E[e_0]) \to (M', E[e'_0])$. Second, the (T-EQ) case is an easy induction. For all other cases where e is not a value:

1. $e = x$
It will not happen, because e is closed.

2. $e = v_1\, v_2$, where $(T-APP)\dfrac{\begin{array}{c} C_1, C_2 = \Psi; \Phi; \Delta; \Gamma; B; \mathrm{limit}_C \\ C_1 \vdash v_1 : \tau_a \xrightarrow{\phi, \mathrm{limit}_f} \tau_b \quad C_2 \vdash v_2 : \tau_a \\ (\mathrm{limit}_C = \mathrm{limit}_f = \infty) \,\mathrm{or}\, (B \vdash \mathrm{limit}_f < \mathrm{limit}_C) \\ \mathrm{or}\, (\mathrm{limit}_C = \infty,\ \mathrm{limit}_f = I,\ \Phi; \Delta \vdash \tau_b \stackrel{\phi_b}{:} 0) \end{array}}{C_1, C_2 \vdash v_1 v_2 : \tau_b}$.
By canonical forms, $v_1 = \lambda x : \tau_a \xrightarrow{\phi, limit_f} e_0$, so e steps by (E-APPABS).

3. $e = v_1 \tau_2$, where $(T-TAPP)\dfrac{\begin{array}{c} C \vdash v_1 : \forall \alpha : K; B.\tau_1 \\ C \vdash \tau_2 : K \\ C \vdash [\alpha \mapsto \tau_2]B \end{array}}{C \vdash v_1 \tau_2 : [\alpha \mapsto \tau_2]\tau_1}$.
By canonical forms, $v_1 = \Lambda \alpha : K; B.v$, so e steps by (E-TAPPABS).

4. $e = \mathrm{unpack}\,\alpha, x = v_1 \,\mathrm{in}\, e_2$, where $(T-UNPACK)\dfrac{\begin{array}{c} C_1 \vdash v_1 : \exists \alpha : K; B.\tau_1 \\ C_2, \alpha : K, x : \tau_1, B \vdash e_2 : \tau_2 \end{array}}{C_1, C_2 \vdash \mathrm{unpack}\,\alpha, x = v_1 \,\mathrm{in}\, e_2 : \tau_2}$.
By canonical forms, $v_1 = \mathrm{pack}[\tau_1, v] \,\mathrm{as}\, \exists \alpha : K; B.\tau_2$, so e steps by (E-UNPACK).

5. $e = \mathrm{in\_domain}(I_1, I_2, v_1, v_2)$, where $(T-INDOMAIN)\dfrac{\begin{array}{c} C_1, C_2 \vdash I_1 : \mathrm{int} \quad C_1, C_2 \vdash I_2 : \mathrm{int} \\ C_1 \vdash v_1 : \mathrm{InDomain}(I_1, \tau_f) \quad C_2 \vdash v_2 : \mathrm{Gen}(\tau_f, I_2) \end{array}}{C_1, C_2 \vdash \mathrm{in\_domain}(I_1, I_2, v_1, v_2) : \wedge \langle \mathrm{Know}(0 \leq I_1 \wedge I_1 < I_2), \mathrm{Gen}(\tau_f, I_2) \rangle}$.
By canonical forms, $v_1 = \mathrm{fact}$ and $v_2 = \mathrm{fact}$, so e steps by (E-DOMAIN).

6. The pattern in the previous cases is pretty clear: the type checking rule implies that any subexpression values have a canonical form, which then allows e to step. The cases for $e_1 \,\mathrm{op}\, e_2$, $\neg e$, $\mathrm{case}(b, e)$, $\mathrm{unroll}(e)$, $\mathrm{apply\_eq}(\tau, e_1, e_2)$, $\mathrm{discard\_fun}(e)$, $\mathrm{define\_fun}(e, \tau)$, $\mathrm{distinguish}(I_1, I_2, e_1, e_2)$, and $\mathrm{in\_domain}(I_1, I_2, e_1, e_2)$ follow exactly the same pattern.

7. $e = \mathrm{new\_fun}(K)$, $e = \mathrm{make\_eq}(\tau)$, $e = \mathrm{coerce}(v)$, $e = \mathrm{fix}\, x : \tau.\mathrm{v}$: these always step, so we don't even need to look at the type checking rules.

8. $e = \mathrm{let}\,\langle x_1, \ldots, x_n\rangle = v_a \,\mathrm{in}\, e_b$, where $(T-LET)\frac{C_a \vdash v_a : \langle \vec{\tau} \rangle \quad C_b, \vec{x}:\vec{\tau} \vdash e_b : \tau_b}{C_a, C_b \vdash \mathrm{let}\langle \vec{x} \rangle = v_a \,\mathrm{in}\, e_b : \tau_b}$.
By canonical forms, $v_a = \phi\langle v_1, \ldots, v_n\rangle$, so e steps by (E-LET).

9. $e = \mathrm{if}\, v_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3$, where $(T-IFE)\frac{C_a \vdash v_1 : \mathrm{Bool}(B) \quad C_b, B \vdash e_2 : \tau \quad C_b, \neg B \vdash e_3 : \tau}{C_a, C_b \vdash \mathrm{if}\, v_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3 : \tau}$.
By canonical forms, $v_1 = b$, so e steps by (E-IF1) or (E-IF2).

10. $e = \mathrm{if}\, B \,\mathrm{then}\, e_1 \,\mathrm{else}\, e_2$, where $(T-IFB)\frac{C = \Psi;\Phi;\Delta;\Gamma;B_C;I \quad C \vdash B : \mathrm{bool} \quad C, B \vdash e_1 : \tau \quad C, \neg B \vdash e_2 : \tau}{C \vdash \mathrm{if}\, B \,\mathrm{then}\, e_1 \,\mathrm{else}\, e_2 : \tau}$.
Since no type variables are in scope, B cannot contain any type variables, which means that $C \vdash B \doteq b$ for some b, so e steps by (E-IFB1) or (E-IFB2).

11. $e = \mathrm{load}(v_{\mathrm{ptr}}, v_{\mathrm{Has}})$, where $(T-LOAD)\frac{C_1 \vdash v_{\mathrm{ptr}} : \mathrm{Int}(I) \quad C_2 \vdash v_{\mathrm{Has}} : \mathrm{Has}(I,\tau)}{C_1, C_2 \vdash \mathrm{load}(v_{\mathrm{ptr}}, v_{\mathrm{Has}}) : \wedge\langle \tau, \mathrm{Has}(I,\tau)\rangle}$.
By canonical forms, $v_{ptr} = i_{ptr}$, and $v_{Has} = \mathrm{fact}$. By inversion, $C_1 \vdash i_{ptr} : Int(i_{ptr})$ and $C_2 = C_2', i_{has} \mapsto \tau \vdash \mathrm{fact} : \mathrm{Has}(i_{has}, \tau)$. Together, these imply $i_{ptr} = i_{has}$. Since $i_{has} \mapsto \tau \in \Psi$ and memory M is well-typed under $C_{Me} = \Psi_{Me}; ...$, where $\Psi_{Me} = \Psi_{spare}, \Psi$, we know $i_{has} \mapsto \tau \in \Psi_{Me}$, so $M(i_{has})$ exists, so $M(i_{ptr})$ exists, so e steps by (E-LOAD).

12. $e = \mathrm{store}(v_{\mathrm{ptr}}, v_{\mathrm{Has}}, v_{\mathrm{v}})$, where $(T-STORE)\frac{C = C_1, C_2, C_3 = \Psi;\Phi;\Delta;\Gamma;B;\infty \quad C_2 \vdash v_{\mathrm{Has}} : \mathrm{Has}(I, \tau_1) \quad C_3 \vdash v_{\mathrm{v}} : \tau_2 \quad C_1 \vdash v_{\mathrm{ptr}} : \mathrm{Int}(I) \quad C \vdash \tau_2 \,\dot{:}\, 1}{C \vdash \mathrm{store}(v_{\mathrm{ptr}}, v_{\mathrm{Has}}, v_{\mathrm{v}}) : \mathrm{Has}(I, \tau_2)}$.
By canonical forms, $v_{ptr} = i_{ptr}$, and $v_{Has} = \mathrm{fact}$, so e steps by (E-STORE).

7 Strong Normalization

This section proves that well-typed terms in a limited environment eventually step to a value: if $\Psi_{Me}; \Phi_{Me}; \varnothing; \varnothing; \mathrm{true}; i \vdash (M, e : \tau)$, then there is some v so that $(M, e) \xrightarrow{*} (M, v)$.

The proof is by induction on the limit i, the maximum nesting depth of let and unpack expressions, and the size of e. Define the depth as follows:

$\mathrm{depth}(x) = 0$
$\mathrm{depth}(\lambda x : \tau \xrightarrow{\phi} e) = 0$
$\mathrm{depth}(\mathrm{fix}\, x.v) = 0$
$\mathrm{depth}(\mathrm{let}\, x_1, \cdots, x_n = e_1 \,\mathrm{in}\, e_2) = \max(\mathrm{depth}(e_1), 1 + \mathrm{depth}(e_2))$
$\mathrm{depth}(\mathrm{unpack}\, \alpha, x = e_1 \,\mathrm{in}\, e_2) = \max(\mathrm{depth}(e_1), 1 + \mathrm{depth}(e_2))$

For all other expressions, the depth is the maximum depth of any of the immediate subexpressions: $\mathrm{depth}(e_1\, e_2) = \max(\mathrm{depth}(e_1), \mathrm{depth}(e_2))$, $\mathrm{depth}(\mathrm{pack}[\tau_1, e] \,\mathrm{as}\, \exists\alpha : K; B.\tau_2) = \mathrm{depth}(e)$, and so on.

Define $\mathrm{size}(v)$ to be 1. For a non-value expression e, define $\mathrm{size}(e)$ to be 1 plus the size of each of e's subexpressions.

Lemma 0: $\mathrm{depth}(v) = 0$. Proof by induction on the structure of v. The only value that has a non-value expression inside it is $\lambda x : \tau \xrightarrow{\phi} e$, which has depth 0 by definition.

Lemma 1: $\mathrm{depth}([x \mapsto v]e) \leq \mathrm{depth}(e)$. Easy proof by induction on the structure of e.

Lemma 2: $[x \mapsto e]v$ is a value. Easy proof by induction on the structure of v.

7.1 LEMMA [SINGLE-STEP SIZE REDUCTION]

If $\Psi_{Me}; \Phi_{Me}; \varnothing; \varnothing; \mathrm{true}; i \vdash (M, e : \tau)$ and $\Psi; \Phi; \varnothing; \varnothing; \mathrm{true}; i \vdash e : \tau$ and $(M, e) \rightarrow (M', e')$ by a non-congruence rule, then $M = M'$ and one of these three must be true:

1. $e' = \mathrm{coerce}(e'')$ and $i' < i$, $\Psi'; \Phi'; \varnothing; \varnothing; \mathrm{true}; i' \vdash e'' : \tau$
2. $\Psi'; \Phi'; \varnothing; \varnothing; \mathrm{true}; i \vdash e' : \tau$, and $\mathrm{depth}(e') < \mathrm{depth}(e)$

3. $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$, and $\text{depth}(e') = \text{depth}(e)$, and $\text{size}(e') < \text{size}(e)$

Proof :

By cases on the derivation that $(M, e) \to (M, e')$.

1. $e = (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2$
and $e' = \text{coerce}([x \mapsto v_2]e_1)$, and $e'' = [x \mapsto v_2]e_1$
By Lemma 6.2.1.(2) and (T-EQ) rule, we know
$\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2 : \tau_b$,
$\Psi_1; \Phi_1; \varnothing; \varnothing; \text{true}; i_f \vdash e_1 : \tau_a \xrightarrow{f} \tau_b$, and $(i > i_f)$
$\Psi_2; \Phi_2; \varnothing; \varnothing; \text{true}; i \vdash v_2 : \tau_a$
By Limit-Change Lemma, $\Psi_2; \Phi_2; \varnothing; \varnothing; \text{true}; i_f \vdash v_2 : \tau_a$
By Term Substitution Lemma, we obtain:
$\Psi; \Phi; \varnothing; \varnothing; \text{true}; i_f \vdash [x \mapsto v_2]e_1 : \tau_b$
$\Psi; \Phi; \varnothing; \varnothing; \text{true}; i_f \vdash e'' : \tau_b$
Now case 1 of the lemma will be true.

2. $e = (\lambda x : \tau \xrightarrow{\phi, \infty} e_1)v_2$ won't type-check, because if $\text{limit}_f = \infty$, then $\text{limit}_C = \infty$.

3. $e = \text{let}\, x_1, \cdots, x_n = e_1 \,\text{in}\, e_2 \quad (e_1 = \phi\langle v_1, \cdots, v_n \rangle)$
$e' = [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e_2$
In Preservation Lemma we have proved, $\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$
From lemma 1, we know $\text{depth}([x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e_2) \leq e_2$
And $\text{depth}(\text{let}\, x_1, \cdots, x_n = e_1 \,\text{in}\, e_2) = \max(\text{depth}(e_1), 1 + \text{depth}(e_2))$,
so $\text{depth}(e') < \text{depth}(\text{let}\, x_1, \cdots, x_n = e_1 \,\text{in}\, e_2)$
Now case 2 of the lemma will be true.

4. $e = \text{unpack}\, \alpha, x = e_1 \,\text{in}\, e_2$
$e' = [\alpha \mapsto \tau_0, x \mapsto v]e_2$
In Preservation Lemma we have proved, $\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$
From lemma 1, we know $\text{depth}([\alpha \mapsto \tau_0, x \mapsto v]e_2) \leq e_2$
And $\text{depth}(\text{unpack}\, \alpha, x = e_1 \,\text{in}\, e_2) = \max(\text{depth}(e_1), 1 + \text{depth}(e_2))$,
so $\text{depth}(e') < \text{depth}(\text{unpack}\, \alpha, x = e_1 \,\text{in}\, e_2)$
Now case 2 of the lemma will be true.

5. All other cases. In Preservation Lemma we have proved, $\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$.
$\text{size}(\text{load}(i, \text{fact})) = 1 + \text{size}(i) + \text{size}(\text{fact}) > \text{size}({}^\wedge\langle M(i), \text{fact}\rangle) = 1$, because $M(i)$ is a value, so ${}^\wedge\langle M(i), \text{fact}\rangle$ is a value, and the size of a value is defined to be 1.
$\text{size}(\text{coerce}(v)) = 1 + \text{size}(v) > \text{size}(v) = 1$
$\text{size}((\Lambda\alpha : K; B.v)\tau) = 1 + \text{size}(v) > \text{size}(v) = 1$
$\text{size}(\text{if}\, v_1 \,\text{then}\, e_2 \,\text{else}\, e_3) = 1 + \text{size}(v_1) + \text{size}(e_2) + \text{size}(e_3) > \text{size}(e_2) \,\text{or}\, \text{size}(e_2)$
$\text{size}(\text{if}\, B \,\text{then}\, e_2 \,\text{else}\, e_3) = 1 + \text{size}(e_2) + \text{size}(e_3) > \text{size}(e_2) \,\text{or}\, \text{size}(e_2)$
$\text{size}(\text{unroll}(\text{roll}[\tau](v))) = 1 + 1 + \text{size}(v) > \text{size}(v)$
$\text{size}(\text{fix}\, x : \tau.v) = 1 + \text{size}(v) > \text{size}([x \mapsto \text{fix}\, x : \tau.v]v) = 1 \quad$ (By Lemma 2)
$\text{size}(\text{case}(b, \text{union}(b, \tau_1, \tau_2, v))) = 1 + 1 + \text{size}(b) + \text{size}(v) + \text{size}(b) > \text{size}(v) = 1$
$\text{size}(\text{in\_domain}(I_1, I_2, \text{fact}, \text{fact})) = 1 + \text{size}(I_1) + \text{size}(I_2) + \text{size}(\text{fact}) + \text{size}(\text{fact}) > \text{size}(e') = 1$
$\text{size}(\text{distinguish}(I_1, I_2, \text{fact}, \text{fact})) = \text{size}(I_1) + \text{size}(I_2) + \text{size}(\text{fact}) + \text{size}(\text{fact}) > \text{size}(e') = 1$
$\text{size}(\text{make\_eq}(\tau)) > \text{size}(\text{fact}) = 1$
$\text{size}(\text{apply\_eq}(\tau, \text{fact}, v)) > \text{size}(v) = 1$
$\text{size}(\text{new\_fun}(K)) > \text{size}(e') = 1$
$\text{size}(\text{discard\_fun}(\text{fact})) > \text{size}(\cdot\langle\rangle) = 1$
$\text{size}(\text{define\_fun}(\text{fact}, \tau)) > \text{size}({}^\wedge\langle \text{fact}, \text{fact}, \text{fact}\rangle) = 1$

Notice that e cannot be a store expression, because store expressions do not type-check in an environment with limit

i. In the absence of the store and congruence evaluation rules, no evaluation rule alters memory, so $M' = M$.

7.2 THEOREM [STRONG NORMALIZATION]

If $\Psi_{Me}; \Phi_{Me}; \varnothing; \varnothing; \text{true}; i \vdash (M, e : \tau)$, then e must step to a value in a finite sequence of zero or more steps, without changing memory: $(M, e) \rightarrow (M, e_1) \rightarrow \cdots \rightarrow (M, e_n) \rightarrow (M, v)$. (Notation: $(M, e) \xrightarrow{*} (M, v)$).

Observe that by the rule for typing (M, e), we know that $\Psi_{Me} = \Psi_{spare}, \Psi$ and $\Phi_{Me} = \Phi_{spare}, \Phi$ and $C \vdash e : \tau$, where $C = \Psi; \Phi; \varnothing; \varnothing; B; \text{limit}$.

Proof :

By induction on the limit i. We assume theorem is true for all $i' < i$

For $i' = i$ by induction on depth(e). We assume theorem is true for all $i' = i$, and depth(e') $<$ depth(e)

For $i' = i$, and depth(e') $=$ depth(e) by induction on size(e). We assume theorem is true for all $i' = i$, depth(e') $=$ depth(e), and size(e') $<$ size(e).

If e is a already a value v, then in zero steps, (M, e) steps to (M, v). Otherwise, we'll show that (M, e) steps to (M, v) in two stages: first, use any applicable congruence rules if $e = E[e_0]$ for some E and e_0. After congruence is exhausted, use the single-step size reduction lemma.

The congruence stage is the same pattern for all expressions, so we arbitrarily pick $e = \text{load}(e_1, e_2)$ to illustrate the pattern; the proof is the same for other expressions. Suppose $e_1$is not a value. By the rule for typing (M, e), we know that $\Psi_{Me} = \Psi_{spare}, \Psi$ and $\Phi_{Me} = \Phi_{spare}, \Phi$ and $C \vdash e : \tau$, where $C = \Psi; \Phi; \varnothing; \varnothing; B; i$, and by inspection of the type checking rules, we conclude that there is some $C_1 = \Psi_1; \Phi_1; \varnothing; \varnothing; B; i_1$, where $i_1 \leq i$, so that $C_1 \vdash e_1 : \tau_1$ (for $e = \text{load}(e_1, e_2)$, we actually know $i_1 = i$, but more generally, we can only say $i_1 \leq i$, since $e = \text{coerce}(e_1)$ requires $i_1 < i$). Since e_1 is a subexpression inside e, it is easy to show that depth(e_1) $\leq$ depth(e), and size(e_1) $<$ size(e). Therefore, either $i_1 < i$, or $i_1 = i$ and depth(e_1) $<$ depth(e), or $i_1 = i$ and depth(e_1) $=$ depth(e) and size(e_1) $<$ size(e), so induction tells us that $(M, e_1) \xrightarrow{*} (M, v_1)$. By repeated application of the congruence rule, $(M, \text{load}(e_1, e_2)) \xrightarrow{*} (M, \text{load}(v_1, e_2))$. We can repeat this argument for e_2 to show that $(M, \text{load}(v_1, e_2)) \xrightarrow{*} (M, \text{load}(v_1, v_2))$. Thus, we step to an expression $e_c = \text{load}(v_1, v_2)$ where no congruence rules apply. If this expression is a value (not possible for $\text{load}(v_1, v_2)$, but possible in other cases, such as tuples), then we've proved that (M, e) steps to some (M, v). Otherwise, type preservation tells us that (M, e_c) is well-typed with the same limit i: $\Psi'_{Me}; \Phi'_{Me}; \varnothing; \varnothing; \text{true}; i \vdash (M, e_c : \tau)$.

Once we've stepped $(M, e) \xrightarrow{*} (M, e_c)$, where e_c is not a value and no congruence rules apply to e_c, type progress tells us that $(M, e_c) \rightarrow (M', e')$ for some (M', e'). Then the single-step size reduction lemma says that $(M, e_c) \rightarrow (M, e')$, where one of three cases must be true:

1. $e' = \text{coerce}(e'')$ and $i' < i$, $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i' \vdash e'' : \tau$
2. $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$, and depth($e'$) $<$ depth(e_c)
3. $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$, and depth($e'$) $=$ depth(e_c), and size(e') $<$ size(e_c)

In each case, we prove that $(M, e_c) \xrightarrow{*} (M, v)$:

1. $e' = \text{coerce}(e'')$ and $i' < i$, $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i' \vdash e'' : \tau$. In this case, e'' type-checks in an environment with a smaller limit $i' < i$. By induction on i, we know $(M, e'') \xrightarrow{*} (M, v)$. By repeated applications of the congruence rule for $\text{coerce}(e'')$, we conclude that $(M, \text{coerce}(e'')) \xrightarrow{*} (M, \text{coerce}(v))$. E-COERCE then says that $(M, \text{coerce}(v)) \rightarrow (M, v)$. Putting the $(M, e_c) \rightarrow (M, e')$ and $(M, \text{coerce}(e'')) \xrightarrow{*} (M, \text{coerce}(v))$ and $(M, \text{coerce}(v)) \rightarrow (M, v)$ steps together, we conclude $(M, e_c) \xrightarrow{*} (M, v)$.

2. $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$, and depth($e'$) $<$ depth(e_c). By induction on depth(e), we know $(M, e') \xrightarrow{*} (M, v)$. Putting this together with $(M, e_c) \rightarrow (M, e')$, we conclude $(M, e_c) \xrightarrow{*} (M, v)$.

3. $\Psi'; \Phi'; \varnothing; \varnothing; \text{true}; i \vdash e' : \tau$, and depth($e'$) $=$ depth(e_c), and size(e') $<$ size(e_c). By induction on size(e), we know $(M, e') \xrightarrow{*} (M, v)$. Putting this together with $(M, e_c) \rightarrow (M, e')$, we conclude $(M, e_c) \xrightarrow{*} (M, v)$.

All three cases prove $(M, e_c) \xrightarrow{*} (M, v)$. This, together with $(M, e) \xrightarrow{*} (M, e_c)$, shows $(M, e) \xrightarrow{*} (M, v)$.

8 Progress after Type Erasure

This section proves that progress holds after type erasure.

If e is a closed, well-typed expression ($C \vdash e : \tau$ for some τ and $C = \Psi; \Phi; \emptyset; \emptyset; B; \text{limit}$), then the following statements hold:

1. (Erasure-Progress-1) If $(M, e) \mapsto (M', e')$ then $\text{erase}((M, e)) \overset{?}{\mapsto} \text{erase}((M', e'))$.
2. (Erasure-Progress-2) If $\text{erase}(e)$ is a value then $(M, e) \overset{*}{\mapsto} (M', v)$, $\text{erase}((M, e)) = \text{erase}((M', v))$.
3. (Erasure-Progress-3) If $\text{erase}((M, e)) \mapsto (L', d')$ then $(M, e) \overset{+}{\mapsto} (M', e')$, $\text{erase}((M', e')) = (L', d')$.

In the cases where e does not effect memory ($M = M'$), $(M, e) \mapsto (M', e')$ is abreviated to $e \mapsto e'$. The cases involving loads, stores, and congruence rules have the potential to affect memory and cannot be shortened.

8.1 Lemmas

8.1.1 LEMMA [ERASE-SIMPLIFY]

If $\text{simplify}(e)$ exists, then $\text{erase}(\text{simplify}(e)) = \text{simplify}(\text{erase}(e))$. Proof by cases of e for which $\text{simplify}(e)$ is defined (in all such cases, $e = \text{erase}(e)$).

8.1.2 LEMMA [ERASE-TERM-SUBSTITUTION]

$\text{erase}([x \mapsto v]e) = [x \mapsto \text{erase}(v)]\,\text{erase}(e)$. Proof by induction on the expression e.

8.1.3 LEMMA [ERASE-TYPE-SUBSTITUTION]

$\text{erase}([\alpha \mapsto \tau]e) = \text{erase}(e)$. Proof by induction on the expression e.

8.1.4 LEMMA [VALUE-ERASE-VALUE]

A value will always erase to a value. Proof by induction on the values.

1. $\text{erase}(i) = i$ by (ER-i).
2. $\text{erase}(b) = b$ by (ER-b).
3. $\text{erase}(\Lambda\alpha : K; B.v) = \text{erase}(v)$by (ER-TFUN). Another value erase rule will subsequently apply.
4. $\text{erase}(\text{pack}[\tau_1, v]\,\text{as}\,\exists\alpha : K; B.\tau_2) = \text{erase}(v)$by (ER-PACK). Another value erase rule will subsequently apply.
5. $\text{erase}(\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v)) = \text{erase}(v)$ by (ER-ROLL). Another value erase rule will subsequently apply.
6. $\text{erase}(\lambda x : \tau \xrightarrow{\phi, I} e) = \langle\rangle$ by (ER-FUNI).
7. $\text{erase}(\lambda x : \tau \xrightarrow{\phi, \infty} e) = \lambda x \longrightarrow \text{erase}(e)$ by (ER-FUN). Another erase rule will apply but this is already a value.
8. $\text{erase}(\phi\langle\overrightarrow{v}\rangle) = \langle\text{erase}(v_1), \text{erase}(v_2), \ldots, \text{erase}(v_n)\rangle$ by (ER-TUPLE). Other value erase rules will subsequently apply.
9. $\text{erase}(\text{union}(b, \tau_1, \tau_2, v)) = \text{erase}(v)$ by (ER-UNION). Another value erase rule will subsequently apply.
10. $\text{erase}(\text{fact}) = \langle\rangle$ by (ER-FACT).

All the value erase rules produce either values or the erase of another value.

8.1.5 LEMMA [ZERO-ERASE-VALUE]

If $C \vdash v : \tau$then $\text{erase}(v : t :\overset{\phi}{0}) = \langle\rangle$. A value with Kind $\overset{\phi}{0}$ will erase to $\langle\rangle$. Proof by induction on the values.

- Case: i. By Inversion and (T-INT), $i : \text{Int}(i)$. By (K-INT), $i : \text{Int}(i) :\dot{1}$. This does not have Kind $\overset{\phi}{0}$.
- Case: b. By Inversion and (T-BOOL), $b : \text{Bool}(b)$. By (K-BOOL), $b : \text{Bool}(b) :\dot{1}$. This does not have Kind $\overset{\phi}{0}$.
- Case: $\Lambda\alpha : K; B.v$. By (ER-TFUN) $\text{erase}(\Lambda\alpha : K; B.v) = \text{erase}(v)$. By (T-TABS), $v : \tau$ and $(\Lambda\alpha : K; B.v) : \forall\alpha : K; B.\tau$. By induction, if $v : t :\overset{\phi}{0}$, $\text{erase}(v) = \langle\rangle$. By (K-ALL), $(\forall\alpha : K; B.\tau) : K_2$ and $\tau : K_2$. Therefore $(\Lambda\alpha : K; B.v) : (\forall\alpha : K; B.\tau) : K_2$ and $v : \tau : K_2$. If $K_2 =\overset{\phi}{0}$, $\text{erase}((\Lambda\alpha : K; B.v) : (\forall\alpha : K; B.\tau) :\overset{\phi}{0})= \text{erase}(v : \tau :\overset{\phi}{0}) = \langle\rangle$.
- Case: $\text{pack}[\tau_1, v] \,\text{as}\, \exists\alpha : K; B.\tau_2$. By Inversion and (T-PACK), $\text{pack}[\tau_1, v] \,\text{as}\, \exists\alpha : K; B.\tau_2 : K; B.\tau_2$ and $\tau_1 : K$ and $v : [\alpha \mapsto \tau_1]\tau_2$. By (ER-PACK), $\text{erase}(\text{pack}[\tau_1 : K, v] \,\text{as}\, \exists\alpha : K; B.\tau_2 : K; B.\tau_2) = \text{erase}(v : \tau_1 : K)$. By induction, if $v : t :\overset{\phi}{0}$, $\text{erase}(v) = \langle\rangle$. If $K =\overset{\phi}{0}$, $\text{erase}(\text{pack}[\tau_1 :\overset{\phi}{0}, v] \,\text{as}\, \exists\alpha :\overset{\phi}{0}; B.\tau_2 :\overset{\phi}{0}; B.\tau_2)= \text{erase}(v : \tau_1 :\overset{\phi}{0}) = \langle\rangle$.
- Case: $\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v)$. By Inversion and (T-ROLL), $\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) : ((\mu\alpha : K.\tau_0)\tau_1 \ldots \tau_n)$ and $v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \ldots \tau_n$ and $\tau_0 : K$. By (K_REC), $(\mu\alpha : K.\tau_0) : K$. Therefore, $v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \ldots \tau_n : K$ and $\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) : ((\mu\alpha : K.\tau_0)\tau_1 \ldots \tau_n) : K$. By (ER-ROLL), $\text{erase}(\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v)) = \text{erase}(v)$. By induction, if $v : t :\overset{\phi}{0}$, $\text{erase}(v) = \langle\rangle$. If $K =\overset{\phi}{0}$, $\text{erase}(\text{roll}[(\mu\alpha :\overset{\phi}{0} .\tau_0)\tau_1 \cdots \tau_n](v) : ((\mu\alpha :\overset{\phi}{0} .\tau_0)\tau_1 \ldots \tau_n) :\overset{\phi}{0})= \text{erase}(v([\alpha \mapsto \mu\alpha :\overset{\phi}{0} .\tau_0]\tau_0)\tau_1 \ldots \tau_n :\overset{\phi}{0}) = \langle\rangle$.
- Case: $\lambda x : \tau \xrightarrow{\phi,\text{limit}} e$. By (T-ABS), $\lambda x : \tau \xrightarrow{\phi,\text{limit}} e : \tau_1 \overset{\phi,\text{limit}}{\mapsto} \tau_2$. If limit $= \infty$, by (K-ABS), $\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2 :\overset{\phi}{1}$. This does not have Kind $\overset{\phi}{0}$. If limit $= I$, by (K-ABS), $\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2 :\overset{\phi}{0}$ and by (ER-FUNI) $\text{erase}(\lambda x : \tau \xrightarrow{\phi,I} e) = \langle\rangle$.
- Case: $\phi\langle\overrightarrow{v}\rangle$. BY (ER-TUPLEv0), $\text{erase}(\phi\langle\overrightarrow{v}\rangle : \tau :\overset{\phi}{0}) = \langle\rangle$.
- Case: $\text{union}(b, \tau_1, \tau_2, v)$. By (T-UNION), $\tau_1 : K$ and $\tau_2 : K$ and $v : \tau_i$ where $i = 1$ if b and $i = 2$ if $\neg b$ and $\text{union}(b, \tau_1, \tau_2, v) : \text{Union}(b, \tau_1, \tau_2)$. Therefore $v : \tau_i : K$. By induction, if $v : t :\overset{\phi}{0}$, $\text{erase}(v) = \langle\rangle$. By (K-UNION), $\text{Union}(b, \tau_1, \tau_2) : K$. By (ER-UNION), $\text{erase}(\text{union}(b, \tau_1, \tau_2, v)) = \text{erase}(v)$. If $K =\overset{\phi}{0}$, $\text{erase}(\text{union}(b, \tau_1, \tau_2, v) : \text{Union}(b, \tau_1, \tau_2) :\overset{\phi}{0}) = \text{erase}(v : \tau_i :\overset{\phi}{0}) = \langle\rangle$.
- Case: fact. By (ER-FACT) $\text{erase}(\text{fact}) = \langle\rangle$.

8.1.6 LEMMA [TYPE-SUBSTITUTION-VALUE]

$[\alpha \mapsto \tau]v$ is a value. Proof by induction on the values.

8.1.7 LEMMA [TERM-SUBSTITUTION-VALUE]

$[x \mapsto v]v$ and $[x \mapsto \text{fix}\, x.v]v$ are values. Proof by induction on the values.

8.2 THEOREM [Erasure-Progress-1]

$$\frac{C \vdash (M, e : \tau) \quad (M, e) \mapsto (M', e')}{\text{erase}((M, e)) \overset{?}{\mapsto} \text{erase}((M', e'))}$$

If (M, e) evaluates in one step to (M', e'), then $\text{erase}((M, e))$ evaluates in zero or one steps to $\text{erase}((M', e'))$. Proof by induction on the typed evaluation rules. There are several cases for this proof. For each, we list the evaluation rules which follow the case and show one proof. The other proofs in each case can be obtained using a similar proof to the example.

8.2.1 Case: $(M, e) \mapsto (M', e')$ using a typed evaluation rule and erase$((M, e)) \mapsto$ erase$((M', e'))$ using the corresponding untyped (or typed) evaluation rule.

1. $(E-LOAD)$If $e \overset{(E-LOAD)}{\mapsto} e'$ then $(M, e) = (M, \text{load}(i, \text{fact}))$ and $(M', e') = (M, ^\wedge \langle M(i), \text{fact}\rangle)$.
2. $(E-STORE)$ If $e \overset{(E-STORE)}{\mapsto} e'$ then $(M, e) = (M, \text{store}(i, \text{fact}, v))$ and $(M', e') = ([i \mapsto v]M, \text{fact})$.
3. $(E-ABSAPP2)$ If $e \overset{(E-ABSAPP2)}{\mapsto} e'$ then $e = (\lambda x : \tau \overset{\phi,\infty}{\longrightarrow} e_1)v_2$ and $e' = [x \mapsto v_2]e_1$.
4. $(E-LET)$ If $e \overset{(E-LET)}{\mapsto} e'$ then $e = \text{let}\,\langle x_1, \ldots, x_n\rangle = \phi\langle v_{1,\ldots,v_n}\rangle\,\text{in}\,e_1$ and $e' = [x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]e_1$.
5. $(E-SIMPLIFY1)$ If $e \overset{(E-SIMPLIFY1)}{\mapsto} e'$ then $e = v_1\,\text{op}\,v_2$ and $e' = \text{simplify}(v_1\,\text{op}\,v_2)$.
6. $(E-SIMPLIFY2)$ If $e \overset{(E-SIMPLIFY2)}{\mapsto} e'$ then $e = \neg v$ and $e' = \text{simplify}(\neg v)$.
7. $(E-IF1)$ If $e \overset{(E-IF1)}{\mapsto} e'$ then $e = \text{if}\,true\,\text{then}\,e_1\,\text{else}\,e_2$ and $e' = e_1$.
8. $(E-IF2)$ If $e \overset{(E-IF1)}{\mapsto} e'$ then $e = \text{if}\,false\,\text{then}\,e_1\,\text{else}\,e_2$ and $e' = e_2$.

Proof :

$(E-ABSAPP2)$ If $e \overset{(E-ABSAPP2)}{\mapsto} e'$ then $e = (\lambda x : \tau \overset{\phi,\infty}{\longrightarrow} e_1)v_2$ and $e' = [x \mapsto v_2]e_1$. erase(e) = erase$((\lambda x : \tau \overset{\phi,\infty}{\longrightarrow} e_1)v_2)$= erase$(\lambda x : \tau \overset{\phi,\infty}{\longrightarrow} e_1)\,$erase$(v_2)$ by (ER-APP). This equals $(\lambda x \longrightarrow$ erase$(e_1))\,$erase(v_2) by (ER-FUN). erase(e') = erase$([x \mapsto v_2]e_1)$= $[x \mapsto$ erase$(v_2)]\,$erase(e_1) by Erase-Term-Substitution. Finally, $(\lambda x \longrightarrow$ erase$(e_1))\,$erase$(v_2) \overset{(U-ABSAPP)}{\mapsto} [x \mapsto$ erase$(v_2)]\,$erase(e_1) so erase$(e) \mapsto$ erase(e').

8.2.2 Case: erase(e) = erase(e').

1. $(E-TAPPTABS)$ If $e \overset{(E-TAPPTABS)}{\mapsto} e'$ then $e = (\Lambda\alpha : K; B.v)\tau$ and $e' = [\alpha \mapsto \tau]v$.
2. $(E-UNROLL)$ If $e \overset{(E-UNROLL)}{\mapsto} e'$ then $e = \text{unroll}(\text{roll}[\tau](v))$ and $e' = v$.
3. $(E-CASE)$ If $e \overset{(E-CASE)}{\mapsto} e'$ then $e = \text{case}(b, \text{union}(b, \tau_1, \tau_2, v))$ and $e' = v$.
4. $(E-MAKEEQ)$ If $e \overset{(E-MAKEEQ)}{\mapsto} e'$ then $e = \text{make\_eq}(\tau)$ and $e' = \text{fact}$.
5. $(E-NEWFUN)$ If $e \overset{(E-NEWFUN)}{\mapsto} e'$ then $e = \text{new\_fun}(K)$ and $e' = \text{pack}[F^K, \text{fact}]\,\text{as}\,\exists\alpha : \text{int} \mapsto K.\text{FunGen}(\alpha, 0)$, where F^K is fresh.

Proof :

$(E-TAPPTABS)$ If $e \overset{(E-TAPPTABS)}{\mapsto} e'$ then $e = (\Lambda\alpha : K; B.v)\tau$ and $e' = [\alpha \mapsto \tau]v$. erase$(e)$ = erase$((\Lambda\alpha : K; B.v)\tau)$= erase$(\Lambda\alpha : K; B.v)$ by (ER-APPT). This equals erase(v) by (ER-TFUN). erase(e') = erase$([\alpha \mapsto \tau]v)$ = erase(v) by Erase-Type-Substitution so erase(e) = erase(e').

8.2.3 Case: erase$((M, e)) \mapsto$ erase$((M', e'))$ using (E-LET).

1. $(E-ABSAPP1)$ If $e \overset{(E-ABSAPP1)}{\mapsto} e'$ then $e = (\lambda x : \tau \overset{\phi,I}{\longrightarrow} e_1)v_2$ and $e' = \text{coerce}([x \mapsto v_2]e_1)$.
2. $(E-APPLYEQ)$ If $e \overset{(E-APPLYEQ)}{\mapsto} e'$ then $e = \text{apply\_eq}(\tau, \text{fact}, v)$ and $e' = v$.
3. $(E-INDOMAIN)$ If $e \overset{(E-INDOMAIN)}{\mapsto} e'$ then $e = \text{in\_domain}(I_1, I_2, \text{fact}, \text{fact})$ and $e' =^\wedge \langle\text{know}(0 \leq I_1 \,\wedge\, I_1 < I_2), \text{fact}\rangle$.

4. $(E-DISTINGUISH)$ If $e \overset{(E-DISTINGUISH)}{\mapsto} e'$ then $e = \text{distinguish}(I_1, I_2, \text{fact}, \text{fact})$ and $e' =^\wedge \langle \text{know}(I_1 \neq I_2), \text{fact}, \text{fact}\rangle$.

5. $(E-DISCARDFUN)$ If $e \overset{(E-DISCARDFUN)}{\mapsto} e'$ then $e = \text{discard\_fun}(\text{fact})$ and $e' = \cdot\langle\rangle$.

6. $(E-DEFINEFUN)$ If $e \overset{(E-DEFINEFUN)}{\mapsto} e'$ then $e = \text{define\_fun}(\text{fact}, \tau)$ and $e' =^\wedge \langle \text{fact}, \text{fact}, \text{fact}\rangle$.

Proof :

$(E-DEFINEFUN)$ If $e \overset{(E-DEFINEFUN)}{\mapsto} e'$ then $e = \text{define\_fun}(\text{fact}, \tau)$ and $e' =^\wedge \langle \text{fact}, \text{fact}, \text{fact}\rangle$. $\text{erase}(e) = \text{erase}(\text{define\_fun}(\text{fact}, \tau)) = \text{let}\, x = \text{erase}(\text{fact})\,\text{in}\,\langle\rangle$ by (ER-DEFINEFUN). This equals $\text{let}\, x = \langle\rangle\,\text{in}\,\langle\rangle$ by (ER-FACT) and evaluates to $\langle\rangle$ using (E-LET). $\text{erase}(e') = \text{erase}(^\wedge\langle \text{fact}, \text{fact}, \text{fact}\rangle) = \langle\rangle$ by (ER-TUPLEv0) so $\text{erase}(e) \mapsto \text{erase}(e')$.

8.2.4 Case: $\textbf{erase}((M, e)) \overset{?}{\mapsto} \textbf{erase}((M', e'))$ using congruence.

1. $(Congruence\, rule) E[e] = e\tau \mid \text{pack}[\tau_1, e]\,\text{as}\,\exists\alpha : K; B.\tau_2 \mid \text{unpack}\,\alpha, x = e\,\text{in}\,e_2$
   $\mid \text{roll}[\tau](e) \mid \text{unroll}(e) \mid ee_2 \mid v_1 e \mid \phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle \mid \text{let}\,\overrightarrow{x} = e\,\text{in}\,e_2$
   $\mid e\,\text{op}\,e_2 \mid v_1\,\text{op}\,e \mid \neg e \mid \text{if}\,e\,\text{then}\,e_2\,\text{else}\,e_3 \mid \text{union}(b, \tau_1, \tau_2, e) \mid \text{case}(b, e) \mid \text{load}(e, e_{\text{Has}})$
   $\mid \text{store}(e, e_{\text{Has}}, e_v) \mid \text{store}(\text{v}_{\text{ptr}}, \text{v}_{\text{Has}}, e) \mid \text{apply\_eq}(\tau, e, e_2) \mid \text{apply\_eq}(\tau, \text{v}_1, e) \mid \text{discard\_fun}(e)$
   $\mid \text{load}(v_{\text{ptr}}, e) \mid \text{store}(\text{v}_{\text{ptr}}, e, e_v) \mid \text{coerce}(e)$

Proof :

$(Congruence\, rule)$ If $(M, e) \overset{(congruence\, rule) E[e] = e\,\tau}{\mapsto} (M', e')$ then $e = e_1\tau$ and $e' = e_1'\tau$ and $(M, e_1) \mapsto (M', e_1')$. By induction, $\text{erase}((M, e_1)) \overset{?}{\mapsto} \text{erase}((M', e_1'))$. $\text{erase}((M, e)) = \text{erase}((M, e_1\tau)) = (\text{erase}(M), \text{erase}(e_1\,\tau)$ $= (\text{erase}(M), \text{erase}(e_1))$ by (ER-M) and (ER-APPT). $\text{erase}((M', e')) = \text{erase}((M', e_1'\tau)) = (\text{erase}(M'), \text{erase}(e_1'\,\tau) =$ $(\text{erase}(M'), \text{erase}(e_1'))$ by (ER-M) and (ER-APPT). Finally, $\text{erase}((M, e_1)) = (\text{erase}(M), \text{erase}(e_1))$ and $\text{erase}(M', e_1') =$ $(\text{erase}(M'), \text{erase}(e_1')$ by (ER-M), so $(\text{erase}(M), \text{erase}(e_1)) \overset{?}{\mapsto} (\text{erase}(M'), \text{erase}(e_1'))$ and $\text{erase}((M, e)) \overset{?}{\mapsto} \text{erase}((M', e'))$.

8.2.5 Case: $\textbf{erase}((M, e)) = \textbf{erase}((M', e'))$ using Zero-Erase-Value.

1. $(E-COERCE)$ If $e \overset{(E-COERCE)}{\mapsto} e'$ then $e = \text{coerce}(v)$ and $e' = v$.

Proof :

$\text{erase}(e) = \text{erase}(\text{coerce}(v)) = \langle\rangle$ by (ER-COERCE). $\text{erase}(e') = \text{erase}(v) = \langle\rangle$ by Zero-Erase-Value. $\langle\rangle = \langle\rangle$ so $\text{erase}(e) = \text{erase}(e')$.

8.2.6 Case: $\textbf{erase}((M, e)) \overset{?, (congruence\, rule) E[e] = \textbf{let}\, x = e\, \textbf{in}\, e_2}{\mapsto} \textbf{erase}((M', e'))$.

1. $(Congruence\, rule) E[e] = \text{distinguish}(I_1, I_2, e, e_2) \mid \text{distinguish}(I_1, I_2, \text{v}_1, e) \mid \text{define\_fun}(e, \tau)$
   $\mid \text{in\_domain}(I_1, I_2, e, e_2) \mid \text{in\_domain}(I_1, I_2, \text{v}_1, e)$

Proof :

$(Congruence\, rule)$If $(M, e) \overset{(congruence\, rule) E[e] = \text{distinguish}(I_1, I_2, e, e_2)}{\mapsto} (M', e')$ then $e = \text{distinguish}(I_1, I_2, e_1, e_2)$ and $e' = \text{distinguish}(I_1, I_2, e_1', e_2)$ and $(M, e_1) \mapsto (M', e_1')$. By induction, $\text{erase}((M, e_1)) \overset{?}{\mapsto} \text{erase}((M', e_1'))$. $\text{erase}((M, e)) =$ $\text{erase}((M, \text{distinguish}(I_1, I_2, e_1, e_2))) = (\text{erase}(M), \text{erase}(\text{distinguish}(I_1, I_2, e_1, e_2)))$ by (ER-M). This equals $(\text{erase}(M), \text{let}\,\langle x, y\rangle = \langle \text{erase}(e_1), \text{erase}(e_2)\rangle\,\text{in}\,\langle\rangle)$ by (ER-DISTINGUISH). $\text{erase}((M', e'))$ $= \text{erase}((M', \text{distinguish}(I_1, I_2, e_1', e_2))) = (\text{erase}(M'), \text{erase}(\text{distinguish}(I_1, I_2, e_1, e_2)))$ by (ER-M). This equals $(\text{erase}(M'), \text{let}\,\langle x, y\rangle = \langle \text{erase}(e_1'), \text{erase}(e_2)\rangle\,\text{in}\,\langle\rangle)$ by (ER-DISTINGUISH). Finally, $\text{erase}((M, e_1)) = (\text{erase}(M), \text{erase}(e_1))$ and $\text{erase}(M', e_1') = (\text{erase}(M'), \text{erase}(e_1')$ by (ER-M), so $(\text{erase}(M), \text{erase}(e_1)) \overset{?}{\mapsto} (\text{erase}(M'), \text{erase}(e_1'))$ and

$$(\text{erase}(M), \langle \text{erase}(e_1), \text{erase}(e_2)\rangle) \overset{?, (congruence\, rule) E[e] = \phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle}{\mapsto} (\text{erase}(M'), \langle \text{erase}(e_1'), \text{erase}(e_2)\rangle)$$

so, $(\text{erase}(M), \text{let}\,\langle x, y\rangle = \langle \text{erase}(e_1), \text{erase}(e_2)\rangle\,\text{in}\,\langle\rangle) \overset{?, (congruence\, rule) E[e] = \text{let}\, x = e\,\text{in}\,e_2}{\mapsto}$ $(\text{erase}(M'), \text{let}\,\langle x, y\rangle = \langle \text{erase}(e_1'), \text{erase}(e_2)\rangle\,\text{in}\,\langle\rangle)$.

8.2.7 Case: unpack.

1. $(E-UNPACK)$ If $e \overset{(E-UNPACK)}{\mapsto} e'$ then $e = \text{unpack}\,\alpha, x = (\text{pack}[\tau_1, v_1]\,\text{as}\,\tau_2)\,\text{in}\,e_2$ and $e' = [\alpha \mapsto \tau_1, x \mapsto v_1]e_2$.

Proof :

erase(e) $=$ erase(unpack $\alpha, x = (\text{pack}[\tau_1, v_1]\,\text{as}\,\tau_2)\,\text{in}\,e_2$) equals let $x =$ erase(pack$[\tau_1, v_1]$ as τ_2) in erase(e_2) by (ER-UNPACK). This equals let $x =$ erase(v_1) in erase(e_2) by (ERPACK). erase(e') $=$ erase($[\alpha \mapsto \tau_1, x \mapsto v_1]e_2$)$=$ $[x \mapsto \text{erase}(v_1)]\,\text{erase}(e_2)$ by Erase-Term-Substitution and Erase-Type-Substitution. Finally, let $x =$ erase(v_1) in erase(e_2) $\overset{(E-LET)}{\mapsto}$ $[x \mapsto \text{erase}(v_1)]\,\text{erase}(e_2)$ so erase(e) $\mapsto$ erase(e').

8.2.8 Case: erase$((M, e))$ $\overset{(U-FIX)}{\mapsto}$ erase$((M', e'))$.

1. $(E-FIX)$ If $e \overset{(E-FIX)}{\mapsto} e'$ then $e = \text{fix}\,x : \tau.v$ and $e' = [x \mapsto \text{fix}\,x : \tau.v]v$.

By the type checking rules, x and v are the same size. If $x : t \overset{\phi}{:} 0$, $e = \text{fix}\,x : t \overset{\phi}{:} 0\ .v$ and $e' = [x \mapsto \text{fix}\,x : t \overset{\phi}{:} 0\ .v]v$. erase($e$) $=$ erase(fix $x : t \overset{\phi}{:} 0\ .v$) $= \langle\rangle$ by (ER-FIX0). erase(e') $=$ erase($[x \mapsto \text{fix}\,x : t \overset{\phi}{:} 0\ .v]v$) $= [x \mapsto \text{erase}(\text{fix}\,x : t \overset{\phi}{:} 0\ .v)]\,\text{erase}(v)$ by Erase-Term-Substitution. This equals $[x \mapsto \langle\rangle]\langle\rangle$ by (ER-FIX0) and Zero-Erase-Value. This equals $\langle\rangle$. Therefore, erase(e) $=$ erase(e').

If $x : t \overset{\phi}{:} i$ where $i > 0$, $e = \text{fix}\,x : t \overset{\phi}{:} i\ .v$ and $e' = [x \mapsto \text{fix}\,x : t \overset{\phi}{:} i\ .v]v$. erase($e$) $=$ erase(fix $x : t \overset{\phi}{:} i\ .v$) $= \text{fix}\,x.\,\text{erase}(v)$ by (ER-FIX). erase(e') $=$ erase($[x \mapsto \text{fix}\,x : t \overset{\phi}{:} i\ .v]v$) $= [x \mapsto \text{erase}(\text{fix}\,x : t \overset{\phi}{:} i\ .v)]\,\text{erase}(v)$ by Erase-Term-Substitution. This equals $[x \mapsto \text{fix}\,x.v]\,\text{erase}(v)$ by (ER-FIX). And $\text{fix}\,x.\,\text{erase}(v) \overset{(U-FIX)}{\mapsto} [x \mapsto \text{fix}\,x.\,\text{erase}(v)]\,\text{erase}(v)$ so erase(e) $\mapsto$ erase(e').

8.2.9 Case: These only appear in coersion functions and erase does not apply to them.

1. $(E-IFB1)$ If $e \overset{(E-IFB1)}{\mapsto} e'$ then $\vdash B \doteq true$ and $e = \text{if}\,B\,\text{then}\,e_1\,\text{else}\,e_2$ and $e' = e_1$.
2. $(E-IFB2)$ If $e \overset{(E-IFB2)}{\mapsto} e'$ then $\vdash B \doteq false$ and $e = \text{if}\,B\,\text{then}\,e_1\,\text{else}\,e_2$ and $e' = e_2$.

8.3 c Set

c is the set of expressions e which erase to a value.

$$c = i \,|\, b \,|\, c\tau \,|\, \phi\langle\overrightarrow{c}\rangle : \overrightarrow{\tau} \overset{\phi}{:} i \text{ where } i > 0 \,|\, \phi\langle\overrightarrow{v}\rangle : \overrightarrow{\tau} \overset{\phi}{:} 0 \,|\, \lambda x : \tau \overset{\phi,\text{limit}}{\longrightarrow} e$$

$$|\, \Lambda\alpha : K; B.v \,|\, \text{union}(b, \tau_1, \tau_2, c) \,|\, \text{pack}[\tau_1, c]\,\text{as}\,\exists\alpha : K; B.\tau_2$$

$$|\, \text{case}(b, c) \,|\, \text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](c) \,|\, \text{unroll}(c) \,|\, \text{fix}\,x : \tau \overset{\phi}{:} 0\ .\text{v}$$

$$|\, \text{coerce}(e) \,|\, \text{make\_eq}(\tau) \,|\, \text{new\_fun}(K) \,|\, \text{fact}$$

8.3.1 LEMMA [C-ERASE-VALUE]

erase(c) $= u$. Proof by induction on c.

1. By (ER-i), erase(i) $= i$.
2. By (ER-b), erase(b) $= b$.
3. By (ER-APPT), erase($c\tau$) $=$ erase(c).

4. By (ER-TUPLE), $\mathrm{erase}(\phi\langle\overrightarrow{c}\rangle : \overrightarrow{\tau} \overset{\phi}{:i}) = \langle\overrightarrow{\mathrm{erase}(c)}\rangle$.
5. By (ER-TUPLEv0), $\mathrm{erase}(\phi\langle\overrightarrow{v}\rangle : \overrightarrow{\tau} \overset{\phi}{:0}) = \langle\rangle$.
6. By (ER-FUN), $\mathrm{erase}(\lambda x : \tau \xrightarrow{\phi,\infty} e) = \lambda x \longrightarrow \mathrm{erase}(e)$.
7. By (ER-FUNI), $\mathrm{erase}(\lambda x : \tau \xrightarrow{\phi,I} e) = \langle\rangle$.
8. By (ER-TFUN), $\mathrm{erase}(\Lambda\alpha : K; B.v) = \mathrm{erase}(v)$. By Value-Erase-Value, $\mathrm{erase}(v) = u$.
9. By (ER-UNION), $\mathrm{erase}(\mathrm{union}(b, \tau_1, \tau_2, c)) = \mathrm{erase}(c)$.
10. By (ER-PACK), $\mathrm{erase}(\mathrm{pack}[\tau_1, c]\,\mathrm{as}\,\exists\alpha : K; B.\tau_2) = \mathrm{erase}(c)$.
11. By (ER-CASE), $\mathrm{erase}(\mathrm{case}(b, c)) = \mathrm{erase}(c)$.
12. By (ER-ROLL), $\mathrm{erase}(\mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](c)) = \mathrm{erase}(c)$.
13. By (ER-UNROLL), $\mathrm{erase}(\mathrm{unroll}(c)) = \mathrm{erase}(c)$.
14. By (ER-FIX0), $\mathrm{erase}(\mathrm{fix}\, x : \tau \overset{\phi}{:0} .\mathrm{v}) = \langle\rangle$.
15. By (ER-COERCE), $\mathrm{erase}(\mathrm{coerce}(e)) = \langle\rangle$.
16. By (ER-MAKEEQ), $\mathrm{erase}(\mathrm{make\_eq}(\tau)) = \langle\rangle$.
17. By (ER-NEWFUN), $\mathrm{erase}(\mathrm{new\_fun}(K)) = \langle\rangle$.
18. By (ER-FACT), $\mathrm{erase}(\mathrm{fact}) = \langle\rangle$.

$\mathrm{erase}(c)$ is either a value or another $\mathrm{erase}(c)$. In the case of $\langle\overrightarrow{\mathrm{erase}(c)}\rangle$, each $\mathrm{erase}(c)$ will eventually erase to a value and a tuple of values is a value.

8.3.2 LEMMA [NON-C-ERASE-NON-VALUE]

If $e \neq c$ then, $\mathrm{erase}(e) \neq u$. Proof by induction on the expressions.

1. By (ER-x), $\mathrm{erase}(x) = x$.
2. By (ER-APP), $\mathrm{erase}(e_1 : (\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2)\, e_2) = \mathrm{erase}(e_1)\,\mathrm{erase}(e_2)$.
3. By (ER-APPI), $\mathrm{erase}(e_1 : (\tau_1 \overset{\phi,I}{\mapsto} \tau_2)\, e_2) = \mathrm{let}\, x = \langle\mathrm{erase}(e_1), \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle$.
4. By (ER-TUPLEe0), $\mathrm{erase}(\phi\langle\overrightarrow{e}\rangle : \overrightarrow{\tau} \overset{\phi}{:0}) = \mathrm{let}\, x = \langle\overrightarrow{\mathrm{erase}(e)}\rangle\, \mathrm{in}\, \langle\rangle$.
5. By (ER-OP), $\mathrm{erase}(e_1\,\mathrm{op}\,e_2) = \mathrm{erase}(e_1)\,\mathrm{op}\,\mathrm{erase}(e_2)$.
6. By (ER-NOT), $\mathrm{erase}(\neg e) = \neg\,\mathrm{erase}(e)$.
7. By (ER-LET), $\mathrm{erase}(\mathrm{let}\,\langle\overrightarrow{x}\rangle = e_1\,\mathrm{in}\,e_2) = \mathrm{let}\,\langle\overrightarrow{x}\rangle = \mathrm{erase}(e_1)\,\mathrm{in}\,\mathrm{erase}(e_2)$.
8. By (ER-IF), $\mathrm{erase}(\mathrm{if}\,e_1\,\mathrm{then}\,e_2\,\mathrm{else}\,e_3) = \mathrm{if}\,\mathrm{erase}(e_1)\,\mathrm{then}\,\mathrm{erase}(e_2)\,\mathrm{else}\,\mathrm{erase}(e_3)$.
9. Since $\mathrm{if}\,B\,\mathrm{then}\,e_1\,\mathrm{else}\,e_2$ is only found inside coersion functions, erase does not apply to it.
10. By (ER-UNION), $\mathrm{erase}(\mathrm{union}(b, \tau_1, \tau_2, e))\,\mathrm{where}\,e \neq c = \mathrm{erase}(e)\,\mathrm{where}\,e \neq c$.
11. By (ER-PACK), $\mathrm{erase}(\mathrm{pack}[\tau_1, e]\,\mathrm{as}\,\exists\alpha : K; B.\tau_2)\,\mathrm{where}\,e \neq c = \mathrm{erase}(e)\,\mathrm{where}\,e \neq c$.
12. By (ER-UNPACK), $\mathrm{erase}(\mathrm{unpack}\,\alpha, x = e_1\,\mathrm{in}\,e_2) = \mathrm{let}\,x = \mathrm{erase}(e_1)\,\mathrm{in}\,\mathrm{erase}(e_2)$.

13. By (ER-CASE), $\text{erase}(\text{case}(b, e))$ where $e \neq c =$ erase(e) where $e \neq c$.

14. By (ER-ROLL), $\text{erase}(\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \dots \tau_n](e))$ where $e \neq c =$ erase(e) where $e \neq c$.

15. By (ER-UNROLL), $\text{erase}(\text{unroll}(e))$ where $e \neq c =$ erase(e) where $e \neq c$.

16. By (ER-FIX), erase(fix $x : \tau \overset{\phi}{:i}$ .v where $i > 0) = \text{fix}\, x.\, \text{erase}(v)$.

17. By (ER-LOAD), $\text{erase}(\text{load}(e_{\text{ptr}}, e_{\text{Has}})) = \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))$.

18. By (ER-STORE), $\text{erase}(\text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}})) = \text{store}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}), \text{erase}(e_v))$.

19. By (ER-DISTINGUISH), $\text{erase}(\text{distinguish}(I_1, I_2, e_1, e_2)) = \text{let}\, \langle x, y \rangle = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \,\text{in}\, \langle\rangle$.

20. By (ER-APPLYEQ), $\text{erase}(\text{apply\_eq}(\tau, e_1, e_2)) = \text{let}\, \langle x, y \rangle = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \,\text{in}\, y$.

21. By (ER-DISCARDFUN), $\text{erase}(\text{discard\_fun}(e)) = \text{let}\, x = \text{erase}(e) \,\text{in}\, \langle\rangle$.

22. By (ER-DEFINEFUN), $\text{erase}(\text{define\_fun}(e, \tau)) = \text{let}\, x = \text{erase}(e) \,\text{in}\, \langle\rangle$.

23. By (ER-INDOMAIN), $\text{erase}(\text{in\_domain}(I_1, I_2, e_1, e_2)) = \text{let}\, \langle x, y \rangle = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \,\text{in}\, \langle\rangle$.

All expressions which are not in c erase to non values (or do not erase).

8.3.3 csize function

Define csize(c) as:

- $\text{csize}(c\tau) = 1 + \text{csize}(c)$
- $\text{csize}(\phi\langle \overrightarrow{c} \rangle : \overrightarrow{\tau} \overset{\phi}{:i})$ where $i > 0 = 1 + \text{csize}(c)$
- $\text{csize}(\text{union}(b, \tau_1, \tau_2, c)) = 1 + \text{csize}(c)$
- $\text{csize}(\text{pack}[\tau_1, c]\,\text{as}\, \exists\alpha : K; B.\tau_2) = 1 + \text{csize}(c)$
- $\text{csize}(\text{case}(b, c)) = 1 + \text{csize}(c)$
- $\text{csize}(\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](c)) = 1 + \text{csize}(c)$
- $\text{csize}(\text{unroll}(c)) = 1 + \text{csize}(c)$
- csize(fix $x : \tau \overset{\phi}{:0}$ .v$) = 1$
- $\text{csize}(\text{coerce}(e)) = 1$
- $\text{csize}(\text{make\_eq}(\tau)) = 1$
- $\text{csize}(\text{new\_fun}(K)) = 1$
- $\text{csize}(\text{discard\_fun}(e)) = 1$
- $\text{csize}(v) = 0$

8.3.4 LEMMA [C-SIZE-DECREASES]

If $C \vdash c : \tau$, where C has an empty Δ and Γ, and (M, c) steps, then there is some c' so that $(M, c) \overset{+}{\mapsto} (M, c')$ and csize$(c) >$ csize(c'). Proof by induction on the possible evaluation rules. In all cases except coerce-congruence, we reach c' by taking only one step. The coerce-congruence case relies on strong normalization, and may take many steps.

1. $\text{coerce}(v) \overset{(E-COERCE)}{\mapsto} v$. $\text{csize}(\text{coerce}(v)) > \text{csize}(v)$.
2. $(\Lambda\alpha : K; B.v)\tau \overset{(E-TAPPTABS)}{\mapsto} [\alpha \mapsto \tau]v$. $\text{csize}((\Lambda\alpha : K; B.v)\tau) = 1$. By Type-Substitution-Value, $[\alpha \mapsto \tau]v$ is a value so $\text{csize}([\alpha \mapsto \tau]v) = 0$.
3. $\text{unroll}(\text{roll}[\tau](v)) \overset{(E-UNROLL)}{\mapsto} v$. $\text{csize}(\text{unroll}(\text{roll}[\tau](v))) > 0$.
4. $\text{fix}\, x : t \overset{\phi}{:0} .v \overset{(E-FIX)}{\mapsto} [x \mapsto \text{fix}\, x : t \overset{\phi}{:0} .v]v$. By Term-Substitution-Value, $[x \mapsto \text{fix}\, x : t \overset{\phi}{:0} .v]v$ is a value. $\text{csize}(\text{fix}\, x : t \overset{\phi}{:0}) = 1 > 0$
5. $\text{case}(b, \text{union}(b, \tau_1, \tau_2, v)) \overset{(E-CASE)}{\mapsto} v$. $\text{csize}(\text{case}(b, \text{union}(b, \tau_1, \tau_2, v))) = 2 > 0$.
6. $\text{make\_eq}(\tau) \overset{(E-MAKEEQ)}{\mapsto} \text{fact}$. $\text{csize}(\text{make\_eq}(\tau)) = 1 > 0$.
7. $\text{new\_fun}(K) \overset{(E-NEWFUN)}{\mapsto} \text{pack}[F^K, \text{fact}]\,\text{as}\, \exists\alpha : \text{int} \mapsto K.\text{FunGen}(\alpha, 0)$, where F^K is fresh. $\text{csize}(\text{new\_fun}(K)) = 1 > 0$.
8. $\text{discard\_fun}(\text{fact}) \overset{(E-DISCARDFUN)}{\mapsto} \cdot\langle\rangle$. $\text{csize}(\text{discard\_fun}(\text{fact})) = 1 > 0$.
9. $c \overset{(congruence\, rule) c = E[c_1]}{\mapsto} c'$. By induction, $(M, c_1) \mapsto (M, c_1')$ and $\text{csize}(c_1) > \text{csize}(c_1')$ so $\text{csize}(E[c_1]) > \text{csize}(E[c_1'])$.
10. $\text{coerce}(e_1) \overset{(congruence\, rule) e = E[e_1]}{\mapsto} \text{coerce}(e_1')$. By inversion, e_1 type-checks in an environment with $limit = i$ for some i. Therefore we can use strong normalization to say $(M, e_1) \overset{*}{\mapsto} (M, v_1)$. By congruence, $(M, \text{coerce}(e_1)) \overset{*}{\mapsto} (M, \text{coerce}(v_1))$. In one additional step, $(M, \text{coerce}(v_1)) \overset{(E-COERCE)}{\mapsto} (M, v_1)$. Let $c' = v_1$, so that $\text{csize}(c) = \text{csize}(\text{coerce}(e_1)) > \text{csize}(v_1) = \text{csize}(c')$.

8.3.5 LEMMA [C-*STEP-VALUE]

If $C \vdash c : \tau$, where C has an empty Δ and Γ, then $(M, c) \overset{*}{\mapsto} (M, v)$.

Proof: By Preservation and Progress, the C-Size-Decreases Lemma will repeatedly apply until a value is reached. This must happen eventually because csize() can only decrease finitely many times and only $\text{csize}(v) = 0$.

8.4 THEOREM [ERASURE-PROGRESS-2]

$$\frac{C \vdash (M, e : \tau)\,\text{where}\, C\,\text{has an empty}\, \Delta\, \text{and}\, \Gamma \quad \text{erase}(e)\,\text{is a value}}{(M, e) \overset{*}{\mapsto} (M, v),\ \text{erase}((M, e)) = \text{erase}((M, v))}$$

If erase(e) is a value, then by Erase-C-Value, $e = c$. By C-*Step-Value, $(M, e) \overset{*}{\mapsto} (M, v)$. By Preservation, $e : \tau$ and $v : \tau$. Either $e = v$ and $\text{erase}((M, e)) = \text{erase}((M, v))$ or $e \neq v$ and $(M, e) \overset{+}{\mapsto} (M, v)$. If $e \neq v$, $e \mapsto e_1 \overset{\text{repeatedly}}{\mapsto} e_n \mapsto v$. By Erasure-Progress-1, $\text{erase}((M, e)) \overset{?}{\mapsto} \text{erase}((M, e_1)) \overset{?,\text{repeatedly}}{\mapsto} \text{erase}((M, e_n)) \overset{?}{\mapsto} \text{erase}((M, v))$. Since erase$(e)$ is a value, $\text{erase}((M, e)) = \text{erase}((M, e_1)) = \text{erase}((M, e_n)) = \text{erase}((M, v))$.

8.4.1 LEMMA [UNTYPED-NON-VALUES-STEP]

If $C \vdash (M, e : \tau)$ where C has an empty Δ and Γ and $\mathrm{erase}(e)$ is a non value, then $\mathrm{erase}((M, e)) \mapsto (L', d')$. Proof by induction on the expressions.

1. Case: $\mathrm{erase}(e)$ is a value.

$$e = i \,|\, b \,|\, \phi\langle\overrightarrow{v}\rangle : \tau \overset{\phi}{:0} \,|\, \lambda x : \tau \overset{\phi,\mathrm{limit}}{\longrightarrow} e_1$$

$$|\, \Lambda\alpha : K; B.v \,|\, \mathrm{fix}\, x : \tau \overset{\phi}{:0} .v \,|\, \mathrm{coerce}(e_1)$$

$$|\, \mathrm{make\_eq}(\tau) \,|\, \mathrm{new\_fun}(K) \,|\, \mathrm{fact}$$

2. Case: $e = x$. This won't typecheck in an empty environment.

3. Case: $e = \mathrm{if}\, B\, \mathrm{then}\, e_1\, \mathrm{else}\, e_2$. This only appears inside coersion functions and will be removed by erase of the coersion function so erase does not apply to this case.

4. Case: $e = e_1\, \tau \,|\, \mathrm{union}(b, \tau_1, \tau_2, e_1) \,|\, \mathrm{pack}[\tau_1, e_1]\, \mathrm{as}\, \exists\alpha : K; B.\tau_2 \,|\, \mathrm{case}(b, e) \,|\, \mathrm{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](e) \,|\, \mathrm{unroll}(e)$.
   The proofs for all of these are similar so we will only show one.
   $e = e_1\, \tau$. By (ER-APPT), $\mathrm{erase}(e_1\, \tau) = \mathrm{erase}(e_1)$. If $e_1 = c$, by C-Erase-Value $\mathrm{erase}(e_1) = u$ so $\mathrm{erase}(e) = u$. If $e_1 \neq c$, e_1 is not a value. By Non-C-Erase-Non-Value $\mathrm{erase}(e_1) \neq u$. By induction, $\mathrm{erase}((M, e_1)) \mapsto (L', d')$. Therefore, $\mathrm{erase}((M, e)) \mapsto (L', d')$.

5. Case: $e = e_1\, e_2$.
   If $e_1 : (\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2)$, $\mathrm{erase}((M, e)) = (\,\mathrm{erase}(M), \mathrm{erase}(e_1)\, \mathrm{erase}(e_2))$ by (ER-M) and (ER-APP).
   If $e_1$is in the set c, then by C-*Step-Value, $(M, e_1) \overset{*}{\mapsto} (M, v_1)$. $(\,\mathrm{erase}(M), \mathrm{erase}(e_1)\, \mathrm{erase}(e_2) \overset{*,(congruence\, rule) E[e]=e\, e_2}{\mapsto}$ $(\,\mathrm{erase}(M), \mathrm{erase}(v_1)\, \mathrm{erase}(e_2))$. By Inversion and Canonical Forms, $v_1 = \lambda x : \tau_1 \overset{\phi,\infty}{\longrightarrow} e_{11}$. $\mathrm{erase}(v_1) = \lambda x \longrightarrow \mathrm{erase}(e_{11})$. If $\mathrm{erase}(e_2)$ is a value, then $(\,\mathrm{erase}(M), (\lambda x \longrightarrow \mathrm{erase}(e_{11}))\, \mathrm{erase}(e_2)) \overset{(U-ABSAPP)}{\mapsto}$ $(\,\mathrm{erase}(M), [x \mapsto \mathrm{erase}(e_2)]\, \mathrm{erase}(e_{11}))$. If $\mathrm{erase}(e_2)$ is not a value, then by induction $\mathrm{erase}((M, e_2)) \mapsto (L', d_2)$. $(\,\mathrm{erase}(M), \mathrm{erase}(v_1)\, \mathrm{erase}(e_2)) \overset{(congruence\, rule) E[e]=v\, e}{\mapsto} (L', \mathrm{erase}(v_1)\, d_2)$.
   If e_1 is not in the set c, then by Not-C-Erase-Not-Value $\mathrm{erase}(e_1)$ is a non value. By induction $\mathrm{erase}((M, e_1)) \mapsto (L', d_1)$. $(\,\mathrm{erase}(M), \mathrm{erase}(e_1)\, \mathrm{erase}(e_2)) \overset{(congruence\, rule) E[e]=e\, e_2}{\mapsto} (L', d_1\, \mathrm{erase}(e_2))$.

   If $e_1 : (\tau_1 \overset{\phi,I}{\mapsto} \tau_2)$, $\mathrm{erase}((M, e)) = (\,\mathrm{erase}(M), \mathrm{let}\, x = \langle\,\mathrm{erase}(e_1), \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle)$ by (ER-M) and (ER-APPI).
   If $\mathrm{erase}(e_1)$ is a value, if $\mathrm{erase}(e_2)$ is a value, then $\mathrm{let}\, x = \langle\,\mathrm{erase}(e_1), \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle \overset{(E-LET)}{\mapsto} \langle\rangle$. If $\mathrm{erase}(e_2)$ is not a value, then by induction $\mathrm{erase}((M, e_2)) \mapsto (L', d_2)$. $(\,\mathrm{erase}(M), \mathrm{let}\, x = \langle\,\mathrm{erase}(e_1), \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle)$ $\overset{(congruence\, rule) E[e]=\phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle}{\mapsto} (L', \mathrm{let}\, x = \langle\,\mathrm{erase}(e_1), d_2\rangle\, \mathrm{in}\, \langle\rangle)$.
   If $\mathrm{erase}(e_1)$ is not a value, then by induction $\mathrm{erase}((M, e_1)) \mapsto (L', d_1)$. $(\,\mathrm{erase}(M), \mathrm{let}\, x = \langle\,\mathrm{erase}(e_1), \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle)$ $\overset{(congruence\, rule) E[e]=\phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle}{\mapsto} (L', \mathrm{let}\, x = \langle d_1, \mathrm{erase}(e_2)\rangle\, \mathrm{in}\, \langle\rangle)$.

6. Case: $e = \phi\langle\overrightarrow{e_1}\rangle$. If $e = \phi\langle\overrightarrow{e_1}\rangle : \overrightarrow{\tau_1} \overset{\phi}{:i}$ where $i > 0$, by (ER-TUPLE) and (ER-M) $\mathrm{erase}((M, e)) =$ $(\,\mathrm{erase}(M), \langle\,\mathrm{erase}(e_{11}), \ldots, \mathrm{erase}(e_{1n})\rangle)$. If $\mathrm{erase}(e)$ is a non value then some $\mathrm{erase}(e_{1i})$ is a non value. By induction, $\mathrm{erase}((M, e_{1i})) \mapsto (L', d_{1i})$. $(\,\mathrm{erase}(M), \langle\,\mathrm{erase}(e_{11}), \ldots, \mathrm{erase}(e_{1n})\rangle) \overset{(congruence\, rule) E[e]=\phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle}{\mapsto}$ $(L', \langle\,\mathrm{erase}(e_{11}), \ldots, d_{1i}, \ldots, \mathrm{erase}(e_{1n})\rangle)$. If $e = \phi\langle\overrightarrow{e_1}\rangle : \overrightarrow{\tau_1} \overset{\phi}{:0}$, by (ER-TUPLEe0) and (ER-M) $\mathrm{erase}((M, e)) =$ $(\,\mathrm{erase}(M), \mathrm{let}\, x = \phi\langle\overrightarrow{\mathrm{erase}(e_1)}\rangle\, \mathrm{in}\, \langle\rangle)$ where some $\mathrm{erase}(e_{1i})$ is not a value. By induction $\mathrm{erase}((M, e_{1i})) \mapsto$ (L', d_{1i}). $(\,\mathrm{erase}(M), \langle\,\mathrm{erase}(e_{11}), \ldots, \mathrm{erase}(e_{1n})\rangle) \overset{(congruence\, rule) E[e]=\phi\langle v_1, \cdots, v_{k-1}, e, e_{k+1}, \cdots, e_n\rangle}{\mapsto}$ $(L', \langle\,\mathrm{erase}(e_{11}), \ldots, d_{1i}, \ldots, \mathrm{erase}(e_{1n})\rangle)$.

7. Case: $e = e_1\, \mathrm{op}\, e_2 \,|\, \neg e_1 \,|\, \mathrm{if}\, e_1\, \mathrm{then}\, e_2\, \mathrm{else}\, e_3 \,|\, \mathrm{unpack}\, \alpha, x = e_1\, \mathrm{in}\, e_2$
   $|\, \mathrm{load}(e_{\mathrm{ptr}}, e_{\mathrm{Has}}) \,|\, \mathrm{store}(e_{\mathrm{ptr}}, e_{\mathrm{Has}}, e_{\mathrm{v}}) \,|\, \mathrm{distinguish}(I_1, I_2, e_1, e_2) \,|\, \mathrm{apply\_eq}(\tau, e_1, e_2) \,|\, \mathrm{discard\_fun}(e_1) \,|\, \mathrm{define\_fun}(e_1, \tau)$
   $|\, \mathrm{in\_domain}(I_1, I_2, e_1, e_2)$.

Proof :
$e = \text{if}\, e_1 \,\text{then}\, e_2 \,\text{else}\, e_3$. By (ER-IF) and (ER-M), $\text{erase}((M,e)) = (\,\text{erase}(M), \text{if erase}(e_1)\,\text{then erase}(e_2)\,\text{else erase}(e_3))$. If e_1 is in the set c, then by (C-*Step-Value) $(M, e_1) \overset{*}{\mapsto} (M, v_1)$. Since $v_1 : Bool(B)$, $v_1 = b$ by Canonical Forms. So, $\text{erase}(v_1) = b$ by (ER-b). If $b = true$ then $(\,\text{erase}(M), \text{if erase}(e_1)\,\text{then erase}(e_2)\,\text{else erase}(e_3)) \overset{(E-IF1)}{\mapsto} (\,\text{erase}(M), \text{erase}(e_2))$. If $b = false$ then $(\,\text{erase}(M), \text{if erase}(e_1)\,\text{then erase}(e_2)\,\text{else erase}(e_3)) \overset{(E-IF2)}{\mapsto} (\,\text{erase}(M), \text{erase}(e_3))$. If e_1 is not in the set c, then $\text{erase}(e_1)$ is not a value and $\text{erase}((M, e_1)) \mapsto (L', d')$ by induction. Therefore $(\,\text{erase}(M), \text{if erase}(e_1)\,\text{then erase}(e_2)\,\text{else erase}(e_3)) \overset{(congruence\,rule) E[e]=\text{if}\, e \,\text{then}\, e_2 \,\text{else}\, e_3}{\mapsto} (L', \text{if}\, d' \,\text{then erase}(e_2)\,\text{else erase}(e_3))$.

8. Case: $e = \text{fix}\, x : \tau :\overset{\phi}{i}$ .v where $i > 0$. By (ER-FIX) $\text{erase}(e) = \text{fix}\, x.\, \text{erase}(v)$. $\text{fix}\, x.\, \text{erase}(v) \overset{(U-FIX)}{\mapsto} [x \mapsto \text{fix}\, x.\, \text{erase}(v)]\, \text{erase}(v)$.

9. Case: $e = \text{let}\, \langle \overrightarrow{x} \rangle = e_1 \,\text{in}\, e_2$. By (ER-LET) $\text{erase}((M, e)) = (\,\text{erase}(M), \text{let}\, \langle \overrightarrow{x} \rangle = \text{erase}(e_1)\,\text{in erase}(e_2)$. If e_1 is in the set c, by (C-*Step-Value) $(M, e_1) \overset{*}{\mapsto} (M, v_1)$. Since e is well-typed, $e_1 : \langle \overrightarrow{t} \rangle$ and we know $\overrightarrow{x : \tau}$ by (T-LET). By Canonical Forms, $v_1 = \langle \overrightarrow{v} \rangle$and $\overrightarrow{x : \tau}$ means v_1 has the same number of elements as $\langle \overrightarrow{x} \rangle$. $\text{erase}(v_1) = \langle \overrightarrow{\text{erase}(v)} \rangle$ and $(\,\text{erase}(M), \text{let}\, \langle \overrightarrow{x} \rangle = \langle \overrightarrow{\text{erase}(v)} \rangle \,\text{in erase}(e_2) \overset{(E-LET)}{\mapsto} [\langle \overrightarrow{x} \rangle \mapsto \langle \overrightarrow{\text{erase}(v)} \rangle]\, \text{erase}(e_2)$.

8.5 THEOREM [ERASURE-PROGRESS-3]

$$\frac{C \vdash (M, e : \tau)\,\text{where}\, C \,\text{has an empty}\, \Delta \,\text{and}\, \Gamma \quad \text{erase}((M, e)) \mapsto (L', d')}{(M, e) \overset{+}{\mapsto} (M', e'),\ \text{erase}((M', e')) = (L', d')}$$

If erase(e) is not a value then, by Untyped-Non-Values-Step, $\text{erase}((M, e))$ evaluates to (L', d'). Additionally, (M, e) evaluates in one or more steps to (M', e'), and $\text{erase}((M', e'))$= (L', d'). Proof by induction on the expression rules. There are several cases for this proof. For each, we list the rules which follow the case and show one proof. The other proofs in each case can be obtained using a similar proof to the example.

8.5.1 Case: The values:

$$e = i \,|\, b \,|\, \Lambda\alpha : K; B.v \,|\, \text{pack}[\tau_1, v]\,\text{as}\, \exists\alpha : K; B.\tau_2 \,|\, \lambda x : \tau \overset{\phi, limit}{\longrightarrow} e_1$$
$$|\, \text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \ldots \tau_n](v) \,|\, \phi\langle \overrightarrow{v} \rangle \,|\, \text{union}(b, \tau_1, \tau_2, v) \,|\, \text{fact}$$

By Value-Erase-Value, $\text{erase}(v) = u$ and Erasure-Progress-3 does not apply.

8.5.2 Case: Most of the non values:

$$e = e_1\, \tau \,|\, \neg e_1 \,|\, \text{union}(b, \tau_1, \tau_2, e_1) \,|\, \text{pack}[\tau_1, e_1]\,\text{as}\, \exists\alpha : K; B.\tau_2$$
$$|\, \text{case}(b, e_1) \,|\, \text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](e_1) \,|\, \text{unroll}(e_1)$$

The proofs for all of these use the same format with the Congruence rule and induction steps that apply to each. Let $e = e_1\, \tau$.
$\text{erase}(e) = \text{erase}(e_1\, \tau) = \text{erase}(e_1)$ by (ER-APPT).

- If $\text{erase}(e_1)$ is a value, then erase(e) is a value and Erasure-Progress-3 does not apply.
- If $\text{erase}(e_1)$ is a non value, then erase(e) is a non value. $\text{erase}((M, e_1)) \mapsto (L', d_1')$ by Untyped-Non-Values-Step and $(M, e_1) \overset{+}{\mapsto} (M', e_1')$, $\text{erase}((M', e_1')) = (L', d_1')$ by induction. $(M, e) = (M, e_1\, \tau) \overset{+,(congruence\, rule) E[e]=e\,\tau}{\mapsto} (M', e_1'\, \tau) = (M', e')$ so $(M, e) \overset{+}{\mapsto} (M', e')$. $\text{erase}((M, e)) = (\,\text{erase}(M), \text{erase}(e))$= $(\,\text{erase}(M), \text{erase}(e_1)) \mapsto (L', d_1') = (L', d')$ so $\text{erase}((M, e)) \mapsto (L', d')$. $\text{erase}((M', e')) = (\,\text{erase}(M'), \text{erase}(e')) = (\,\text{erase}(M'), \text{erase}(e_1'\, \tau)) = (\,\text{erase}(M'), \text{erase}(e_1'))$= $(L', d_1') = (L', d')$ by (ER-M) and (ER-APPT) so $\text{erase}((M', e')) = (L', d')$.

8.5.3 Case: Let $e = \textbf{unpack}\,\alpha, x = e_1\,\textbf{in}\,e_2$.

$\text{erase}(e) = \text{erase}(\text{unpack}\,\alpha, x = e_1 \text{ in } e_2) = \text{let}\,x = \text{erase}(e_1) \text{ in erase}(e_2)$by (ER-UNPACK).

- Let $\text{erase}(e_1)$ be a value.
  $(M, e_1) \overset{*}{\mapsto} (M, v_1)$, and $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$ by Erasure-Progress-2. $(M, e) = (M, \text{unpack}\,\alpha, x = e_1 \text{ in } e_2)$ $\overset{*,(congruence\,rule)E[e]=\text{unpack}\,\alpha,x=e\text{ in }e_2}{\mapsto}$ $(M, \text{unpack}\,\alpha, x = v_1 \text{ in } e_2)$. By Canonical Forms, $v_1 = \text{pack}[\tau_1, v_{11}] \text{ as } \exists\alpha_1 : K; B.\tau_2$. So $(M, \text{unpack}\,\alpha, x = v_1 \text{ in } e_2) = (M, \text{unpack}\,\alpha, x = \text{pack}[\tau_1, v_{11}] \text{ as } \exists\alpha_1 : K; B.\tau_2 \text{ in } e_2) \overset{(E-UNPACK)}{\mapsto}$ $(M, [\alpha \mapsto \tau_1, x \mapsto v_{11}]e_2) = (M, e')$. $\text{erase}((M, e)) = (\text{erase}(M), \text{let}\,x = \text{erase}(e_1) \text{ in erase}(e_2)) \overset{(E-LET)}{\mapsto}$ $(\text{erase}(M), [x \mapsto \text{erase}(e_1)]\,\text{erase}(e_2)) = (L', d')$. $\text{erase}(v_1) = \text{erase}(\text{pack}[\tau_1, v_{11}] \text{ as } \exists\alpha_1 : K; B.\tau_2) = \text{erase}(v_{11})$ by (ER-PACK). $\text{erase}((M, e')) = \text{erase}((M, [\alpha \mapsto \tau_1, x \mapsto v_{11}]e_2)) = (\text{erase}(M), \text{erase}([\alpha \mapsto \tau_1, x \mapsto v_{11}]e_2))=$ $(\text{erase}(M), [x \mapsto \text{erase}(v_{11})]\,\text{erase}(e_2))$ by (ER-M), Erase-Term-Substitution and Erase-Type-Substitution. $(\text{erase}(M), [x \mapsto \text{erase}(v_{11})]\,\text{erase}(e_2))= (\text{erase}(M), [x \mapsto \text{erase}(v_1)]\,\text{erase}(e_2))= (\text{erase}(M), [x \mapsto \text{erase}(e_1)]\,\text{erase}(e_2))$ $= (L', d')$.

- Let $\text{erase}(e_1)$be a non value.
  $\text{erase}((M, e_1)) \mapsto (L', d_1)$ by Untyped-Non-Values-Step and $(M, e_1) \overset{+}{\mapsto} (M', e_1')$, $\text{erase}((M', e_1')) = (L', d_1)$ by induction. $(M, e) = (M, \text{unpack}\,\alpha, x = e_1 \text{ in } e_2)$ $\overset{+,(congruence\,rule)E[e]=\text{unpack}\,\alpha,x=e\text{ in }e_2}{\mapsto}$ $(M', \text{unpack}\,\alpha, x = e_1' \text{ in } e_2) = (M', e')$. $\text{erase}((M, e)) = (\text{erase}(M), \text{let}\,x = \text{erase}(e_1) \text{ in erase}(e_2))$ $\overset{(congruence\,rule)E[e]=\text{let}\,x=e\text{ in }e_2}{\mapsto}$ $(\text{erase}(M'), \text{let}\,x = d_1 \text{ in erase}(e_2)) = (L', d')$. $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\text{unpack}\,\alpha, x = e_1' \text{ in } e_2)) =$ $(\text{erase}(M'), \text{let}\,x = \text{erase}(e_1') \text{ in erase}(e_2))$ by (ER-M) and (ER-UNPACK). $(\text{erase}(M'), \text{let}\,x = \text{erase}(e_1') \text{ in erase}(e_2)) =$ $(L', \text{let}\,x = d_1 \text{ in erase}(e_2)) = (L', d')$.

8.5.4 Case: Let $e = x$.

$\text{erase}(e) = \text{erase}(x) = x$ by (ER-x). This is not a value but, x does not progress to another expression so Erasure-Progress-3 does not apply.

8.5.5 Case: Let $e = e_1\,e_2$.

Since e is well typed, $e_1 : (\tau_1 \overset{\phi,\text{limit}}{\mapsto} \tau_2)$ has limit I or ∞.

- Let limit $= \infty$ so, $e_1 : (\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2)$.
  $\text{erase}(e) = \text{erase}(e_1 : (\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2)\,e_2) = \text{erase}(e_1)\,\text{erase}(e_2)$ by (ER-APP).

- If $\text{erase}(e_1)$ is not a value, $\text{erase}((M, e_1)) \mapsto (L', d_1)$ by Untyped-Non-Values-Step and $(M, e_1) \overset{+}{\mapsto} (M', e_1')$, $\text{erase}((M', e_1')) = (L', d_1)$ by induction. $\text{erase}((M, e)) = (\text{erase}(M), \text{erase}(e_1)\,\text{erase}(e_2)) \overset{(congruence\,rule)E[e]=e\,e_2}{\mapsto}$ $(\text{erase}(M'), d_1\,\text{erase}(e_2))$
  $= (L', d')$. $(M, e) = (M, e_1\,e_2) \overset{+,(congruence\,rule)E[e]=e\,e_2}{\mapsto} (M', e_1'\,e_2) = (M', e')$. $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(e_1' :$ $(\tau_1 \overset{\phi,\infty}{\mapsto} \tau_2)\,e_2)) = (\text{erase}(M'), \text{erase}(e_1')\,\text{erase}(e_2))$ by (ER-APP). This equals $(L', d_1\,\text{erase}(e_2)) = (L', d')$.

- If $\text{erase}(e_1)$ is a value and $\text{erase}(e_2)$ is not a value, then by C-Erase-Value, $e = c$ and by C-*Step-Value, $(M, e_1) \overset{*}{\mapsto} (M, v_1)$. Then $\text{erase}(e_2)$ steps, e_2 steps, and e steps by congruence, using the same argument as in the previous case.

- If $\text{erase}(e_1)$ is a value and $\text{erase}(e_2)$ is a value, then by C-Erase-Value, $e = c$ and by C-*Step-Value, $(M, e_1) \overset{*}{\mapsto}$ (M, v). By Inversion and Canonical Forms, $v = \lambda x : \tau \overset{\phi,\text{limit}}{\longrightarrow} e_{11}$. By Erasure-Progress-2 $\text{erase}(e_1) = \text{erase}(v)$ so $\text{erase}(e_1) = \text{erase}(v) = \lambda x \longrightarrow \text{erase}(e_{11})$. $\text{erase}(e) = (\lambda x \longrightarrow \text{erase}(e_{11}))\,\text{erase}(e_2) \overset{(U-ABSAPP)}{\mapsto} [x \mapsto$ $\text{erase}(e_2)]\,\text{erase}(e_{11}) = d'$. $e = (\lambda x : \tau_1 \overset{\phi,\infty}{\longrightarrow} e_{11})\,e_2 \overset{(E-ABSAPP2)}{\mapsto} [x \mapsto e_2]e_{11} = e'$. $\text{erase}(e') = \text{erase}([x \mapsto$ $e_2]e_{11}) = [x \mapsto \text{erase}(e_2)]\,\text{erase}(e_{11}) = d'$ by Erase-Term-Substitution.

- Let limit $= I$ so, $e_1 : (\tau_1 \overset{\phi,I}{\mapsto} \tau_2)$.
  $\text{erase}(e) = \text{erase}(e_1 : (\tau_1 \overset{\phi,I}{\mapsto} \tau_2)\, e_2) = \text{let}\, x = \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle$ by (ER-APPI).

- Let $\text{erase}(e_1)$ and $\text{erase}(e_2)$ be values.
  $(M, e_1) \overset{*}{\mapsto} (M, v_1)$, and $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$ by Erasure-Progress-2. $(M, e_2) \overset{*}{\mapsto} (M, v_2)$, and $\text{erase}((M, e_2)) = \text{erase}((M, v_2))$ by Erasure-Progress-2.
  $\text{erase}((M, e)) = (\,\text{erase}(M), \text{let}\, x = \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle) \overset{(E-LET)}{\mapsto} (\,\text{erase}(M), \langle\rangle) = (L', d')$. $(L' = \text{erase}(M))$. $(M, e) = (M, e_1\, e_2) \overset{*,(congruence\, rule) E[e]=e\, e_2}{\mapsto} (M, v_1\, e_2)$ $\overset{*,(congruence\, rule) E[e]=v\, e}{\mapsto} (M, v_1\, v_2)$. By Inversion and Canonical Forms, $e_1 = \lambda x : \tau \xrightarrow{\phi,I} e_{11}$. $(M, v_1\, v_2) = (M, (\lambda x : \tau \xrightarrow{\phi,I} e_{11}) v_2) \overset{(E-ABSAPP1)}{\mapsto} (M, \text{coerce}([x \mapsto v_2] e_{11})) = (M, e')$.
  $\text{erase}((M, e')) = (\,\text{erase}(M)\, \text{erase}(\text{coerce}([x \mapsto v_2] e_{11})))= (\,\text{erase}(M), \langle\rangle) = (L', d')$ by (ER-COERCE).

- Let $\text{erase}(e_1)$ be a value and $\text{erase}(e_2)$ be a non value.
  $(M, e_1) \overset{*}{\mapsto} (M, v_1)$, and $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$ by Erasure-Progress-2. $\text{erase}((M, e_2)) \mapsto (L'_2, d_2)$ by Untyped-Non-Values-Step and $(M, e_2) \overset{+}{\mapsto} (M'_2, e'_2)$, $\text{erase}((M'_2, e'_2)) = (L'_2, d_2)$ by induction. $\text{erase}((M, e)) = (\,\text{erase}(M), \text{let}\, x = \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle)$ by (ER-M). Since $(\,\text{erase}(M), \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle)$ $\overset{(congruence\, rule) E[e]=\phi\langle v_1,\cdots,v_{k-1},e,e_{k+1},\cdots,e_n\rangle}{\mapsto} (L'_2, \langle\, \text{erase}(e_1), d_2\rangle)$, $(\,\text{erase}(M), \text{let}\, x = \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle)$ $\overset{(congruence\, rule) E[e]=\text{let}\, \overrightarrow{x}=e\, \text{in}\, e_2}{\mapsto} (L'_2, \text{let}\, x = \langle\, \text{erase}(e_1), d_2\rangle\, \text{in}\, \langle\rangle) = (L', d')$. $(L' = L'_2)$.
  $(M, e) = (M, e_1\, e_2) \overset{*,(congruence\, rule) E[e]=e\, e_2}{\mapsto} (M, v_1\, e_2)$ $\overset{+,(congruence\, rule) E[e]=v\, e}{\mapsto} (M'_2, v_1\, e'_2) = (M', e')$. $(M' = M'_2)$. By Inversion and Canonical Forms, $v_1 = \lambda x : \tau \xrightarrow{\phi,I} e_{11}$. $\text{erase}((M', e')) = (\,\text{erase}(M'),\, \text{erase}(v_1\, e'_2))= (\,\text{erase}(M'),\, \text{erase}((\lambda x : \tau \xrightarrow{\phi,I} e_{11}) e'_2))= (\,\text{erase}(M'), \text{let}\, x = \langle\, \text{erase}(v_1),\, \text{erase}(e'_2)\rangle\, \text{in}\, \langle\rangle)$ by (ER-APPI). This equals $(L'_2, \text{let}\, x = \langle\, \text{erase}(e_1), d_2\rangle\, \text{in}\, \langle\rangle) = (L', d')$.

- Let $\text{erase}(e_1)$ be a non value.
  $\text{erase}((M, e_1)) \mapsto (L', d_1)$ by Untyped-Non-Values-Step and $(M, e_1) \overset{+}{\mapsto} (M', e'_1)$, $\text{erase}((M', e'_1)) = (L', d_1)$ by induction. $\text{erase}((M, e)) = (\,\text{erase}(M), \text{let}\, x = \text{erase}(e_1),\, \text{erase}(e_2)\, \text{in}\, \langle\rangle)$. Since $(\,\text{erase}(M), \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle)$ $\overset{(congruence\, rule) E[e]=\phi\langle v_1,\cdots,v_{k-1},e,e_{k+1},\cdots,e_n\rangle}{\mapsto} (\,\text{erase}(M'), \langle d_1,\, \text{erase}(e_2)\rangle)$, $(\,\text{erase}(M), \text{let}\, x = \langle\, \text{erase}(e_1),\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle)$ $\overset{(congruence\, rule) E[e]=\text{let}\, \overrightarrow{x}=e\, \text{in}\, e_2}{\mapsto} (L', \text{let}\, x = \langle d_1,\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle) = (L', d')$. $(M, e) = (M, e_1\, e_2) \overset{+,(congruence\, rule) E[e]=e\, e_2}{\mapsto}$ $(M', e'_1\, e_2) = (M', e')$. By Preservation, $e'_1 : (\tau_1 \overset{\phi,I}{\mapsto} \tau_2)$. $\text{erase}((M', e')) = (\,\text{erase}(M'),\, \text{erase}(e'_1\, e_2))=$ $(\,\text{erase}(M'), \text{let}\, x = \langle\, \text{erase}(e'_1),\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle)= (L', \text{let}\, x = \langle d_1,\, \text{erase}(e_2)\rangle\, \text{in}\, \langle\rangle) = (L', d')$.

8.5.6 Case: Let $e = \phi\langle \overrightarrow{e} \rangle$.

- Let $e = \phi\langle e_1, e_2, \ldots, e_n\rangle : t \overset{\phi}{:} i$ where $i > 0$
  $\text{erase}(e) = \text{erase}(\phi\langle e_1, e_2, \ldots, e_n\rangle : t \overset{\phi}{:} i) = \langle\, \text{erase}(e_1), \ldots,\, \text{erase}(e_n)\rangle$ by (ER-TUPLE). If this is a value then Erasure-Progress-3 does not apply. If this is not a value, then for some k, $\text{erase}((M, e_k)) \mapsto (L', d_k)$ by Untyped-Non-Values-Step and $(M, e_k) \overset{+}{\mapsto} (M', e'_k)$, $\text{erase}((M', e'_k)) = (L', d_k)$ by induction.
  $\text{erase}((M, e)) = (\,\text{erase}(M), \langle\, \text{erase}(e_1), \ldots,\, \text{erase}(e_n)\rangle)$
  $\overset{(congruence\, rule) E[e]=\phi\langle v_1,\cdots,v_{k-1},e,e_{k+1},\cdots,e_n\rangle}{\mapsto} (L', \langle\, \text{erase}(e_1), \ldots, d_k, \ldots,\, \text{erase}(e_n)\rangle) = (L', d')$.
  $(M, e) = (M, \phi\langle e_1, \ldots, e_k, \ldots, e_n\rangle : t \overset{\phi}{:} i) \overset{+,(congruence\, rule) E[e]=\phi\langle v_1,\cdots,v_{k-1},e,e_{k+1},\cdots,e_n\rangle}{\mapsto} (M', \phi\langle e_1, \ldots, e'_k, \ldots, e_n\rangle : t \overset{\phi}{:} i) = (M', e')$.
  $\text{erase}((M', e')) = (\,\text{erase}(M'),\, \text{erase}(\phi\langle e_1, \ldots, e_k, \ldots, e_n\rangle : t \overset{\phi}{:} i))= (\,\text{erase}(M'), \langle\, \text{erase}(e_1), \ldots,\, \text{erase}(e_k), \ldots,\, \text{erase}(e_n)\rangle)$ $= (L', \langle\, \text{erase}(e_1), \ldots, d_k, \ldots,\, \text{erase}(e_n)\rangle) = (L', d')$ by (ER-M) and (ER-TUPLE).

- Let $e = \phi\langle e_1, e_2, \ldots, e_n\rangle : t \overset{\phi}{:} 0$
  $\text{erase}(e) = \text{erase}(\phi\langle e_1, e_2, \ldots, e_n\rangle : t \overset{\phi}{:} 0) = \text{let}\, x = \langle\, \text{erase}(e_1), \ldots,\, \text{erase}(e_n)\rangle\, \text{in}\, \langle\rangle$ by (ER-TUPLEe0). If

this is a value then Erasure-Progress-3 does not apply. If this is not a value, then there is some k so that $\mathrm{erase}((M,e_k)) \mapsto (L',d_k)$ by Untyped-Non-Values-Step and $(M,e_k) \overset{+}{\mapsto} (M',e'_k)$, $\mathrm{erase}((M',e'_k)) = (L',d_k)$ by induction. $\mathrm{erase}((M,e)) = (\mathrm{erase}(M), \mathrm{let}\, x = \langle \mathrm{erase}(e_1),\ldots,\mathrm{erase}(e_n)\rangle \,\mathrm{in}\, \langle\rangle)$. Since $(\mathrm{erase}(M), \langle \mathrm{erase}(e_1),\ldots,\mathrm{erase}(e_n)\rangle) \overset{(congruence\, rule) E[e]=\phi\langle v_1,\cdots,v_{k-1},e,e_{k+1},\cdots,e_n\rangle}{\mapsto} (L', \langle \mathrm{erase}(e_1),\ldots,d_k,\ldots,\mathrm{erase}(e_n)\rangle)$ $(\mathrm{erase}(M), \mathrm{let}\, x = \langle \mathrm{erase}(e_1),\ldots,\mathrm{erase}(e_n)\rangle \,\mathrm{in}\, \langle\rangle) \overset{(congruence\, rule) E[e]=\mathrm{let}\ \overrightarrow{x}=e \,\mathrm{in}\, e_2}{\mapsto}$ $(L', \mathrm{let}\, x = \langle \mathrm{erase}(e_2),\ldots,d_k,\ldots,\mathrm{erase}(e_n)\rangle \,\mathrm{in}\, \langle\rangle) = (L',d')$. $(M,e) = (M, \phi\langle e_1,e_2,\ldots,e_n\rangle : t \overset{\phi}{:}0)$ $\overset{+,(congruence\, rule) E[e]=\phi\langle v_1,\cdots,v_{k-1},e,e_{k+1},\cdots,e_n\rangle}{\mapsto} (M', \phi\langle e_1,\ldots,e'_k,\ldots,e_n\rangle : t \overset{\phi}{:}0) = (M',e')$. $\mathrm{erase}((M',e'))$ $= (\mathrm{erase}(M'), \mathrm{erase}(\phi\langle e_1,\ldots,e'_k,\ldots,e_n\rangle : t \overset{\phi}{:}0)) = (\mathrm{erase}(M'), \mathrm{let}\, x = \langle \mathrm{erase}(e_1),\ldots,\mathrm{erase}(e'_k),\ldots,\mathrm{erase}(e_n)\rangle \,\mathrm{in}\, \langle\rangle) =$ $(L', \mathrm{let}\, x = \langle \mathrm{erase}(e_1),\ldots,d_k,\ldots,\mathrm{erase}(e_n)\rangle \,\mathrm{in}\, \langle\rangle) = (L',d')$ by (ER-M) and (ER-TUPLEe0).

8.5.7 Case: Let $e = e_1 \,\mathbf{op}\, e_2 \,|\, \mathbf{let}\, x = e_1 \,\mathbf{in}\, e_2$.

These cases are similar so we only show one.

Let $e = e_1 \,\mathrm{op}\, e_2$.

$\mathrm{erase}(e) = \mathrm{erase}(e_1 \,\mathrm{op}\, e_2) = \mathrm{erase}(e_1) \,\mathrm{op}\, \mathrm{erase}(e_2)$ by (ER-OP).

- Let $\mathrm{erase}(e_1)$ and $\mathrm{erase}(e_2)$ be values.
  $(M,e_1) \overset{*}{\mapsto} (M,v_1)$ and $\mathrm{erase}((M,e_1)) = \mathrm{erase}((M,v_1))$ by Erasure-Progress-2. Similarly, $(M,e_2) \overset{*}{\mapsto} (M,v_2)$ and $\mathrm{erase}((M,e_2)) = \mathrm{erase}((M,v_2))$ by Erasure-Progress-2. $(M,e) = (M, e_1 \,\mathrm{op}\, e_2) \overset{*,(congruence\, rule) E[e]=e \,\mathrm{op}\, e_2}{\mapsto}$ $(M, v_1 \,\mathrm{op}\, e_2)$ and $(M, v_1 \,\mathrm{op}\, e_2) \overset{*,(congruence\, rule) E[e]=v \,\mathrm{op}\, e}{\mapsto} (M, v_1 \,\mathrm{op}\, v_2)$. By canonical forms, v_1 and v_2 are both integers for an integer op, or booleans for a boolean op, so that $\mathrm{simplify}(v_1 \,\mathrm{op}\, v_2)$ exists and $(M, v_1 \,\mathrm{op}\, v_2) \overset{(E-SIMPLIFY1)}{\mapsto} (M, \mathrm{simplify}(v_1 \,\mathrm{op}\, v_2)) = (M,e')$.
  $\mathrm{erase}((M,e)) = (\mathrm{erase}(M), \mathrm{erase}(e_1) \,\mathrm{op}\, \mathrm{erase}(e_2)) \overset{(E-SIMPLIFY1)}{\mapsto} ((\mathrm{erase}(M), \mathrm{simplify}(\mathrm{erase}(e_1) \,\mathrm{op}\, \mathrm{erase}(e_2))) =$ (L',d').
  $(\mathrm{erase}(M) = L')$. $\mathrm{erase}((M,e')) = (\mathrm{erase}(M), \mathrm{erase}(\mathrm{simplify}(v_1 \,\mathrm{op}\, v_2)))$
  $= (\mathrm{erase}(M), \mathrm{simplify}(\mathrm{erase}(v_1) \,\mathrm{op}\, \mathrm{erase}(v_2)))$ by (ER-M) and Erase-Simplify. This equals $(\mathrm{erase}(M), \mathrm{simplify}(\mathrm{erase}(v_1) \,\mathrm{op}\, \mathrm{erase}(e_2))) = (\mathrm{erase}(M), \mathrm{simplify}(\mathrm{erase}(e_1) \,\mathrm{op}\, \mathrm{erase}(e_2))) = (L',d')$.

- Let $\mathrm{erase}(e_1)$ be a value and $\mathrm{erase}(e_2)$ be a non value.
  $(M,e_1) \overset{*}{\mapsto} (M,v_1)$ and $\mathrm{erase}((M,e_1)) = \mathrm{erase}((M,v_1))$ by Erasure-Progress-2. $\mathrm{erase}((M,e_2)) \mapsto (L'_2,d_2)$ by Untyped-Non-Values-Step and $(M,e_2) \overset{+}{\mapsto} (M'_2,e'_2)$ and $\mathrm{erase}((M'_2,e'_2)) = (L'_2,d_2)$ by induction. $(M,e) =$ $(M, e_1 \,\mathrm{op}\, e_2)$
  $\overset{*,(congruence\, rule) E[e]=e \,\mathrm{op}\, e_2}{\mapsto} (M, v_1 \,\mathrm{op}\, e_2)$ and $(M, v_1 \,\mathrm{op}\, e_2) \overset{+,(congruence\, rule) E[e]=v \,\mathrm{op}\, e}{\mapsto} (M'_2, v_1 \,\mathrm{op}\, e'_2) = (M',e')$. $(M'_2 = M')$. $\mathrm{erase}((M,e)) = (\mathrm{erase}(M), \mathrm{erase}(e_1) \,\mathrm{op}\, \mathrm{erase}(e_2)) \overset{(congruence\, rule) E[e]=v \,\mathrm{op}\, e}{\mapsto} (L'_2, \mathrm{erase}(e_1) \,\mathrm{op}\, d_2) =$ (L',d'). $(L'_2 = L')$. $\mathrm{erase}((M',e')) = (\mathrm{erase}(M'), \mathrm{erase}(v_1 \,\mathrm{op}\, e'_2)) = (\mathrm{erase}(M'), \mathrm{erase}(v_1) \,\mathrm{op}\, \mathrm{erase}(e'_2))$ by (ER-M) and (ER-OP). This equals $(\mathrm{erase}(M'_2), \mathrm{erase}(v_1) \,\mathrm{op}\, \mathrm{erase}(e'_2)) = (L'_2, \mathrm{erase}(e_1) \,\mathrm{op}\, d_2) = (L',d')$.

- Let $\mathrm{erase}(e_1)$ be a non value.
  $\mathrm{erase}((M,e_1)) \mapsto (L',d_1)$ by Untyped-Non-Values-Step and $(M,e_1) \overset{+}{\mapsto} (M',e'_1)$, $\mathrm{erase}((M',e'_1)) = (L',d_1)$ by induction. $(M,e) = (M, e_1 \,\mathrm{op}\, e_2) \overset{+,(congruence\, rule) E[e]=e \,\mathrm{op}\, e_2}{\mapsto} (M', e'_1 \,\mathrm{op}\, e_2) = (M',e')$. $\mathrm{erase}((M,e)) =$ $(\mathrm{erase}(M), \mathrm{erase}(e_1) \,\mathrm{op}\, \mathrm{erase}(e_2)) \overset{(congruence\, rule) E[e]=e \,\mathrm{op}\, e_2}{\mapsto} (L', d_1 \,\mathrm{op}\, \mathrm{erase}(e_2)) = (L',d')$. $\mathrm{erase}((M',e')) =$ $(\mathrm{erase}(M'), \mathrm{erase}(e'_1 \,\mathrm{op}\, e_2)) = (\mathrm{erase}(M'), \mathrm{erase}(e'_1) \,\mathrm{op}\, \mathrm{erase}(e_2))$ by (ER-M) and (ER-OP). This equals $(L', d_1 \,\mathrm{op}\, \mathrm{erase}(e_2)) = (L',d')$.

8.5.8 Case: Let $e = \mathbf{if}\, e_1 \,\mathbf{then}\, e_2 \,\mathbf{else}\, e_3$.

$\mathrm{erase}(e) = \mathrm{erase}(\mathrm{if}\, e_1 \,\mathrm{then}\, e_2 \,\mathrm{else}\, e_3) = \mathrm{if}\, \mathrm{erase}(e_1) \,\mathrm{then}\, \mathrm{erase}(e_2) \,\mathrm{else}\, \mathrm{erase}(e_3)$ by ER-IF.

- Let $\mathrm{erase}(e_1)$ be a value.
  $(M,e_1) \overset{*}{\mapsto} (M,v_1)$ and $\mathrm{erase}((M,e_1)) = \mathrm{erase}((M,v_1))$ by Erasure-Progress-2.
  By canonical forms, v_1 must be a boolean.

- Let v_1 be true.
$(M,e) = (M, \text{if}\, e_1 \,\text{then}\, e_2 \,\text{else}\, e_3) \overset{*,(congruence\,rule)E[e]=\text{if}\,e\,\text{then}\,e_2\,\text{else}\,e_3}{\mapsto} (M, \text{if}\, v_1 \,\text{then}\, e_2 \,\text{else}\, e_3)$ and $(M, \text{if}\, v_1 \,\text{then}\, e_2 \,\text{else}\, e_3) \overset{(E-IF1)}{\mapsto} (M, e_2) = (M, e')$. $\text{erase}(v_1) = \text{erase}(\text{true}) = \text{true}$ so $\text{erase}(e_1)$ is true. $\text{erase}((M,e)) = (\text{erase}(M), \text{if}\,\text{erase}(e_1)\,\text{then}\,\text{erase}(e_2)\,\text{else}\,\text{erase}(e_3)) \overset{(E-IF1)}{\mapsto} (\text{erase}(M), \text{erase}(e_2)) = (L', d')$. $\text{erase}((M,e'))= (\text{erase}(M), \text{erase}(e_2)) = (L', d')$.

- Let v_1 be false.
$(M,e) = (M, \text{if}\, e_1 \,\text{then}\, e_2 \,\text{else}\, e_3) \overset{*,(congruence\,rule)E[e]=\text{if}\,e\,\text{then}\,e_2\,\text{else}\,e_3}{\mapsto} (M, \text{if}\, v_1 \,\text{then}\, e_2 \,\text{else}\, e_3)$ and $(M, \text{if}\, v_1 \,\text{then}\, e_2 \,\text{else}\, e_3) \overset{(E-IF2)}{\mapsto} (M, e_3) = (M', e')$. $\text{erase}(v_1) = \text{erase}(\text{false}) = \text{false}$ so $\text{erase}(e_1)$ is false. $\text{erase}((M,e)) = (\text{erase}(M), \text{if}\,\text{erase}(e_1)\,\text{then}\,\text{erase}(e_2)\,\text{else}\,\text{erase}(e_3)) \overset{(E-IF2)}{\mapsto} (\text{erase}(M), \text{erase}(e_3)) = (L', d')$. $\text{erase}((M',e'))= (\text{erase}(M), \text{erase}(e_3)) = (L', d')$.

- Let $\text{erase}(e_1)$ be a non value.
$\text{erase}((M,e_1)) \mapsto (M', d_1)$ by Untyped-Non-Values-Step and $(M, e_1) \overset{+}{\mapsto} (M', e_1')$, $\text{erase}((M', e_1')) = (L', d_1)$ by induction. $(M,e) = (M, \text{if}\, e_1 \,\text{then}\, e_2 \,\text{else}\, e_3) \overset{+,(congruence\,rule)E[e]=\text{if}\,e\,\text{then}\,e_2\,\text{else}\,e_3}{\mapsto} (M', \text{if}\, e_1' \,\text{then}\, e_2 \,\text{else}\, e_3) = (M', e')$. $\text{erase}((M,e)) = (\text{erase}(M), \text{if}\,\text{erase}(e_1)\,\text{then}\,\text{erase}(e_2)\,\text{else}\,\text{erase}(e_3)) \overset{(congruence\,rule)E[e]=\text{if}\,e\,\text{then}\,e_2\,\text{else}\,e_3}{\mapsto} (L', \text{if}\, d_1 \,\text{then}\,\text{erase}(e_2)\,\text{else}\,\text{erase}(e_3)) = (L', d')$. $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\text{if}\, e_1' \,\text{then}\, e_2 \,\text{else}\, e_3))= (\text{erase}(M'), \text{if}\,\text{erase}(e_1')\,\text{then}\,\text{erase}(e_2)\,\text{else}\,\text{erase}(e_3))$ by (ER-M) and (ER-IF). This equals $(L', \text{if}\, d_1 \,\text{then}\,\text{erase}(e_2)\,\text{else}\,\text{erase}(e_3)) = (L', d')$.

8.5.9 Case: Let $e = \textbf{if}\, B \,\textbf{then}\, e_1 \,\textbf{else}\, e_2$.

This expression only occurs inside type coersion functions so $\text{if}\, B \,\text{then}\, e_1 \,\text{else}\, e_2$ will be removed by an erase() of a higher expression and $\text{erase}(\text{if}\, B \,\text{then}\, e_1 \,\text{else}\, e_2)$ is undefined.

8.5.10 Case: Let $e = \textbf{fix}\, x : t \overset{\phi}{:i} .v$.

- Let $i > 0$
$\text{erase}(e) = \text{erase}(\text{fix}\, x : t \overset{\phi}{:i} .v) = \text{fix}\, x. \text{erase}(v)$ by (ER-FIX). $e = \text{fix}\, x : t \overset{\phi}{:i} .v \overset{(E-FIX)}{\mapsto} [x \mapsto \text{fix}\, x : t \overset{\phi}{:i} .v]v = e'$. $\text{erase}(e) = \text{fix}\, x. \text{erase}(v) \overset{(U-FIX)}{\mapsto} [x \mapsto \text{fix}\, x. \text{erase}(v)]\, \text{erase}(v) = d'$. $\text{erase}(e') = \text{erase}([x \mapsto \text{fix}\, x : t \overset{\phi}{:i} .v]v) = [x \mapsto \text{erase}(\text{fix}\, x : t \overset{\phi}{:i} .v)]\, \text{erase}(v) = d'$ by Erase-Term-Substitution. This equals $[x \mapsto \text{fix}\, x. \text{erase}(v)]v = d'$.

- Let $i = 0$
$\text{erase}(e) = \text{erase}(\text{fix}\, x : t \overset{\phi}{:0} .v) = \langle\rangle$ by (ER-FIX0). This is a value so Erasure-Progress-3 does not apply.

8.5.11 Case: Let $(M,e) = (M, \textbf{load}(e_{ptr}, e_{Has})) \,|\, (M, \textbf{store}(e_{ptr}, e_{Has}, e_v))$

The proofs for these cases are similar. We will only show one of the proofs.
Let $(M,e) = (M, \text{load}(e_{ptr}, e_{Has}))$.
$\text{erase}((M,e)) = \text{erase}((M, \text{load}(e_{ptr}, e_{Has})))= (\text{erase}(M), \text{erase}(\text{load}(e_{ptr}, e_{Has})))$
$= (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has})))$ by (ER-M) and (ER-LOAD).

- Let $\text{erase}(e_{ptr})$ and $\text{erase}(e_{Has})$ be values.
$(M, e_{ptr}) \overset{*}{\mapsto} (M, v_{ptr})$ and $\text{erase}((M, e_{ptr})) = \text{erase}((M, v_{ptr}))$ by Erasure-Progress-2. $(M, e_{Has}) \overset{*}{\mapsto} (M, v_{Has})$ and $\text{erase}((M, e_{Has})) = \text{erase}((M, v_{Has}))$ also by Erasure-Progress-2. By canonical forms, $v_{ptr} = i$ and $v_{Has} = \text{fact}$. $\text{erase}((M,e)) = (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \overset{(U-LOAD)}{\mapsto} (\text{erase}(M), \langle M(i), \langle\rangle\rangle) = (L', d')$.$(M,e) = (M, \text{load}(e_{ptr}, e_{Has}))$
$\overset{*,(congruence\,rule)E[e]=\text{load}(e, e_{Has})}{\mapsto} (M, \text{load}(i, e_{Has}))$ and $(M, \text{load}(i, e_{Has})) \overset{*,(congruence\,rule)E[e]=\text{load}(v_{ptr}, e)}{\mapsto}$
$(M, \text{load}(i, \text{fact}))$ and $(M, \text{load}(i, \text{fact})) \overset{(E-LOAD)}{\mapsto} (M, \wedge\langle M(i), \text{fact}\rangle) = (M, e')$. $\text{erase}((M, e'))$
$= \text{erase}((M, \wedge\langle M(i), \text{fact}\rangle))= (\text{erase}(M), \langle M(i), \langle\rangle\rangle)= (L, d')$ by (ER-M), (ER-M2), (ER-i), and (ER-FACT).

- Let $\text{erase}(e_{ptr})$ be a value and $\text{erase}(e_{Has})$ be a non value.
$(M, e_{ptr}) \overset{*}{\mapsto} (M, v_{ptr})$ and $\text{erase}((M, e_{ptr})) = \text{erase}((M, v_{ptr}))$ by Erasure-Progress-2. By canonical forms, $v_{ptr} = i$. $\text{erase}((M, e_{Has})) \mapsto (L_2', d_{Has})$ by Untyped-Non-Values-Step and
$(M, e_{Has}) \overset{+}{\mapsto} (M_2', e_{Has}')$, $\text{erase}((M_2', e_{Has}')) = (L_2', d_{Has})$ by induction.
$\text{erase}((M, e)) = (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \overset{(congruence\,rule) E[e]=\text{load}(v_{ptr}, e)}{\mapsto}$
$(L_2', \text{load}(\text{erase}(e_{ptr}), d_{Has})) = (L_2', \text{load}(i, d_{Has})) = (L', d')$. $(L_2' = L')$. $(M, e) = (M, \text{load}(e_{ptr}, e_{Has}))$
$\overset{*,(congruence\,rule) E[e]=\text{load}(e, e_{Has})}{\mapsto} (M, \text{load}(i, e_{Has}))$ and $(M, \text{load}(i, e_{Has})) \overset{+,(congruence\,rule) E[e]=\text{load}(v_{ptr}, e)}{\mapsto}$
$(M_2', \text{load}(i, e_{Has}')) = (M', e')$. $(M_2' = M')$.
$\text{erase}((M', e')) = \text{erase}((M_2', \text{load}(i, e_{Has}'))) = (\text{erase}(M_2'), \text{load}(i, \text{erase}(e_{Has}'))$ by (ER-M), (ER-LOAD), and (ER-i). This equals $(L_2', \text{load}(i, d_{Has}) = (L', d')$.

- Let $\text{erase}(e_{ptr})$ be a non value.
$\text{erase}((M, e_{ptr})) \mapsto (L, d_{ptr})$ by Untyped-Non-Values-Step and $(M, e_{ptr}) \overset{+}{\mapsto} (M', e_{ptr}')$, $\text{erase}((M', e_{ptr}')) = (L', d_{ptr})$ by induction. $e_{ptr}' = i$ so $\text{erase}(e_{ptr}') = \text{erase}(i) = i = d_{ptr}$. $\text{erase}((M, e)) =$
$(\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \overset{(congruence\,rule) E[e]=\text{load}(e, e_{Has})}{\mapsto} (L', \text{load}(d_{ptr}, \text{erase}(e_{Has}))) = (L', d')$.
$(M, e) = (M, \text{load}(e_{ptr}, e_{Has}))$
$\overset{+,(congruence\,rule) E[e]=\text{load}(e, e_{Has})}{\mapsto} (M', \text{load}(i, e_{Has})) = (M', e')$. $\text{erase}((M', e')) = \text{erase}((M', \text{load}(i, e_{Has}))) =$
$(\text{erase}(M'), \text{load}(i, \text{erase}(e_{Has})))$ by (ER-M), (ER-LOAD), and (ER-i). This equals $(L', \text{load}(d_{ptr}, \text{erase}(e_{Has}))) =$
(L', d').

8.5.12 Case: Let $e = \mathbf{coerce}(e) \,|\, \mathbf{make\_eq}(\tau) \,|\, \mathbf{new\_fun}(K)$.

All of these erase to values. These proofs are similar therefore we will only show one.

Let $e = \text{make\_eq}(\tau)$.

$\text{erase}(e) = \text{erase}(\text{make\_eq}(\tau)) = \langle\rangle$ by (ER-MAKEEQ). This is a value so Erasure-Progress-3 does not apply.

8.5.13 Case: $e = \mathbf{distinguish}(I_1, I_2, e_1, e_2) \,|\, \mathbf{apply\_eq}(\tau, e_1, e_2) \,|\, \mathbf{discard\_fun}(e_1) \,|\, \mathbf{define\_fun}(e_1, \tau)$ $|\, \mathbf{in\_domain}(I_1, I_2, e_1, e_2)$.

The proofs for these are all similar. We will show only one of these proofs.

Let $e = \text{discard\_fun}(e_1)$.

$\text{erase}(e) = \text{erase}(\text{discard\_fun}(e_1)) = \text{let}\, x = \text{erase}(e_1) \,\text{in}\, \langle\rangle$ by (ER-DISCARDFUN). By Untyped-Non-Values-Step, $\text{erase}((M, e)) \mapsto (L', d')$.

- Let $\text{erase}(e_1)$ be a value.
$(M, e_1) \overset{*}{\mapsto} (M, v_1)$ and $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$ by Erasure-Progress-2. By Value-Erase-Value, $\text{erase}(v_1)$ is a value. $\text{erase}((M, e_1)) = \text{erase}((M, \langle\rangle))$. $\text{erase}((M, e)) = (\text{erase}(M), \text{let}\, x = \text{erase}(e_1) \,\text{in}\, \langle\rangle) \overset{(E-LET)}{\mapsto}$
$(\text{erase}(M), \langle\rangle) = (L', d')$. $(M, e) = (M, \text{discard\_fun}(e_1)) \overset{*,(congruence\,rule) E[e]=\text{discard\_fun}(e)}{\mapsto} (M, \text{discard\_fun}(v_1))$
$\overset{(E-DISCARDFUN)}{\mapsto} (M, \cdot\langle\rangle) = (M, e')$. $\text{erase}((M, e')) = (\text{erase}(M), \text{erase}(\cdot\langle\rangle))$. $= (\text{erase}(M), \langle\rangle) = (L', d')$ by (ER-M) and (ER-TUPLE) or (ER-TUPLEv0).

- Let $\text{erase}(e_1)$ be a non value.
$\text{erase}((M, e_1)) \mapsto (L', d_1)$ by Untyped-Non-Values-Step and $(M, e_1) \overset{+}{\mapsto} (M', e_1')$, $\text{erase}((M', e_1')) = (L', d_1)$ by induction. $(M, e) = (M, \text{discard\_fun}(e_1)) \overset{+,(congruence\,rule) E[e]=\text{discard\_fun}(e)}{\mapsto} (M', \text{discard\_fun}(e_1')) = (M', e')$.
$\text{erase}((M, e)) = (\text{erase}(M), \text{let}\, x = \text{erase}(e_1) \,\text{in}\, \langle\rangle \overset{(congruence\,rule) E[e]=\text{let}\, x=e \,\text{in}\, e_2}{\mapsto} (L', \text{let}\, x = d_1 \,\text{in}\, \langle\rangle) = (L', d')$.
$\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\text{discard\_fun}(e_1'))) = (\text{erase}(M'), \text{let}\, x = \text{erase}(e_1') \,\text{in}\, \langle\rangle)$ by (ER-M) and (ER-DISCARDFUN). This equals $(L', \text{let}\, x = d_1 \,\text{in}\, \langle\rangle) = (L', d')$.

9 Converting λ^C to λ^{low}

We define a translation from λ^C to λ^{low}, and prove the correctness of the translation. The language λ^C was defined by Morrisett et al[3] as follows:

$\tau \;=\; \alpha \,|\, \mathrm{int} \,|\, \forall[\overrightarrow{\alpha}].(\overrightarrow{\tau}) \rightarrow \mathrm{void} \,|\, \langle \overrightarrow{\tau} \rangle \,|\, \exists \alpha.\tau$

$e \;=\; v(\overrightarrow{v}) \,|\, \mathrm{if0}(v,\, e_1,\, e_2) \,|\, \mathrm{halt}[\tau]v \,|\, \mathrm{let}\, x = v \,\mathrm{in}\, e \,|\, \mathrm{let}\, x = \pi_i v \,\mathrm{in}\, e$
$\quad |\, \mathrm{let}\, x = v_1\, p\, v_2 \,\mathrm{in}\, e \,|\, \mathrm{let}\, [\alpha, x] = \mathrm{unpack}\, v \,\mathrm{in}\, e$

$v = x \,|\, i \,|\, \langle \overrightarrow{v} \rangle \,|\, v[\tau] \,|\, \mathrm{pack}[\tau,\, v]\, \mathrm{as}\, \exists \alpha.\tau'$
$\quad |\, fixcode\, x[\overrightarrow{\alpha}](\overrightarrow{x : \tau}).e$

$p = + \,|\, - \,|\, *$

The languages λ^{C_2} to λ^{flat}, defined below, are intermediate languages to assist the ultimate translation from λ^C to λ^{low}.

9.1 Converting λ^C to λ^{C_2}

The language λ^{C_2} extends λ^C with two distinct kinds (pointer and primitive):

$K \;=\; \mathrm{ptr} \;\;|\;\; \mathrm{primitive}$

$\tau \;=\; \alpha \,|\, \mathrm{int} \,|\, \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \mathrm{void} \,|\, \langle \overrightarrow{\tau} \rangle \,|\, \exists \alpha : K.\tau$

$e \;=\; v(\overrightarrow{v}) \,|\, \mathrm{if0}(v,\, e_1,\, e_2) \,|\, \mathrm{halt}[\tau]v \,|\, \mathrm{let}\, x = v \,\mathrm{in}\, e$
$\quad |\, \mathrm{let}\, x = v_1\, p\, v_2 \,\mathrm{in}\, e \,|\, \mathrm{let}\, [\alpha, x] = \mathrm{unpack}\, v \,\mathrm{in}\, e$

$v = x \,|\, i \,|\, \langle \overrightarrow{v} \rangle \,|\, v[\tau] \,|\, \mathrm{pack}[\tau,\, v]\, \mathrm{as}\, \exists \alpha : K.\tau' \,|\, \pi_i v$
$\quad |\, fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e$

$p = + \,|\, - \,|\, *$

To make the λ^C-to-λ^{C_2} translation neater, λ^{C_2} defines $\pi_i v$ to be a v. The later λ^{C_2}-to-λ^{flat} translation removes $\pi_i v$ from v.

Type Checking Rules for $\lambda^{\mathbf{C_2}}$:

(KD-VAR) $\quad \Delta, \alpha : K \vdash \alpha : K$

(KD-INT) $\quad \Delta \vdash \mathrm{int{:}primitive}$

(KD-ALL) $\quad \dfrac{\Delta,\, \overrightarrow{\alpha : K} \vdash \tau_i : K_i}{\Delta \vdash \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \mathrm{void} : \mathrm{primitive}}$

(KD-TUPLE) $\quad \dfrac{\Delta \vdash \tau_i : K_i}{\Delta \vdash \langle \overrightarrow{\tau} \rangle : \mathrm{ptr}}$

(KD-SOME) $\quad \dfrac{\Delta,\, \alpha : K \vdash \tau : K'}{\Delta \vdash (\exists \alpha : K.\tau) : K'}$

(TD-APP) $\quad \dfrac{\Delta; \Gamma \vdash v : (\tau_1, \cdots, \tau_n) \rightarrow \mathrm{void} \qquad \Delta; \Gamma \vdash v_i : \tau_i}{\Delta; \Gamma \vdash v(\overrightarrow{v})}$

(TD-IF) $\quad \dfrac{\Delta; \Gamma \vdash v : \mathrm{int} \quad \Delta; \Gamma \vdash e_1 \;\; \Delta; \Gamma \vdash e_2}{\Delta; \Gamma \vdash \mathrm{if0}(v,\, e_1,\, e_2)}$

(TD-HALT) $\dfrac{\Delta;\Gamma \vdash v : \tau}{\Delta;\Gamma \vdash \mathrm{halt}[\tau]v}$

(TD-SUB) $\dfrac{\Delta;\Gamma \vdash v : \tau \quad \Delta;\Gamma,\, x : \tau \vdash e}{\Delta;\Gamma \vdash \mathrm{let}\, x = v \,\mathrm{in}\, e}$ $(x \notin \Gamma)$

(TD-PROJECT) $\dfrac{\Delta;\Gamma \vdash v : \langle \tau_1, \cdots, \tau_n \rangle}{\Delta;\Gamma \vdash \pi_i v : \tau_i}$ $(1 \leq i \leq n)$

(TD-UNPACK) $\dfrac{\Delta;\Gamma \vdash v : \exists \alpha : K.\tau \quad (\Delta, \alpha : K);(\Gamma, x : \tau) \vdash e}{\Delta;\Gamma \vdash \mathrm{let}\, [\alpha, x] = \mathrm{unpack}\, v \,\mathrm{in}\, e}$ $(x \notin \Gamma \wedge \alpha \notin \Delta)$

(TD-OP) $\dfrac{\Delta;\Gamma \vdash v_1 : \mathrm{int} \;\; \Delta;\Gamma \vdash v_2 : \mathrm{int} \;\; \Delta;\Gamma,\, x : \mathrm{int} \vdash e}{\Delta;\Gamma \vdash \mathrm{let}\, x = v_1\, p\, v_2 \,\mathrm{in}\, e}$ $(x \notin \Gamma)$

(TD-VAR) $\Delta;\Gamma, x : \tau \vdash x : \tau$

(TD-INT) $\Delta;\Gamma \vdash i : \mathrm{int}$

(TD-TAPP) $\dfrac{\Delta \vdash \tau_0 : K \quad \Delta;\Gamma \vdash v : \forall[\alpha : K, \overrightarrow{\beta : K'}].(\overrightarrow{\tau}) \to \mathrm{void}}{\Delta;\Gamma \vdash v[\tau_0] : (\forall[\overrightarrow{\beta : K'}].(\overrightarrow{\tau}[\tau_0/\alpha]) \to \mathrm{void})}$

(TD-TUPLE) $\dfrac{\Delta;\Gamma \vdash v_i : \tau_i}{\Delta;\Gamma \vdash \langle \overrightarrow{v} \rangle : \langle \overrightarrow{\tau} \rangle}$ $(x_0 \notin \Gamma)$

(TD-PACK) $\dfrac{\Delta \vdash \tau : K \quad \Delta;\Gamma \vdash v : \tau'[\tau/\alpha]}{\Delta;\Gamma \vdash \mathrm{pack}[\tau,\, v]\,\mathrm{as}\, \exists \alpha : K.\tau' :\; \exists \alpha : K.\tau'}$

(TD-FIX) $\dfrac{\overrightarrow{\alpha : K} \vdash \tau_i : K_i' \quad \overrightarrow{\alpha : K};\, x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \mathrm{void},\, x_1 : \tau_1, \cdots, x_n : \tau_n \vdash e}{\Delta;\Gamma \vdash \;\; \mathit{fixcode}\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e \;\; : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \mathrm{void}}$

(TD-ENV) $\dfrac{\Delta \vdash \tau_i : K_i}{\Delta \vdash \; \{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\}}$

The translation from λ^C to λ^{C_2} boxes all non-pointer types:

$\mathcal{B}(\{\alpha_1, \ldots, \alpha_n\}) = \{\alpha_1 \mapsto \mathrm{ptr}, \ldots, \alpha_n \mapsto \mathrm{ptr}\}$

$\mathcal{B}(\{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\}) = \{x_1 \mapsto \mathcal{B}(\tau_1), \ldots, x_n \mapsto \mathcal{B}(\tau_n)\}$

$\mathcal{B}(\alpha) = \alpha$

$\mathcal{B}(\mathrm{int}) = \langle \mathrm{int} \rangle$

$\mathcal{B}(\forall \overrightarrow{\alpha}.\overrightarrow{\tau} \to \mathrm{void}) = \langle \forall \overrightarrow{\alpha : \mathrm{ptr}}.\overrightarrow{B(\tau)} \to \mathrm{void} \rangle$

$\mathcal{B}(\langle \overrightarrow{\tau} \rangle) = \langle \overrightarrow{B(\tau)} \rangle$

$\mathcal{B}(\exists \alpha.\tau') = \exists \alpha : \mathrm{ptr}.\mathcal{B}(\tau')$

$\mathcal{B}(v(\overrightarrow{v})) = (\pi_1(\mathcal{B}(v)))(\overrightarrow{\mathcal{B}(v)})$

$\mathcal{B}(\mathrm{if0}(v, e_1, e_2)) = \mathrm{if0}(\pi_1(\mathcal{B}(v)), \mathcal{B}(e_1), \mathcal{B}(e_2))$

$\mathcal{B}(\mathrm{halt}[\tau]v) = \mathrm{halt}[\mathcal{B}(\tau)]\mathcal{B}(v)$

$\mathcal{B}(\mathrm{let}\, x = v \,\mathrm{in}\, e) = \mathrm{let}\, x = \mathcal{B}(v) \,\mathrm{in}\, \mathcal{B}(e)$

$\mathcal{B}(\mathrm{let}\, x = \pi_i(v)\,\mathrm{in}\, e) = \mathrm{let}\, x = \pi_i(\mathcal{B}(v))\,\mathrm{in}\,\mathcal{B}(e)$

$\mathcal{B}(\mathrm{let}\, x = v_1\, p\, v_2\,\mathrm{in}\, e) = \mathrm{let}\, x = \langle(\pi_1(\mathcal{B}(v_1)))\, p\,(\pi_1(\mathcal{B}(v_2)))\rangle\,\mathrm{in}\,\mathcal{B}(e)$

$\mathcal{B}(\mathrm{let}\,[\alpha, x] = \mathrm{unpack}\, v\,\mathrm{in}\, e) = \mathrm{let}\,[\alpha, x] = \mathrm{unpack}\,\mathcal{B}(v)\,\mathrm{in}\,\mathcal{B}(e)$

$\mathcal{B}(x) = x$

$\mathcal{B}(i) = \langle i\rangle$

$\mathcal{B}(\langle\overrightarrow{v}\rangle) = \langle\overrightarrow{\mathcal{B}(v)}\rangle$

$\mathcal{B}(v[\tau]) = \langle(\pi_1(\mathcal{B}(v)))[\mathcal{B}(\tau)]\rangle$

$\mathcal{B}(\mathrm{pack}[\tau,\, v]\,\mathrm{as}\,\exists\alpha : K.\tau') = \mathrm{pack}[\mathcal{B}(\tau),\, \mathcal{B}(v)]\,\mathrm{as}\,\exists\alpha : \mathrm{ptr}.\mathcal{B}(\tau')$

$\mathcal{B}(\mathit{fixcode}\, x[\overrightarrow{\alpha}](\overrightarrow{x:\tau}).e) = \langle\mathit{fixcode}\, x_0[\overrightarrow{\alpha:\mathrm{ptr}}](\overrightarrow{x:\mathcal{B}(\tau)}).\mathrm{let}\, x = \langle x_0\rangle\,\mathrm{in}\,\mathcal{B}(e)\rangle$

Lemma1: If $\mathcal{B}(\Delta);\mathcal{B}(\Gamma)\vdash_{C_2} e : \mathcal{B}(\tau[\tau'/\alpha])$, then $\mathcal{B}(\Delta);\mathcal{B}(\Gamma)\vdash_{C_2} e : \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$

Proof :

$1a.\ \tau\ =\ \alpha$
$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau')$
$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\alpha)[\mathcal{B}(\tau')/\alpha] = \alpha[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\tau')$
Thus, $\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$.

$1b.\ \tau\ =\ \beta \neq \alpha$
$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\beta) = \beta$
$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\beta)[\mathcal{B}(\tau')/\alpha] = \beta[\mathcal{B}(\tau')/\alpha] = \beta$
Thus, $\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$.

2. $\tau = \mathrm{int}$
$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\mathrm{int}[\tau'/\alpha]) = \mathcal{B}(\mathrm{int}) = \langle\mathrm{int}\rangle$
$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\mathrm{int})[\mathcal{B}(\tau')/\alpha] = \langle\mathrm{int}\rangle[\mathcal{B}(\tau')/\alpha] = \langle\mathrm{int}\rangle$
Thus, $\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$.

3. $\tau = \forall[\overrightarrow{\beta}].(\overrightarrow{\tau}) \rightarrow \mathrm{void}$
$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}((\forall[\overrightarrow{\beta}].(\overrightarrow{\tau}) \rightarrow \mathrm{void})[\tau'/\alpha]) = \mathcal{B}(\forall[\overrightarrow{\beta}].(\overrightarrow{\tau[\tau'/\alpha]}) \rightarrow \mathrm{void})$
$= \langle\forall\overrightarrow{\alpha:\mathrm{ptr}}.\overrightarrow{\mathcal{B}(\tau[\tau'/\alpha])} \rightarrow \mathrm{void}\rangle$

$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\forall[\overrightarrow{\beta}].(\overrightarrow{\tau}) \rightarrow \mathrm{void})[\mathcal{B}(\tau')/\alpha] = \langle\forall\overrightarrow{\alpha:\mathrm{ptr}}.\overrightarrow{B(\tau)} \rightarrow \mathrm{void}\rangle[\mathcal{B}(\tau')/\alpha]$
$= \langle\forall\overrightarrow{\alpha:\mathrm{ptr}}.\overrightarrow{B(\tau)[\mathcal{B}(\tau')/\alpha]} \rightarrow \mathrm{void}\rangle$
By induction, we know $\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \langle\forall\overrightarrow{\alpha:\mathrm{ptr}}.\overrightarrow{\mathcal{B}(\tau[\tau'/\alpha])} \rightarrow \mathrm{void}\rangle$
Thus, $\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$.

4. $\tau = \langle\overrightarrow{\tau}\rangle$
$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\langle\overrightarrow{\tau}\rangle[\tau'/\alpha]) = \mathcal{B}(\langle\overrightarrow{\tau[\tau'/\alpha]}\rangle) = \langle\overrightarrow{\mathcal{B}(\tau[\tau'/\alpha])}\rangle$
$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\langle\overrightarrow{\tau}\rangle)[\mathcal{B}(\tau')/\alpha] = \langle\overrightarrow{B(\tau)}\rangle[\mathcal{B}(\tau')/\alpha] = \langle\overrightarrow{B(\tau)[\mathcal{B}(\tau')/\alpha]}\rangle$
By induction, we can get $\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \langle\overrightarrow{\mathcal{B}(\tau[\tau'/\alpha])}\rangle$

Thus, $\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$.

5. $\tau = \exists\beta.\tau_0$
$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}((\exists\beta.\tau_0)[\tau'/\alpha]) = \mathcal{B}(\exists\beta : \text{ptr}.(\tau_0[\tau'/\alpha])) = \exists\beta : \text{ptr}.\mathcal{B}((\tau_0[\tau'/\alpha]))$
$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\exists\beta.\tau_0)[\mathcal{B}(\tau')/\alpha] = \exists\beta : \text{ptr}.\mathcal{B}(\tau_0)[\mathcal{B}(\tau')/\alpha] = \exists\beta : \text{ptr}.(\mathcal{B}(\tau_0)[\mathcal{B}(\tau')/\alpha])$
By induction, we obtain $\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \exists\beta : \text{ptr}.(\mathcal{B}(\tau_0)[\mathcal{B}(\tau')/\alpha]) = \exists\beta : \text{ptr}.\mathcal{B}((\tau_0[\tau'/\alpha]))$
Thus, $\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]$.

Correctness :

1. If $\Delta \vdash_C \tau$, then $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr}$
2. If $\Delta \vdash_C \Gamma$, then $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\Gamma)$
3. If $\Delta; \Gamma \vdash_C e$, then $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(e)$
4. If $\Delta; \Gamma \vdash_C v : \tau$, then $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \mathcal{B}(\tau)$

Proof :

The proof of $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\Gamma)$ comes directly from $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau)$: ptr. The other proofs are by structural induction on the typing and kinding derivations:

1. $\tau = \alpha$, $\mathcal{B}(\tau) = \alpha$
Because $\Delta \vdash_C \tau \quad (FTV(\tau) \subseteq \Delta)$,
$\mathcal{B}(\Delta) = \Delta', \alpha : \text{ptr}$
By the (KD-VAR) rule, we know $\Delta', \alpha : \text{ptr} \vdash_{C_2} \alpha : \text{ptr}$
Thus, $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr}$

2. $\tau = \text{int}$, $\mathcal{B}(\tau) = \langle \text{int} \rangle$
From (KC-INT), $\Delta \vdash_C \text{int}$
By (KD-INT), $\mathcal{B}(\Delta) \vdash_{C_2} \text{int} : \text{primitive}$
By the (KD-TUPLE), we know $\mathcal{B}(\Delta) \vdash_{C_2} \langle \text{int} \rangle : \text{ptr}$
Thus, $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr}$

3. $\tau = \forall\overrightarrow{\alpha}.\overrightarrow{\tau} \to \text{void}$, $B(\tau) = \langle \forall\overrightarrow{\alpha : \text{ptr}}.\overrightarrow{B(\tau)} \to \text{void} \rangle$
$\Delta \vdash_C \forall\overrightarrow{\alpha}.\overrightarrow{\tau} \to \text{void}$ where $\Delta, \overrightarrow{\alpha} \vdash_C \tau_i$.
By induction, $\mathcal{B}(\Delta, \overrightarrow{\alpha}) \vdash_C \mathcal{B}(\tau_i) : \text{ptr}$.
By (KD-ALL), $\mathcal{B}(\Delta) \vdash_C (\forall\overrightarrow{\alpha : \text{ptr}}.\overrightarrow{B(\tau)} \to \text{void}) : \text{primitive}$
By the (KD-TUPLE) rule, we know $B(\Delta) \vdash_{C_2} \langle \forall\overrightarrow{\alpha : \text{ptr}}.\overrightarrow{B(\tau)} \to \text{void} \rangle : \text{ptr}$
Thus, $B(\Delta) \vdash_{C_2} B(\tau) : \text{ptr}$

4. $\tau_0 = \langle \overrightarrow{\tau} \rangle$, $B(\tau_0) = \langle \overrightarrow{B(\tau)} \rangle$
$\Delta \vdash_C \langle \overrightarrow{\tau} \rangle$, and $\Delta \vdash_C \tau_i$
By induction, $B(\Delta) \vdash_{C_2} B(\tau_i) : K_i$
From the (KD-TUPLE) rule, we know $B(\Delta) \vdash_{C_2} \langle \overrightarrow{B(\tau)} \rangle : \text{ptr}$
Thus, $B(\Delta) \vdash_{C_2} B(\tau_0) : \text{ptr}$

5. $\tau = \exists\alpha.\tau'$, $\mathcal{B}(\tau) = \exists\alpha : \text{ptr}.\mathcal{B}(\tau')$
Because $\Delta \vdash_C \exists\alpha.\tau'$, then $\Delta, \alpha \vdash_C \tau'$
By induction, we know $\mathcal{B}(\Delta), \alpha : \text{ptr} \vdash_{C_2} \mathcal{B}(\tau') : \text{ptr}$
By the (KD-SOME), $\Delta \vdash_{C_2} \exists\alpha : \text{ptr}.\mathcal{B}(\tau') : \text{ptr}$

6. $e = v(\overrightarrow{v})$, $\mathcal{B}(e) = (\pi_1(\mathcal{B}(v)))(\overrightarrow{\mathcal{B}(v)})$
By the (TC-APP), $\Delta; \Gamma \vdash_C v : (\tau_1, \cdots, \tau_n) \rightarrow \text{void}$ and $\Delta; \Gamma \vdash_C v_i : \tau_i$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \langle \overrightarrow{\mathcal{B}(\tau)} \rightarrow \text{void} \ \rangle$
and $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v_i) : \mathcal{B}(\tau_i)$
By (TD-PROJECT), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \pi_1(\mathcal{B}(v)) : \overrightarrow{\mathcal{B}(\tau)} \rightarrow \text{void}$,
By (TD-APP), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} (\pi_1(\mathcal{B}(v)))(\overrightarrow{\mathcal{B}(v)})$

7. $e = \text{if0}(v, e_1, e_2)$, $\mathcal{B}(e) = \text{if0}(\pi_1(\mathcal{B}(v)), \mathcal{B}(e_1), \mathcal{B}(e_2))$
By the (TC-IF), $\Delta; \Gamma \vdash_C v : \text{int}$ and $\Delta; \Gamma \vdash_C e_1$, $\Delta; \Gamma \vdash_C e_2$
By induction,$\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \langle \text{int} \rangle$,
and $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(e_1)$, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(e_2)$
By (TD-PROJECT), $\Delta; \Gamma \vdash_{C_2} \pi_1(\mathcal{B}(v)) : \text{int}$
By (TD-IF), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \text{if0}(\pi_1(\mathcal{B}(v)), \mathcal{B}(e_1), \mathcal{B}(e_2))$

8. $e = \text{halt}[\tau]v$, $\mathcal{B}(\text{e}) = \text{halt}[\mathcal{B}(\tau)]\mathcal{B}(v)$
By (TC-HALT), $\Delta; \Gamma \vdash_C v : \tau$
By induction,$\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \mathcal{B}(\tau)$,
By (TD-HALT), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \text{halt}[\mathcal{B}(\tau)]\mathcal{B}(v)$

9. $e_0 = \text{let}\, x = v \,\text{in}\, e$, $\mathcal{B}(e_0) = \text{let}\, x = \mathcal{B}(v) \,\text{in}\, \mathcal{B}(e)$
By the (TC-SUB), $\Delta; \Gamma \vdash_C v : \tau$ and $\Delta; \Gamma;\ x : \tau \vdash_C e$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \mathcal{B}(\tau)$ and $\mathcal{B}(\Delta); \mathcal{B}(\Gamma),\ x : \mathcal{B}(\tau) \vdash_{C_2} \mathcal{B}(e)$
By (TD-SUB) rule, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \text{let}\, x = \mathcal{B}(v) \,\text{in}\, \mathcal{B}(e)$

10. $e_0 = \text{let}\, x = \pi_i(v) \,\text{in}\, e$, $\mathcal{B}(e_0) = \text{let}\, x = \pi_i(\mathcal{B}(v)) \,\text{in}\, \mathcal{B}(e)$
By (TC-PROJECT), $\Delta; \Gamma \vdash_C v : \langle \tau_{1,} \cdots, \tau_n \rangle$, and $\Delta; \Gamma,\ x : \tau_i \vdash_C e$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \langle \overrightarrow{\mathcal{B}(\tau_i)} \rangle$ and $\mathcal{B}(\Delta); \mathcal{B}(\Gamma),\ x : \mathcal{B}(\tau_i) \vdash_{C_2} \mathcal{B}(e)$
By (TD-PROJECT) rule, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \text{let}\, x = \pi_i(\mathcal{B}(v)) \,\text{in}\, \mathcal{B}(e)$

11. $e_0 = \text{let}\, x = v_1\, p\, v_2 \,\text{in}\, e$, $\mathcal{B}(e_0) = \text{let}\, x = \langle (\pi_1(\mathcal{B}(v_1)))\, p\, (\pi_1(\mathcal{B}(v_2)))) \rangle \,\text{in}\, \mathcal{B}(e)$
By (TC-OP), we know $\Delta; \Gamma \vdash_C v_1 : \text{int}$, $\Delta; \Gamma \vdash_C v_2 : \text{int}$,
and $\Delta; \Gamma,\ x : \text{int} \vdash_C e$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v_1) : \langle \text{int} \rangle$,
$\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v_2) : \langle \text{int} \rangle$, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma),\ x : \langle \text{int} \rangle \vdash_{C_2} \mathcal{B}(e)$
By (TD-PROJECT), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \pi_1(\mathcal{B}(v_1)) : \text{int}$
By (TD-PROJECT), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \pi_1(\mathcal{B}(v_2)) : \text{int}$
By (TD-TUPLE) and (TD-OP), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle (\pi_1(\mathcal{B}(v_1)))\, p\, (\pi_1(\mathcal{B}(v_2)))) \rangle : \langle \text{int} \rangle$
By (TD-VAR), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(e_0)$

12. $e_0 = \text{let}\, [\alpha, x] = \text{unpack}\, v \,\text{in}\, e$, $\mathcal{B}(e_0) = \text{let}\, [\alpha, x] = \text{unpack}\, \mathcal{B}(v) \,\text{in}\, \mathcal{B}(e)$
From (TC-UNPACK), we know $\Delta; \Gamma \vdash_C v : \exists \alpha.\tau$, and
$(\Delta, \alpha); (\Gamma, x : \tau) \vdash_C e$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \exists \alpha : \text{ptr}.\mathcal{B}(\tau)$
$\mathcal{B}(\Delta),\ \alpha : \text{ptr}; \mathcal{B}(\Gamma),\ x : \mathcal{B}(\tau) \vdash_{C_2} \mathcal{B}(e)$
By (TD-UNPACK) rule, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \text{let}\, [\alpha, x] = \text{unpack}\, \mathcal{B}(v) \,\text{in}\, \mathcal{B}(e)$

13. $v = x$, $\mathcal{B}(v) = x$
From (TC-VAR), $\Delta; \Gamma, x : \tau \vdash_C x : \tau$

From (TD-VAR), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma), x : \mathcal{B}(\tau) \vdash_{C_2} x : \mathcal{B}(\tau)$

14. $v = i$, $\mathcal{B}(v) = \langle i \rangle$
By (TC-INT), $\Delta; \Gamma \vdash_C i : \text{int}$
By (TD-INT), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} i : \text{int}$,
and by (TD-TUPLE), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle i \rangle : \langle \text{int} \rangle$

15. $v = \langle \overrightarrow{v} \rangle$, $\mathcal{B}(v) = \langle \overrightarrow{\mathcal{B}(v)} \rangle$
By (TC-TUPLE), $\Delta; \Gamma \vdash_C \langle \overrightarrow{v} \rangle : \langle \overrightarrow{\tau} \rangle$, and $\Delta; \Gamma \vdash_C v_i : \tau_i$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v_i) : \mathcal{B}(\tau_i)$
By (TD-TUPLE) rule, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle \overrightarrow{\mathcal{B}(v)} \rangle : \langle \overrightarrow{\mathcal{B}(\tau)} \rangle$

16. $v_0 = v[\tau]$, $\mathcal{B}(v_0) = \langle (\pi_1(\mathcal{B}(v)))[\mathcal{B}(\tau)] \rangle$
From (TC-TAPP), $\Delta; \Gamma \vdash_C v : \forall[\alpha, \overrightarrow{\beta}].\overrightarrow{\tau'} \to \text{void}$, and $\Delta \vdash_C \tau$
By induction, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \langle \forall[\alpha : \text{ptr}, \overrightarrow{\beta : \text{ptr}}].\overrightarrow{\mathcal{B}(\tau')} \to \text{void} \rangle$,
and $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr}$
By (TD-PROJECT), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} (\pi_1 \mathcal{B}(v)) : \forall[\alpha : \text{ptr}, \overrightarrow{\beta : \text{ptr}}].\overrightarrow{\mathcal{B}(\tau')} \to \text{void}$
By (TD-TAPP), we obtain
$\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \pi_1(\mathcal{B}(v))[\mathcal{B}(\tau)] : \forall \overrightarrow{\beta : \text{ptr}}.\overrightarrow{\mathcal{B}(\tau')}[\mathcal{B}(\tau)/\alpha] \to \text{void}$
By the Lemma above, $\forall \overrightarrow{\beta : \text{ptr}}.\overrightarrow{\mathcal{B}(\tau')}[\mathcal{B}(\tau)/\alpha] \to \text{void} = \forall \overrightarrow{\beta : \text{ptr}}.\overrightarrow{\mathcal{B}([\tau/\alpha]\tau')} \to \text{void}$
and $\langle \forall \overrightarrow{\beta : \text{ptr}}.\overrightarrow{\mathcal{B}([\tau/\alpha]\tau')} \to \text{void} \rangle = \mathcal{B}(\forall \overrightarrow{\beta}.\overrightarrow{[\tau/\alpha]\tau'} \to \text{void})$
Thus by (TD-TUPLE), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle (\pi_1(\mathcal{B}(v)))[\mathcal{B}(\tau)] \rangle : \mathcal{B}(\forall \overrightarrow{\beta}.\overrightarrow{[\tau/\alpha]\tau'} \to \text{void})$

17. $e = \mathit{fixcode}\, x[\overrightarrow{\alpha}](\overrightarrow{x : \tau}).e$, $\mathcal{B}(e) = \langle \mathit{fixcode}\, x_0[\overrightarrow{\alpha : \text{ptr}}](\overrightarrow{x : \mathcal{B}(\tau)}).\text{let}\, x = \langle x_0 \rangle \,\text{in}\, \mathcal{B}(e) \rangle$
From (TC-FIX), $\quad \overrightarrow{\alpha} \vdash_C \tau_i \quad , \overrightarrow{\alpha}; x : \forall[\overrightarrow{\alpha}].(\overrightarrow{\tau}) \to \text{void}, x_1 : \tau_1, \cdots, x_n : \tau_n \vdash_C e$
By induction, $\quad \overrightarrow{\alpha : \text{ptr}} \vdash_{C_2} \mathcal{B}(\tau_i) : \text{ptr} \quad ,$
$\overrightarrow{\alpha : \text{ptr}}; x : \langle \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\mathcal{B}(\tau)}) \to \text{void} \rangle, x_1 : \mathcal{B}(\tau_1), \cdots, x_n : \mathcal{B}(\tau_n) \vdash_{C_2} \mathcal{B}(e)$
we know $x_0 : \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\mathcal{B}(\tau)}) \to \text{void} \vdash_{C_2} \langle x_0 \rangle : \langle \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\mathcal{B}(\tau)}) \to \text{void} \rangle$
By weakening lemma, $\overrightarrow{\alpha : \text{ptr}}; x : \langle \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\mathcal{B}(\tau)}) \to \text{void} \rangle, x_1 : \mathcal{B}(\tau_1), \cdots, x_n : \mathcal{B}(\tau_n), x_0 : \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\tau}) \to \text{void} \vdash_{C_2} \mathcal{B}(e)$
By (TD-SUB), $\overrightarrow{\alpha : \text{ptr}}; x_1 : \mathcal{B}(\tau_1), \cdots, x_n : \mathcal{B}(\tau_n), x_0 : \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\mathcal{B}(\tau)}) \to \text{void} \vdash_{C_2} \text{let}\, x = \langle x_0 \rangle \,\text{in}\, \mathcal{B}(e)$
By (TD-FIX), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathit{fixcode}\, x_0[\overrightarrow{\alpha : \text{ptr}}](\overrightarrow{x : \mathcal{B}(\tau)}).\text{let}\, x = \langle x_0 \rangle \,\text{in}\, \mathcal{B}(e) : \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\tau}) \to \text{void}$
By (TD-TUPLE), $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle \mathit{fixcode}\, x_0[\overrightarrow{\alpha : \text{ptr}}](\overrightarrow{x : \mathcal{B}(\tau)}).\text{let}\, x = \langle x_0 \rangle \,\text{in}\, \mathcal{B}(e) \rangle : \langle \forall[\overrightarrow{\alpha : \text{ptr}}].(\overrightarrow{\tau}) \to \text{void} \rangle$

18. $e = \text{pack}[\tau, v] \,\text{as}\, \exists \alpha : K.\tau'$, $\mathcal{B}(e) = \text{pack}[\mathcal{B}(\tau), \mathcal{B}(v)] \,\text{as}\, \exists \alpha : \text{ptr}.\mathcal{B}(\tau')$
From (TC-PACK), $\quad \Delta \vdash_C \tau \quad , \Delta; \Gamma \vdash_C v : \tau'[\tau/\alpha]$
By induction, we get $\quad \mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr} \quad , \mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \mathcal{B}(\tau'[\tau/\alpha])$
By lemma 1, we can write $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \mathcal{B}(\tau')[\mathcal{B}(\tau)/\alpha]$
By (TD-PACK) rule, $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(e) : \exists \alpha : \text{ptr}.\mathcal{B}(\tau')$

9.2 Converting λ^{C_2} to λ^{flat}

Translating λ^{C_2} to λ^{low} requires some effort to break up nested tuple allocations, so we define a "flattened" variant of λ^{C_2} called λ^{flat} to factor out this effort ahead of time. λ^{flat} is strictly a subset of λ^{C_2}: they share the same kind and type systems, and any λ^{flat} expression is also a λ^{C_2} expression:

$K =$ primitive $|$ ptr

$\tau \; = \; \alpha \,|\, \mathrm{int} \,|\, \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \mathrm{void} \,|\, \langle \overrightarrow{\tau} \rangle \,|\, \exists \alpha : K.\tau$

$e \; = \; x(\overrightarrow{x}) \,|\, \mathrm{if0}(x,\, e_1,\, e_2) \,|\, \mathrm{halt}[\tau]x \,|\, \mathrm{let}\, x = t \,\mathrm{in}\, e \,|\, \mathrm{let}\, x_0 = \pi_i x \,\mathrm{in}\, e$
$\quad |\, \mathrm{let}\,[\alpha, x] = \mathrm{unpack}\, x_0 \,\mathrm{in}\, e \,|\, \mathrm{let}\, x_0 = \langle \overrightarrow{x} \rangle \,\mathrm{in}\, e$

$t = x \,|\, i \,|\, x[\tau] \,|\, \mathrm{pack}[\tau,\, x] \,\mathrm{as}\, \exists \alpha : K.\tau' \,|\, x_1 \, p \, x_2$
$\quad |\, fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e$

$p = + \,|\, - \,|\, *$

Type Checking Rules For $\lambda^{\mathbf{Flat}}$

(KF-VAR) $\quad \Delta, \alpha : K \vdash \alpha : K$

(KF-INT) $\quad \Delta \vdash \mathrm{int{:}primitive}$

(KF-ALL) $\quad \dfrac{\Delta,\, \overrightarrow{\alpha : K} \vdash \tau_i : K_i}{\Delta \vdash \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \mathrm{void} : \mathrm{primitive}}$

(KF-TUPLE) $\quad \dfrac{\Delta \vdash \tau_i : K_i}{\Delta \vdash \langle \overrightarrow{\tau} \rangle : \mathrm{ptr}}$

(KF-SOME) $\quad \dfrac{\Delta,\, \alpha : K \vdash \tau : K'}{\Delta \vdash (\exists \alpha : K.\tau) : K'}$

(TF-APP) $\quad \dfrac{\Delta; \Gamma \vdash x : (\tau_1, \cdots, \tau_n) \rightarrow \mathrm{void} \qquad \Delta; \Gamma \vdash x_i : \tau_i}{\Delta; \Gamma \vdash x(\overrightarrow{x})}$

(TF-IF) $\quad \dfrac{\Delta; \Gamma \vdash x : \mathrm{int} \quad \Delta; \Gamma \vdash e_1 \;\; \Delta; \Gamma \vdash e_2}{\Delta; \Gamma \vdash \mathrm{if0}(x,\, e_1,\, e_2)}$

(TF-HALT) $\quad \dfrac{\Delta; \Gamma \vdash x : \tau}{\Delta; \Gamma \vdash \mathrm{halt}[\tau]x}$

(TF-LET) $\quad \dfrac{\Delta; \Gamma \vdash t : \tau \quad \Delta; \Gamma,\, x : \tau \vdash e}{\Delta; \Gamma \vdash \mathrm{let}\, x = t \,\mathrm{in}\, e} \; (x \notin \Gamma)$

(TF-PROJECT) $\quad \dfrac{\Delta; \Gamma \vdash x : \langle \tau_1, \cdots, \tau_n \rangle \quad \Delta; \Gamma,\, x_0 : \tau_i \vdash e}{\Delta; \Gamma \vdash \mathrm{let}\, x_0 = \pi_i x \,\mathrm{in}\, e} \; (x \notin \Gamma \wedge 1 \leq i \leq n)$

(TF-UNPACK) $\quad \dfrac{\Delta; \Gamma \vdash x_0 : \exists \alpha : K.\tau \quad (\Delta, \alpha : K); (\Gamma, x : \tau) \vdash e}{\Delta; \Gamma \vdash \mathrm{let}\,[\alpha, x] = \mathrm{unpack}\, x_0 \,\mathrm{in}\, e} \; (x \notin \Gamma \wedge \alpha \notin \Delta)$

(TF-TUPLE) $\quad \dfrac{\Delta; \Gamma \vdash x_i : \tau_i \quad \Delta; \Gamma,\, x_0 : \langle \tau_1, \cdots, \tau_n \rangle \vdash e}{\Delta; \Gamma \vdash \mathrm{let}\, x_0 = \langle \overrightarrow{x} \rangle \,\mathrm{in}\, e} \; (x_0 \notin \Gamma)$

(TF-OP) $\quad \dfrac{\Delta; \Gamma \vdash x_1 : \mathrm{int} \;\; \Delta; \Gamma \vdash x_2 : \mathrm{int}}{\Delta; \Gamma \vdash x_1 \, p \, x_2 \;{}_{:\mathrm{int}}}$

(TF-VAR) $\quad \Delta; \Gamma, x : \tau \vdash x : \tau$

(TF-INT) $\quad \Delta; \Gamma \vdash i : \mathrm{int}$

(TF-TAPP) $\quad \dfrac{\Delta \vdash \tau_0 : K \quad \Delta; \Gamma \vdash x : \forall[\alpha : K, \overrightarrow{\beta : K'}].(\overrightarrow{\tau}) \rightarrow \mathrm{void}}{\Delta; \Gamma \vdash x[\tau_0] : (\forall[\overrightarrow{\beta : K'}].(\overrightarrow{\tau}[\tau_0/\alpha]) \rightarrow \mathrm{void})}$

$$\text{(TF-PACK)} \quad \frac{\Delta \vdash \tau : K \quad \Delta; \Gamma \vdash x : \tau'[\tau/\alpha]}{\Delta; \Gamma \vdash \text{pack}[\tau,\, x]\,\text{as}\,\exists\alpha : K.\tau' :\ \exists\alpha : K.\tau'}$$

$$\text{(TF-FIX)} \quad \frac{\overrightarrow{\alpha : K} \vdash \tau_i : K'_i \quad \overrightarrow{\alpha : K};\, x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \text{void},\, x_1 : \tau_1,\, \cdots,\, x_n : \tau_n \vdash e}{\Delta; \Gamma \vdash\ \mathit{fixcode}\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e \ \ : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \text{void}}$$

$$\text{(TF-ENV)} \quad \frac{\Delta \vdash \tau_i : K_i}{\Delta \vdash\ \{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\}}$$

To aid the conversion, define a declaration d as:

$d\ = x = t \,|\, x = \pi_i x' \,|\, x = \langle \overrightarrow{x} \rangle$

Say that $(\text{let}\, d_1, \ldots, d_n \,\text{in}\, e)$ is short for $(\text{let}\, d_1 \,\text{in} \ldots \text{let}\, d_n \,\text{in}\, e)$.
Also define abbreviations for typing declarations and typing terms prefixed by sequences of declarations:

$\Delta; \Gamma \vdash (x = t) : (x \mapsto \tau) \Leftrightarrow \Delta; \Gamma \vdash t : \tau$

$\Delta; \Gamma \vdash (x = \pi_i x') : (x \mapsto \tau) \Leftrightarrow \Delta; \Gamma \vdash x' : \langle \tau_1, \cdots, \tau_n \rangle \quad \tau = \tau_i$

$\Delta; \Gamma \vdash (x = \langle \overrightarrow{x} \rangle) : (x \mapsto \langle \overrightarrow{\tau} \rangle) \Leftrightarrow \Delta; \Gamma \vdash x_i : \tau_i$

$$\begin{aligned} \Delta; \Gamma \vdash d_1, \ldots, d_n, t : \tau \quad \Leftrightarrow \quad & (\Delta; \Gamma \vdash d_1 : (x_1 \mapsto \tau_1)) \\ & \wedge (\Delta; \Gamma, (x_1 \mapsto \tau_1) \vdash d_2, \ldots, d_n, t : \tau) \end{aligned}$$

The conversion from λ^{C_2} to λ^{flat} changes expressions e to expressions $\mathcal{F}(e)$ and
changes values v to declarations followed by terms $\mathcal{F}(v) = \overrightarrow{d}, t$. It does not modify types or kinds.

$\mathcal{F}(x) = x$

$\mathcal{F}(i) = i$

$\mathcal{F}(v[\tau]) = \overrightarrow{d},\, x = t,\, x[\tau]$
where $\mathcal{F}(v) = \overrightarrow{d},\, t$

$\mathcal{F}(\langle \overrightarrow{v} \rangle) = \overrightarrow{d},\, \overrightarrow{x = t},\, x_0 = \langle \overrightarrow{x} \rangle,\, x_0$
where $\mathcal{F}(v_i) = \overrightarrow{d_i},\, t_i$

$\mathcal{F}(\pi_i v) = \overrightarrow{d},\, x_1 = t,\, x_2 = \pi_i x_1,\, x_2$
where $\mathcal{F}(v) = \overrightarrow{d},\, t$

$\mathcal{F}(\text{pack}[\tau, v]\, \mathit{as}\, \exists\alpha : K.\tau') = \overrightarrow{d},\, x = t,\, \text{pack}[\tau, x]\, \mathit{as}\, \exists\alpha : K.\tau'$
where $\mathcal{F}(v) = \overrightarrow{d},\, t$

$\mathcal{F}(\mathit{fixcode}\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e) = \mathit{fixcode}\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).\mathcal{F}(e)$

$\mathcal{F}(v(\overrightarrow{v})) = \text{let}\ \overrightarrow{d},\, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, x = t, \overrightarrow{x = t}\, \text{in}\, x(\overrightarrow{x})$
where $\mathcal{F}(v_i) = \overrightarrow{d_i},\, t_i,\ \ \mathcal{F}(v) = \overrightarrow{d},\, t$

$\mathcal{F}(\text{if0}(v,\, e_1,\, e_2)) = \text{let}\ \overrightarrow{d}, x = t\, \text{in}\, \text{if0}(x,\, \mathcal{F}(e_1),\, \mathcal{F}(e_2))$

where $\mathcal{F}(v) = \overrightarrow{d}, t$

$\mathcal{F}(\text{halt}[\tau]v) = \text{let } \overrightarrow{d}, x = t \text{ in halt}[\tau]x$
where $\mathcal{F}(v) = \overrightarrow{d}, t$

$\mathcal{F}(\text{let } x = v \text{ in } e) = \text{let } \overrightarrow{d} \text{ in let } x = t \text{ in } \mathcal{F}(e)$
where $\mathcal{F}(v) = \overrightarrow{d}, t$

$\mathcal{F}(\text{let } x_0 = v_1\, p\, v_2 \text{ in } e) = \text{let } \overrightarrow{d_1},\ \overrightarrow{d_2},\ x_1 = t_1,\ x_2 = t_2 \text{ in let } x_0 = x_1\, p\, x_2 \text{ in } \mathcal{F}(e)$
where $\mathcal{F}(v_1) = \overrightarrow{d_1},\ t_1,\ \mathcal{F}(v_2) = \overrightarrow{d_2},\ t_2$

$\mathcal{F}(\text{let } [\alpha, x] = \text{unpack } v \text{ in } e) = \text{let } \overrightarrow{d}, x_0 = t \text{ in let } [\alpha, x] = \text{unpack } x_0 \text{ in } \mathcal{F}(e)$
where $\mathcal{F}(v) = \overrightarrow{d}, t$

Correctness :

1. If $\Delta \vdash_{C_2} \tau : K$, then $\Delta \vdash_F \tau : K$
2. If $\Delta \vdash_{C_2} \Gamma$, then $\Delta \vdash_F \Gamma$
3. If $\Delta; \Gamma \vdash_{C_2} e$, then $\Delta; \Gamma \vdash_F \mathcal{F}(e)$
4. If $\Delta; \Gamma \vdash_{C_2} v : \tau$, then $\Delta; \Gamma \vdash_F \mathcal{F}(v) : \tau$

Proof :
Prove by induction on typing derivations.

1. τ and Γ
The rules for$\vdash_F$ are the same as for $\vdash_{C_2}$, so $\Delta \vdash_{C_2} \tau : K$ implies $\Delta \vdash_F \tau : K$,
and $\Delta \vdash_{C_2} \Gamma$ implies $\Delta \vdash_F \Gamma$

2. $v = x$: the TD-VAR and TF-VAR rules are identical

3. $v = i$, $\mathcal{F}(v) = i$
Because $\Delta \vdash_{C_2} i : \text{int}$, then $\Delta \vdash_F \mathcal{F}(v) : \text{int}$

4. $v_0 = v[\tau_0]$, $\mathcal{F}(v[\tau_0]) = \overrightarrow{d},\ x = t,\ x[\tau_0]$, where $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-TAPP), we know $\Delta \vdash_{C_2} \tau_0 : K$,
and $\Delta; \Gamma \vdash_{C_2} v : \forall[\alpha : K, \overrightarrow{\beta : K'}].(\overrightarrow{\tau}) \rightarrow \text{void}$
By induction, $\Delta \vdash_F \tau_0 : K$, and $\Delta; \Gamma \vdash_F \overrightarrow{d}, t : \forall[\alpha : K, \overrightarrow{\beta : K'}].(\overrightarrow{\tau}) \rightarrow \text{void}$
By (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d},\ x = t,\ x : \forall[\alpha : K, \overrightarrow{\beta : K'}].(\overrightarrow{\tau}) \rightarrow \text{void}$
By (TF-TAPP), $\Delta; \Gamma \vdash_F \overrightarrow{d},\ x = t,\ x[\tau_0] : (\forall[\overrightarrow{\beta : K'}].(\overrightarrow{\tau}[\tau_0/\alpha]) \rightarrow \text{void})$

5. $v = \langle \overrightarrow{v} \rangle$, $\mathcal{F}(\langle \overrightarrow{v} \rangle) = \overrightarrow{d},\ \overrightarrow{x = t},\ x_0 = \langle \overrightarrow{x} \rangle,\ x_0$; $\quad \mathcal{F}(v_i) = \overrightarrow{d_i},\ t_i$
From (TD-TAPP), we know $\Delta; \Gamma \vdash_{C_2} v_i : \tau_i$.
By induction, $\Delta; \Gamma \vdash_F \overrightarrow{d_i},\ t_i : \tau_i$.
By weakening, $\Delta; \Gamma \vdash_F \overrightarrow{d},\ t_i : \tau_i$.
By weakening and (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d},\ \overrightarrow{x = t},\ x_i : \tau_i$.
By (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d},\ \overrightarrow{x = t},\ x_0 = \langle \overrightarrow{x} \rangle,\ x_0 : \langle \overrightarrow{\tau} \rangle$

6. $v = \text{pack}[\tau, v]\, as\, \exists\alpha : K.\tau'$,
$\mathcal{F}(\text{pack}[\tau, v]\, as\, \exists\alpha : K.\tau') = \overrightarrow{d},\ x = t,\ \text{pack}[\tau, x]\, as\, \exists\alpha : K.\tau'$; $\ \mathcal{F}(v) = \overrightarrow{d}, t$

From (TD-PACK), we know $\Delta \vdash_{C_2} \tau : K$, $\Delta; \Gamma \vdash_{C_2} v : \tau'[\tau/\alpha]$.
By induction, $\Delta \vdash_F \tau : K$, $\Delta; \Gamma \vdash_F \overrightarrow{d}, t : \tau'[\tau/\alpha]$.
By (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d}, x = t, x : \tau'[\tau/\alpha]$.
By (TF-PACK), $\Delta; \Gamma \vdash_F \overrightarrow{d}, x = t, \text{pack}[\tau, x] \,\text{as}\, \exists\alpha : K.\tau' : \exists\alpha : K.\tau'$

7. $v = fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e$, $\mathcal{F}(v) = fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).\mathcal{F}(e)$
From (TD-FIX), $\overrightarrow{\alpha : K} \vdash_{C_2} \tau_i : K'_i$,
and $\overrightarrow{\alpha : K}; x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \text{void}, x_1 : \tau_1, \cdots, x_n : \tau_n \vdash_{C_2} e$
By induction, we get $\overrightarrow{\alpha : K} \vdash_F \tau_i : K'_i$,
and $\overrightarrow{\alpha : K}; x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \text{void}, x_1 : \tau_1, \cdots, x_n : \tau_n \vdash_F \mathcal{F}(e)$
By (TF-FIX) rule, $\Delta; \Gamma \vdash_F fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).\mathcal{F}(e) : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \text{void}$

8. $e = v(\overrightarrow{v})$, $\mathcal{F}(e) = \text{let}\, \overrightarrow{d}, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, x = t, \overrightarrow{x = t} \,\text{in}\, x(\overrightarrow{x})$; $\mathcal{F}(v_i) = \overrightarrow{d_i}, t_i$, $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-APP), we can get $\Delta; \Gamma \vdash_{C_2} v : (\tau_1, \cdots, \tau_n) \to \text{void}$ $\Delta; \Gamma \vdash_{C_2} v_i : \tau_i$.
By induction, $\Delta; \Gamma \vdash_F \overrightarrow{d}, t : (\tau_1, \cdots, \tau_n) \to \text{void}$ $\Delta; \Gamma \vdash_F \overrightarrow{d_i}, t_i : \tau_i$.
By weakening, $\Delta; \Gamma \vdash_F \overrightarrow{d}, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, t : (\tau_1, \cdots, \tau_n) \to \text{void}$ $\Delta; \Gamma \vdash_F \overrightarrow{d}, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, t_i : \tau_i$.
By weakening and (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d}, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, x = t, \overrightarrow{x = t}, x : (\tau_1, \cdots, \tau_n) \to \text{void}$
and $\Delta; \Gamma \vdash_F \overrightarrow{d}, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, , x = t, \overrightarrow{x = t}, x : \tau_i$.
By (TF-LET/TF-PROJECT/TF-TUPLE) and (TF-APP), $\Delta; \Gamma \vdash_F \text{let}\, \overrightarrow{d}, \overrightarrow{d_1}, \cdots, \overrightarrow{d_n}, x = t, \overrightarrow{x = t} \,\text{in}\, x(\overrightarrow{x})$.

9. $e = \text{if0}(v, e_1, e_2)$, $\mathcal{F}(e) = \text{let}\, \overrightarrow{d}, x = t \,\text{in}\, \text{if0}(x, \mathcal{F}(e_1), \mathcal{F}(e_2))$; $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-IF), we know $\Delta; \Gamma \vdash_{C_2} v : \text{int}$, $\Delta; \Gamma \vdash_{C_2} e_1$ $\Delta; \Gamma \vdash_{C_2} e_2$.
By induction, $\Delta; \Gamma \vdash_F \overrightarrow{d}, t : \text{int}$, $\Delta; \Gamma \vdash_F \mathcal{F}(e_1)$, $\Delta; \Gamma \vdash_F \mathcal{F}(e_2)$.
By (TF-VAR) and weakening, $\Delta; \Gamma \vdash_F \overrightarrow{d}, x = t, x : \text{int}$.
By weakening, (TF-LET/TF-PROJECT/TF-TUPLE), and (TF-IF), $\Delta; \Gamma \vdash_F \text{let}\, \overrightarrow{d}, x = t \,\text{in}\, \text{if0}(x, \mathcal{F}(e_1), \mathcal{F}(e_2))$

10. $e = \text{halt}[\tau]v$, $\mathcal{F}(\text{halt}[\tau]v) = \text{let}\, \overrightarrow{d}, x = t \,\text{in}\, \text{halt}[\tau]x$; $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-HALT), we know $\Delta; \Gamma \vdash_{C_2} v : \tau$.
By induction, $\Delta; \Gamma \vdash_F \overrightarrow{d}, t : \tau$.
By (TF-VAR) and weakening, $\Delta; \Gamma \vdash_F \overrightarrow{d}, x = t, x : \tau$.
By (TF-LET/TF-PROJECT/TF-TUPLE) and (TF-HALT), $\Delta; \Gamma \vdash_F \text{let}\, \overrightarrow{d}, x = t \,\text{in}\, \text{halt}[\tau]x$

11. $e = \text{let}\, x = v \,\text{in}\, e$, $\mathcal{F}(\text{let}\, x = v \,\text{in}\, e) = \text{let}\, \overrightarrow{d} \,\text{in}\, \text{let}\, x = t \,\text{in}\, \mathcal{F}(e)$; $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-SUB) rule, $\Delta; \Gamma \vdash_{C_2} v : \tau$, $\Delta; \Gamma, x : \tau \vdash_{C_2} e$.
By induction, $\Delta; \Gamma \vdash_{C_2} \overrightarrow{d}, t : \tau$, $\Delta; \Gamma, x : \tau \vdash_{C_2} \mathcal{F}(e)$.
By weakening and (TF-LET/TF-PROJECT/TF-TUPLE), $\Delta; \Gamma \vdash \mathcal{F}(\text{let}\, x = v \,\text{in}\, e)$

12. $v_0 = \pi_i v$, $\mathcal{F}(v_0) = \overrightarrow{d}, x_1 = t, x_2 = \pi_i x_1, x_2$; $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-PROJECT) rule, $\Delta; \Gamma \vdash_{C_2} v : \langle \tau_1, \cdots, \tau_n \rangle$.
By induction, $\Delta; \Gamma \vdash_F \overrightarrow{d}, t : \langle \tau_1, \cdots, \tau_n \rangle$.
By (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d}, x_1 = t, x_1 : \langle \tau_1, \cdots, \tau_n \rangle$.
By (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d}, x_1 = t, x_2 = \pi_i x_1, x_2 : \tau_i$.

13. $e_0 = \text{let}\, x_0 = v_1\, p\, v_2 \,\text{in}\, e$, $\mathcal{F}(e_0) = \text{let}\, \overrightarrow{d_1}, \overrightarrow{d_2}, x_1 = t_1, x_2 = t_2 \,\text{in}\, \text{let}\, x_0 = x_1\, p\, x_2 \,\text{in}\, F(e)$, $\mathcal{F}(v_1) = \overrightarrow{d_1}, t_1$, $\mathcal{F}(v_2) = \overrightarrow{d_2}, t_2$
From (TD-OP), $\Delta; \Gamma \vdash_{C_2} v_1 : \text{int}$ $\Delta; \Gamma \vdash_{C_2} v_2 : \text{int}$, $\Delta; \Gamma, x_0 : \text{int} \vdash_{C_2} e$.
By induction, $\Delta; \Gamma \vdash_F \overrightarrow{d_1}, t_1 : \text{int}$ $\Delta; \Gamma \vdash_F \overrightarrow{d_2}, t_2 : \text{int}$

$\Delta; \Gamma, x_0 : \text{int} \vdash_F \mathcal{F}(e)$.
By weakening, $\Delta; \Gamma \vdash_F \overrightarrow{d_1}, \overrightarrow{d_2}, t_1 : \text{int} \;\; \Delta; \Gamma \vdash_F \overrightarrow{d_1}, \overrightarrow{d_2}, t_2 : \text{int}$.
By weakening and (TF-VAR), $\Delta; \Gamma \vdash_F \overrightarrow{d_1}, \overrightarrow{d_2}, x_1 = t_1, x_2 = t_2, x_1 : \text{int} \;\; \Delta; \Gamma \vdash_F \overrightarrow{d_1}, \overrightarrow{d_2}, x_1 = t_1, x_2 = t_2, x_2 : \text{int}$.
By (TF-LET/TF-PROJECT/TF-TUPLE), weakening, and (TF-OP), $\Delta; \Gamma \vdash_F \mathcal{F}(e_0)$.

14. $e_0 = \text{let}\,[\alpha, x] = \text{unpack}\, v \,\text{in}\, e$, $\mathcal{F}(e_0) = \text{let}\, \overrightarrow{d}, x_0 = t \,\text{in}\, \text{let}\,[\alpha, x] = \text{unpack}\, x_0 \,\text{in}\, \mathcal{F}(e)$; $\mathcal{F}(v) = \overrightarrow{d}, t$
From (TD-UNPACK), $\Delta; \Gamma \vdash v : \exists \alpha : K.\tau$, $(\Delta, \alpha : K); (\Gamma, x : \tau) \vdash e$
By induction, $\Delta; \Gamma \vdash \overrightarrow{d}, t : \exists \alpha : K.\tau$, $(\Delta, \alpha : K); (\Gamma, x : \tau) \vdash \mathcal{F}(e)$
By weakening, $\Delta; \Gamma \vdash \overrightarrow{d}, x_0 = t, x_0 : \exists \alpha : K.\tau$.
By (TF-LET/TF-PROJECT/TF-TUPLE), weakening, and (TF-UNPACK), $\Delta; \Gamma \vdash \mathcal{F}(e_0)$

9.3 Converting λ^{flat} to λ^{low}

Kind conversion

$K_R = \text{int} \to \text{int} \to \text{int} \to \dot{1}$
$\mathcal{A}(\text{primitive}) = \dot{1}$
$\mathcal{A}(\text{ptr}) = \dot{1}$
$\mathcal{A}_{tyarg}(\text{primitive}) = \dot{1}$
$\mathcal{A}_{tyarg}(\text{ptr}) = \text{int} \to K_R \to \text{int} \to \text{int} \to \dot{0}$

Type conversion

We assume that source types have been checked for well-formedness, and we rely on the resulting kind annotations to guide the translation.

$\mathcal{A}(\tau : \text{primitive}) = \lambda R : K_R.\lambda E : \text{int}.\mathcal{A}_{prim}(\tau)$
$\mathcal{A}_{tyarg}(\tau : \text{primitive}) = \mathcal{A}_{prim}(\tau)$
$\mathcal{A}(\tau : \text{ptr}) = \lambda R : K_R.\lambda E : \text{int}.\exists L : \text{int}; L \geq 0.(GcPtr(\mathcal{A}_{ptr}(\tau)\, L\, R)\, E)$
$\mathcal{A}_{tyarg}(\tau : \text{ptr}) = \mathcal{A}_0(\tau : \text{ptr})$
$\mathcal{A}_{ptr}(\tau) = \mu\alpha : \mathcal{A}_{tyarg}(\text{ptr}).\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}. \cdot \langle \mathcal{A}_0(\tau)\, L\, R\, E\, I,\ GcCoerce\, L\, R\, (\alpha\, L\, R)\rangle$

$\mathcal{A}_0(\alpha) = \alpha$
$\mathcal{A}_0(\exists \alpha : K.\tau) = \lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\exists \alpha : \mathcal{A}_{tyarg}(K);\ L \geq 1.(\mathcal{A}_{ptr}(\tau)\,(L-1)\, R\, E\, I)$
$\mathcal{A}_0(\langle \tau_1 : K_1, \cdots, \tau_n : K_n\rangle) = \lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.$
$\quad \cdot\langle \exists F : \text{int} \to \text{int} \to \dot{1}\ . \cdot \langle \text{Eq}(F\, 0, \mathcal{A}(\tau_1)\, R), \ldots, \text{Eq}(F\,(n-1), \mathcal{A}(\tau_n)\, R),\ (GcWord\, R\, E\, I\, (GcHdr\, R\, F\, Tag\, n))\rangle,$
$\qquad (GcWord\, R\, E\, (I+1)\, (\mathcal{A}(\tau_1)\, R)), \ldots, (GcWord\, R\, E\, (I+n)\, ((\mathcal{A}(\tau_n)\, R)))\rangle$
$\quad where\, Tag = (\sum_{i=1}^{n} 2^{i-1} * tag(K_i)),\ where\, tag(\text{prim}) = 0\, and\, tag(\text{ptr}) = 1$

$\mathcal{A}_{prim}(\exists \alpha : K.\tau) = \exists \alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{prim}(\tau)$
$\mathcal{A}_{prim}(\alpha) = \alpha$
$\mathcal{A}_{prim}(\text{int}) = \tau_{int}$
$\mathcal{A}_{prim}(\forall[\langle\overrightarrow{\alpha : K}\rangle].\langle\overrightarrow{\tau}\rangle \to \text{void}) = \forall\langle\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}\rangle.\forall R : K_R.\forall E : \text{int}.\hat{}\,\langle \tau_g(R, E),\ \langle\overrightarrow{\mathcal{A}(\tau)\, R\, E}\rangle\rangle \to \tau_{halt}$

$\tau_{int} = \exists I : \text{int}.Int(I)$

Environment conversion:

$$\mathcal{A}_{tyarg}(\{\alpha_1 \mapsto K_1, \ldots, \alpha_n \mapsto K_n\}) = \{\alpha_1 \mapsto \mathcal{A}_{tyarg}(K_1), \ldots, \alpha_n \mapsto \mathcal{A}_{tyarg}(K_n)\}$$

$$\mathcal{A}(\{x_1 \mapsto \tau_1, \ldots, x_n \mapsto \tau_n\})\,R\,E = \{x_1 \mapsto (\mathcal{A}(\tau_1)\,R\,E), \ldots, x_n \mapsto (\mathcal{A}(\tau_n)\,R\,E)\}$$

Expression conversion:

$\mathcal{A}(x)\,R\,E\,g = x$
$\mathcal{A}(i)\,R\,E\,g = \mathrm{pack}(i,\, i)\,\mathrm{as}\,\tau_{int}$
$\mathcal{A}(x_1\,p\,x_2)\,R\,E\,g = OP_p \cdot \langle x_1,\, x_2\rangle,$

$\mathcal{A}(x[\tau])\,R\,E\,g = x[\mathcal{A}_{tyarg}(\tau)]$

$\mathcal{A}(\mathrm{pack}[\tau,\, x]\,\mathrm{as}\,\exists\beta : K.\tau' : \mathrm{primitive})\,R\,E\,g = \mathrm{pack}\,\mathcal{A}_{tyarg}(\tau),\, x\,\mathrm{as}\,\exists\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}(\tau')\,R\,E$
$\mathcal{A}(\mathrm{pack}[\tau,\, x]\,\mathrm{as}\,\exists\beta : K.\tau' : \mathrm{ptr})\,R\,E\,g = \mathit{ptr\_pack}_K\,R\,E\,(\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau'))\mathcal{A}_{tyarg}(\tau)x$

$\mathcal{A}(\mathit{fixcode}\,x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e)\,R\,E\,g \;=\; \mathit{fix}\,x \;:\; \forall\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\forall R' \;:\; K_R.\forall E' \;:\; \mathrm{int}.{}^\wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\,R'\,E'}\rangle \;\to\; \tau_{halt}(R').$
$\quad\Lambda\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\Lambda R' : K_R.\Lambda E' : \mathrm{int}.\lambda y :{}^\wedge\, \langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\,R'\,E'}\rangle \to \mathrm{let}\,\langle g',\, \overrightarrow{x}\rangle = y\,\mathrm{in}\,\mathcal{A}(e)\,R'\,E'\,g'$

$\mathcal{A}(x(\overrightarrow{x}))\,R\,E\,g = x\,R\,E\,{}^\wedge\langle g,\, \overrightarrow{x}\rangle$
$\mathcal{A}(\mathrm{if0}(x,\, e_1,\, e_2))\,R\,E\,g = \mathrm{unpack}\,\beta, x_1 = \; x\,\mathrm{in}\,\mathrm{if}\,x_1 = 0\,\mathrm{then}\,\mathcal{A}(e_1)\,R\,E\,g\;\;\mathrm{else}\,\mathcal{A}(e_2)\,R\,E\,g$
$\mathcal{A}(\mathrm{halt}[\tau]x)\,R\,E\,g = \mathit{halt}\,R\,E\,(\mathcal{A}(\tau)\,R\,E)\,{}^\wedge\langle g,\, x\rangle$
$\mathcal{A}(\mathrm{let}\,x = t\,\mathrm{in}\,e)\,R\,E\,g = \mathrm{let}\,x = \mathcal{A}(t)\,R\,E\,g\,\mathrm{in}\,\mathcal{A}(e)\,R\,E\,g$

$\mathcal{A}(\mathrm{let}\,x_0 = \pi_i(x_1 : \langle\overrightarrow{\tau : K}\rangle)\,\mathrm{in}\,e)\,R\,E\,g = \mathrm{let}\,\langle g', x_0\rangle = \mathit{obj\_load}_{\overrightarrow{K},i}\,R\,E\,\overrightarrow{\mathcal{A}_{tyarg}(\tau)}\,{}^\wedge\langle g, x_1\rangle\,\mathrm{in}\,\mathcal{A}(e)\,R\,E\,g'$

$\mathcal{A}(\mathrm{let}\,[\alpha : K, x_0 : \tau_0 : \mathrm{primitive}] = \mathrm{unpack}\,x_1 : \exists\beta : K.\tau\,\mathrm{in}\,e)\,R\,E\,g = \mathrm{unpack}\,\alpha,\, x_0 = x_1\,\mathrm{in}\,\mathcal{A}(e)\,R\,E\,g$
$\mathcal{A}(\mathrm{let}\,[\alpha : K, x_0 : \tau_0 : \mathrm{ptr}] = \mathrm{unpack}\,x_1 : \exists\beta : K.\tau\,\mathrm{in}\,e)\,R\,E\,g =$
$\quad\mathrm{unpack}\,\alpha, x_0 = \mathit{ptr\_unpack}_K R\,E\,(\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau))\,x_1\,\mathrm{in}\,\mathcal{A}(e)\,R\,E\,g$

$\mathcal{A}(\mathrm{let}\,x_0 = \langle\overrightarrow{x : \tau_x}\rangle\,\mathrm{in}\,e)\,R\,E\,g = \mathrm{unpack}\,E',\, z = \mathit{alloc}_{\overrightarrow{K},n} R\,E\,\overrightarrow{\mathcal{A}_{tyarg}(\tau_x)}\,\overrightarrow{\mathcal{A}_{tyarg}(\tau_y)}\,{}^\wedge\langle g, \overrightarrow{x}, \overrightarrow{y}\rangle\,\mathrm{in}$
$\quad\mathrm{let}\,\langle g', x_0, \overrightarrow{y}\rangle = z\,\mathrm{in}\,\mathcal{A}(e)\,R\,E'\,g'; \quad \mathrm{where}\ \{y_1, \cdots, y_n\} = \{y : \tau_y : \mathrm{ptr} \in FV(e)\} - \{x_0\}$

Types of λ^{low} Run-time System Functions:

To simplify the forms of the types below, we treat type application $(\alpha\,\beta)$ as if it were a λ^{flat} type of kind ptr, so that we can use our definitions of $\mathcal{A}(\tau)$ for $(\alpha\,\beta)$, letting $\mathcal{A}_0(\alpha\,\beta) = \alpha\,\beta$ (a slight notational abuse).

$K_W = \mathrm{int} \to \dot{1}$
$K_A = \mathrm{int} \to \mathrm{int} \to \dot{0}$
$K_F = \mathrm{int} \to \mathrm{int} \to \dot{1}$
$K_R = \mathrm{int} \to \mathrm{int} \to \mathrm{int} \to \dot{1}$
$\tau_{int} = \exists I : \mathrm{int}.Int(I)$
$\emptyset; R : K_R,\, E : \mathrm{int} \vdash_L \tau_g(R,\, E) : K_g\;,\;\; (K_g = \hat{i}_g)$
$\emptyset; \emptyset \vdash_L \tau_{halt} : K_{halt}\;,\;\; (K_{halt} = \hat{i}_{halt})$

$\emptyset;\emptyset \vdash_L GcWord : K_R \to \text{int} \to \text{int} \to K_W \to \dot{0}$
$\emptyset;\emptyset \vdash_L GcPtr : K_A \to \text{int} \to \dot{1}$
$\emptyset;\emptyset \vdash_L GcCoerce : \text{int} \to K_R \to K_A \to \dot{0}$
$\emptyset;\emptyset \vdash_L GcHdr : K_R \to K_F \to \text{int} \to \text{int} \to \text{int} \to \dot{1}$
$\emptyset;\emptyset;\emptyset;\emptyset; true; \infty \vdash_L halt : \forall R : K_R.\forall E : \text{int}.\forall \alpha : \dot{1}\ .^\wedge \langle \tau_g(R,E),\ \alpha \rangle \to \tau_{halt}$

$\emptyset;\emptyset;\emptyset;\emptyset; true; \infty \vdash_L OP_p : \cdot \langle \tau_{int},\ \tau_{int} \rangle \to \tau_{int}$

$\emptyset;\emptyset;\emptyset;\emptyset; true; \infty \vdash_L$
$ptr\_pack_K\ : \forall R : K_R.\forall E : \text{int}.\forall \alpha : \mathcal{A}_{tyarg}(K) \to \mathcal{A}_{tyarg}(\text{ptr}).\forall \beta : \mathcal{A}_{tyarg}(K).\mathcal{A}(\alpha\,\beta)\,R\,E \to \mathcal{A}(\exists \gamma : K.(\alpha\,\gamma))\,R\,E$

$\emptyset;\emptyset;\emptyset;\emptyset; true; \infty \vdash_L$
$ptr\_unpack_K\ : \forall R : K_R.\forall E : \text{int}.\forall \alpha : \mathcal{A}_{tyarg}(K) \to \mathcal{A}_{tyarg}(\text{ptr}).\mathcal{A}(\exists \beta : K.(\alpha\,\beta))\,R\,E \to \exists \beta : \mathcal{A}_{tyarg}(K).(\mathcal{A}(\alpha\,\beta)\,R\,E)$

$\emptyset;\emptyset;\emptyset;\emptyset; true; \infty \vdash_L$
$obj\_load_{\overrightarrow{K},i}\ : \forall R : K_R.\forall E : \text{int}.\forall \overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.^\wedge \langle \tau_g(R,E),\ \mathcal{A}(\langle \overrightarrow{\alpha : K} \rangle)\,R\,E \rangle \to^\wedge\ \langle \tau_g(R,E),\ \mathcal{A}(\alpha_i : K_i)\,R\,E \rangle$

$\emptyset;\emptyset;\emptyset;\emptyset; true; \infty \vdash_L$
$alloc_{\overrightarrow{K},n}\ : \forall R : K_R.\forall E : \text{int}.\forall \overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\forall \beta_1 : \mathcal{A}_{tyarg}(\text{ptr}).\cdots$
$\forall \beta_n : \mathcal{A}_{tyarg}(\text{ptr}).^\wedge \langle \tau_g(R,E),\ \overrightarrow{\mathcal{A}(\alpha : K)\,R\,E},\ \overrightarrow{\mathcal{A}(\beta : \text{ptr})\,R\,E} \rangle \to \exists E'.^\wedge \langle \tau_g(R,E'),\ \mathcal{A}(\langle \overrightarrow{\alpha : K} \rangle)\,R\,E',\ \overrightarrow{\mathcal{A}(\beta : \text{ptr})\,R\,E'} \rangle$

Lemma2 :
If $\Delta \vdash \tau_0 : K_0$ and $\Delta, \alpha \mapsto K_0 \vdash \tau : K$, then:

- $\mathcal{A}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}(\tau[\tau_0/\alpha])$
- $\mathcal{A}_{prim}(\tau : \text{primitive})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}_{prim}((\tau[\tau_0/\alpha]) : \text{primitive})$
- $\mathcal{A}_{ptr}(\tau : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}_{ptr}((\tau[\tau_0/\alpha]) : \text{ptr})$
- $\mathcal{A}_0(\tau : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}_0((\tau[\tau_0/\alpha]) : \text{ptr})$
- $\mathcal{A}_{tyarg}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}_{tyarg}((\tau[\tau_0/\alpha]))$

where $\mathcal{A}(\tau[\tau_0/\alpha]), \mathcal{A}_{prim}(\tau[\tau_0/\alpha]), \mathcal{A}_{ptr}(\tau[\tau_0/\alpha]), \mathcal{A}_0(\tau[\tau_0/\alpha])$ are defined based on the judgment $\Delta \vdash \tau[\tau_0/\alpha] : K$.

Proof by induction on τ. For each τ, the $\mathcal{A}(\tau)$ case relies on the $\mathcal{A}_{ptr}(\tau)$ and $\mathcal{A}_{prim}(\tau)$ cases, and the $\mathcal{A}_{ptr}(\tau)$ case relies on the $\mathcal{A}_0(\tau)$ case. The $\mathcal{A}_{tyarg}(\tau)$ case follows directly from the $\mathcal{A}_{prim}(\tau)$ and $\mathcal{A}_0(\tau)$ cases.

1. $\mathcal{A}(\tau : \text{primitive})[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$
$= (\lambda R : K_R.\lambda E : \text{int}.\mathcal{A}_{prim}(\tau))[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$
$= \lambda R : K_R.\lambda E : \text{int}.(\mathcal{A}_{prim}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha])$
$= \lambda R : K_R.\lambda E : \text{int}.\mathcal{A}_{prim}((\tau[\tau_0/\alpha]) : \text{primitive})$ by the $\mathcal{A}_{prim}(\tau)$ case
$= \mathcal{A}(\tau[\tau_0/\alpha])$

2. $\mathcal{A}(\tau : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$
$= (\lambda R : K_R.\lambda E : \text{int}.\exists L : \text{int}; L \geq 0.(GcPtr(\mathcal{A}_{ptr}(\tau)\,L\,R)\,E))[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$
$= \lambda R : K_R.\lambda E : \text{int}.\exists L : \text{int}; L \geq 0.(GcPtr((\mathcal{A}_{ptr}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha])\,L\,R)\,E)$
$= \lambda R : K_R.\lambda E : \text{int}.\exists L : \text{int}; L \geq 0.(GcPtr(\mathcal{A}_{ptr}((\tau[\tau_0/\alpha]) : \text{ptr})\,L\,R)\,E)$ by the $\mathcal{A}_{ptr}(\tau)$ case
$= \mathcal{A}(\tau[\tau_0/\alpha])$

3. $\mathcal{A}_{ptr}(\tau : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha']$
$= (\mu\alpha : \mathcal{A}_{tyarg}(\text{ptr}).\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\ \cdot \langle \mathcal{A}_0(\tau)\,L\,R\,E\,I,\ GcCoerce\,L\,R\,(\alpha\,L\,R) \rangle)[\mathcal{A}_{tyarg}(\tau_0)/\alpha']$
$= \mu\alpha : \mathcal{A}_{tyarg}(\text{ptr}).\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\ \cdot \langle (\mathcal{A}_0(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha'])\,L\,R\,E\,I,\ GcCoerce\,L\,R\,(\alpha\,L\,R) \rangle$

$= \mu\alpha : \mathcal{A}_{tyarg}(\text{ptr}).\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}. \cdot \langle \mathcal{A}_0(\tau[\tau_0/\alpha])\, L\, R\, E\, I,\ GcCoerce\, L\, R\, (\alpha\, L\, R)\rangle$ by the $\mathcal{A}_0(\tau)$ case
$= \mathcal{A}_{ptr}(\tau[\tau_0/\alpha])$

4,5,6: We show several cases for $\mathcal{A}_0(\tau)$ and $\mathcal{A}_{prim}(\tau)$. The proofs for the other cases are similar.

4. $\mathcal{A}_0(\alpha : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \alpha[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}_{tyarg}(\tau_0)$.
Both α and τ_0 have the same kind $K_0 = \text{ptr}$, so $\mathcal{A}_{tyarg}(\tau_0 : \text{ptr}) = \mathcal{A}_0(\tau_0)$.
Finally, $\mathcal{A}_0(\alpha[\tau_0/\alpha]) = \mathcal{A}_0(\tau_0)$.

5. $\mathcal{A}_{prim}(\alpha : \text{primitive})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \alpha[\mathcal{A}_{tyarg}(\tau_0)/\alpha] = \mathcal{A}_{tyarg}(\tau_0)$.
Both α and τ_0 have the same kind $K_0 = \text{primitive}$, so $\mathcal{A}_{tyarg}(\tau_0 : \text{primitive}) = \mathcal{A}_{prim}(\tau_0)$.
Finally, $\mathcal{A}_{prim}(\alpha[\tau_0/\alpha]) = \mathcal{A}_{prim}(\tau_0)$.

6. $\mathcal{A}_0(\exists\beta : K_\beta.\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$
$= (\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\exists\beta : \mathcal{A}_{tyarg}(K_\beta);\ L \geq 1.(\mathcal{A}_{ptr}(\tau)\,(L-1)\,R\,E\,I))[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$
$= \lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\exists\beta : \mathcal{A}_{tyarg}(K_\beta);\ L \geq 1.((\mathcal{A}_{ptr}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha])\,(L-1)\,R\,E\,I)$
$= \lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\exists\beta : \mathcal{A}_{tyarg}(K_\beta);\ L \geq 1.(\mathcal{A}_{ptr}(\tau[\tau_0/\alpha])\,(L-1)\,R\,E\,I)$ by induction
$= \mathcal{A}_0(\exists\beta : K_\beta.\tau[\tau_0/\alpha])$

Correctness :

1. If $\Delta \vdash_F \tau : K$,
then $\emptyset; \mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}(\tau) : K_R \rightarrow \text{int} \rightarrow \dot{1}$
(corollary: $\emptyset; \mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int} \vdash_L \mathcal{A}(\tau)\, R\, E : \dot{1}$)
and $\emptyset; \mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{prim}(\tau : \text{primitive}) : \dot{1}$
and $\emptyset; \mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{ptr}(\tau : \text{ptr}) : \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow \dot{0}$
and $\emptyset; \mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_0(\tau : \text{ptr}) : \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow \dot{0}$
and $\emptyset; \mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{tyarg}(\tau) : \mathcal{A}_{tyarg}(K)$
2. If $\Delta \vdash_F \Gamma$ and $\Delta; \Gamma \vdash_F t : \tau$
then $\emptyset; \emptyset; \mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)\, R\, E,\ \{g : \tau_g(R, E)\}; true; \infty \vdash_L \mathcal{A}(t)\, R\, E\, g : \mathcal{A}(\tau)\, R\, E$,
3. If $\Delta \vdash_F \Gamma$ and $\Delta; \Gamma \vdash_F e$, then $\emptyset; \emptyset; \mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma) R E,\ \{g : \tau_g(R, E)\}; true; \infty \vdash_L \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$

Proof :

Proof by induction on the typing and kinding derivations of t, e, and τ. For each τ, the $\mathcal{A}(\tau)$ case relies on the $\mathcal{A}_{ptr}(\tau)$ and $\mathcal{A}_{prim}(\tau)$ cases, and the $\mathcal{A}_{ptr}(\tau)$ case relies on the $\mathcal{A}_0(\tau)$ case. In order to write the proof concisely, we delete $\Psi = \emptyset, \Phi = \emptyset,\ B = true,\ Limit = \infty$ from environment in the proof.

1. $\Delta \vdash_F \tau : K$,
$\mathcal{A}(\text{primitive}) = \dot{1}$
$\mathcal{A}(\text{ptr}) = \dot{1}$

case 1 :$\mathcal{A}(\tau : \text{ptr}) = \lambda R : K_R.\lambda E : \text{int}.\exists L : \text{int}; L \geq 0.(GcPtr(\mathcal{A}_{ptr}(\tau)\, L\, R)\, E)$
By the $\mathcal{A}_{ptr}(\tau)$ case, $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{ptr}(\tau) : \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow \dot{0}$.
So $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int},\ L : \text{int} \vdash_L GcPtr(\mathcal{A}_{ptr}(\tau)\, L\, R)\, E : \dot{1}$.
By (K-SOME), $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}(\tau : \text{ptr}) : K_R \rightarrow \text{int} \rightarrow \dot{1}$.

$case\,2: \mathcal{A}(\tau : \text{primitive}) = \lambda R : \text{int}.\lambda E : \text{int}.\mathcal{A}_{prim}(\tau)$

By the $\mathcal{A}_{prim}(\tau)$ case, $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{prim}(\tau) : \dot{1}$.

So $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}(\tau : \text{ptr}) : K_R \to \text{int} \to \dot{1}$.

$case\,3: \mathcal{A}_{ptr}(\tau : \text{ptr}) = \mu\alpha : \mathcal{A}_{tyarg}(\text{ptr}).\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}. \cdot\langle \mathcal{A}_0(\tau)\, L\, R\, E\, I,\ GcCoerce\, L\, R\, (\alpha\, L\, R)\rangle$

By induction, $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_0(\tau : \text{ptr}) : \text{int} \to K_R \to \text{int} \to \text{int} \to \dot{0}$.

By the kinding rules, $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{ptr}(\tau) : \text{int} \to K_R \to \text{int} \to \text{int} \to \dot{0}$.

cases 4,5,6: We show several cases for $\mathcal{A}_0(\tau)$ and $\mathcal{A}_{prim}(\tau)$. The proofs for the other cases are similar.

$case\,4:$
$\mathcal{A}_{prim}((\exists\alpha : K.\tau) : \text{primitive}) = \exists\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{prim}(\tau)$

By induction, $\mathcal{A}_{tyarg}(\Delta, \alpha : K) \vdash_L \mathcal{A}_{prim}(\tau : \text{primitive}) : \dot{1}$.

So $\mathcal{A}_{tyarg}(\Delta), \alpha : \mathcal{A}_{tyarg}(K) \vdash_L \mathcal{A}_{prim}(\tau) : \dot{1}$.

From (K-SOME) rule, $\mathcal{A}_{tyarg}(\Delta) \vdash_L \mathcal{A}_{prim}(\exists\alpha : K.\tau) : \dot{1}$

$case\,5:$
$\mathcal{A}_{prim}(\alpha : \text{primitive}) = \alpha$
From (KF-TVAR) rule, $\Delta', \alpha : \text{primitive} \vdash_F \alpha : \text{primitive}$
By (K-TVAR) rule, $\mathcal{A}_{tyarg}(\Delta'), \alpha : \mathcal{A}_{tyarg}(\text{primitive}) \vdash_L \alpha : \mathcal{A}_{tyarg}(\text{primitive})$
Thus, $\mathcal{A}_{tyarg}(\Delta', \alpha : \text{primitive}) \vdash_L \alpha : \dot{1}$

$case\,6:$
$\mathcal{A}_0(\alpha : \text{ptr}) = \alpha$
From (KF-TVAR) rule, $\Delta', \alpha : \text{ptr} \vdash_F \alpha : \text{ptr}$
By (K-TVAR) rule, $\mathcal{A}_{tyarg}(\Delta'), \alpha : \mathcal{A}_{tyarg}(\text{ptr}) \vdash_L \alpha : \mathcal{A}_{tyarg}(\text{ptr})$
Thus, $\mathcal{A}_{tyarg}(\Delta', \alpha : \text{ptr}) \vdash_L \alpha : \text{int} \to K_R \to \text{int} \to \text{int} \to \dot{0}$

2. $\mathcal{A}(x)\, R\, E\, g = x$
By (T-VAR), $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\},\ x : \mathcal{A}(\tau)RE \vdash_L x : \mathcal{A}(\tau)RE$
So $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma, x : \tau)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L x : \mathcal{A}(\tau)RE$

3. $\mathcal{A}(i)\, R\, E\, g = \text{pack}(i,\ i)\,\text{as}\,\exists I : \text{int}.Int(I)$
From (TF-INT), $\Delta; \Gamma \vdash_F i : \text{int}$
Becuase by (K-IVAR) $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L i : \text{int}$,
by (T-PACK), $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L \text{pack}(i,\ i)\,\text{as}\,\exists I : \text{int}.Int(I) : \exists I : \text{int}.Int(I)$
Thus, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L \mathcal{A}(i)\, R\, E\, g : \mathcal{A}(\text{int})RE$

4. $\mathcal{A}(x_1\, p\, x_2)\, R\, E\, g = OP_p \cdot \langle x_1,\ x_2\rangle,\ (\Delta'; \Gamma' \vdash_L OP_p : \cdot\langle \tau_{int},\ \tau_{int}\rangle \to \tau_{int})$
From (TF-OP) rule, $\Delta; \Gamma \vdash_F x_1 : \text{int}$, and $\Delta; \Gamma \vdash_F x_2 : \text{int}$
By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L x_1 : \tau_{int}$
$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L x_2 : \tau_{int}$
Because $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L OP_p : \cdot\langle \tau_{int},\ \tau_{int}\rangle \to \tau_{int}$
By (T-APP) rule, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\ E)\} \vdash_L \mathcal{A}(x_1\, p\, x_2)\, R\, E\, g : \tau_{int}$

5. $\mathcal{A}(x[\tau_0])\, R\, E\, g = x[\mathcal{A}_{tyarg}(\tau_0)]$
From (TF-TAPP) rule, we know $\Delta \vdash_F \tau_0 : K$, $\Delta; \Gamma \vdash_F x : \forall[\alpha : K, \overrightarrow{\beta : K'}].(\overrightarrow{\tau}) \to \text{void}$
By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int} \vdash_L \mathcal{A}_{tyarg}(\tau_0) : \mathcal{A}_{tyarg}(K)$

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$x : \forall\alpha : \mathcal{A}_{tyarg}(K),\, \overrightarrow{\beta : \mathcal{A}_{tyarg}(K')}.\forall R : K_R.\forall E : \text{int}. \wedge \langle\tau_g(R, E),\, \overrightarrow{\mathcal{A}(\tau)}\rangle \rightarrow \tau_{halt}$

By (T-TAPP) rule, $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$x[\mathcal{A}_{tyarg}(\tau_0)] : \forall\overrightarrow{\beta : \mathcal{A}_{tyarg}(K')}.\forall R : K_R.\forall E : \text{int}.(\wedge\langle\tau_g(R, E),\, \overrightarrow{\mathcal{A}(\tau)}\rangle \rightarrow \tau_{halt})[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$

Using Lemma2, $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$x[\mathcal{A}_{tyarg}(\tau_0)] : \forall\overrightarrow{\beta : \mathcal{A}_{tyarg}(K')}.\forall R : K_R.\forall E : \text{int}. \wedge \langle\tau_g(R, E),\, \overrightarrow{\mathcal{A}(\tau[\tau_0/\alpha])}\rangle \rightarrow \tau_{halt}$

Thus, $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$\mathcal{A}(x[\tau_0])\, R\, E\, g : \mathcal{A}(\forall[\overrightarrow{\beta : K'}].(\overrightarrow{\tau}[\tau_0/\alpha]) \rightarrow \text{void})RE$

6.

case 1. $\mathcal{A}(\text{pack}[\tau,\, x]\,\text{as}\,\exists\beta : K.\tau' : \text{primitive})\, R\, E\, g = \text{pack}\,\mathcal{A}_{tyarg}(\tau),\, x\,\text{as}\,\exists\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}(\tau')\, R\, E$

From (TF-PACK) rule, $\quad \Delta \vdash_F \tau : K, \quad \Delta; \Gamma \vdash_F x : \tau'[\tau/\beta]$

and $\quad \Delta; \Gamma \vdash_F \text{pack}[\tau,\, x]\,\text{as}\,\exists\beta : K.\tau' :\ \exists\beta : K.\tau'$

By induction, $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int} \vdash_L \mathcal{A}_{tyarg}(\tau) : \mathcal{A}_{tyarg}(K)$

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L x : \mathcal{A}(\tau'[\tau/\beta])\, R\, E$

using lemma2, we have $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L x : \mathcal{A}(\tau')[\mathcal{A}_{tyarg}(\tau)/\beta]\, R\, E$

By (T-PACK) rule, $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$\text{pack}\,\mathcal{A}_{tyarg}(\tau),\, x\,\text{as}\,\exists\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}(\tau')\, R\, E : \exists\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}(\tau')\, R\, E$

case 2. $\mathcal{A}(\text{pack}[\tau,\, x]\,\text{as}\,\exists\beta : K.\tau' : \text{ptr})\, R\, E\, g = ptr\_pack_K\, R\, E\, (\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau'))\mathcal{A}_{tyarg}(\tau)x$

We already know, $\Delta; \Gamma \vdash_F \text{pack}[\tau,\, x]\,\text{as}\,\exists\beta : K.\tau' : \text{ptr} : \exists\beta : K.\tau'$

$\Delta; \Gamma \vdash_F x : \tau'[\tau/\beta]$

By induction, $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int} \vdash_L \mathcal{A}_{tyarg}(\tau) : \mathcal{A}_{tyarg}(K)$

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L x : \mathcal{A}(\tau'[\tau/\beta])\, R\, E$

Lemma2 tells us that $(\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau'))\mathcal{A}_{tyarg}(\tau) \equiv \mathcal{A}_{tyarg}(\tau')[\mathcal{A}_{tyarg}(\tau)/\beta] = \mathcal{A}_{tyarg}(\tau'[\tau/\beta]) = \mathcal{A}_0(\tau'[\tau/\beta])$ and $(\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau'))\mathcal{A}_{tyarg}(\gamma) \equiv \mathcal{A}_{tyarg}(\tau')[\mathcal{A}_{tyarg}(\gamma)/\beta] = \mathcal{A}_{tyarg}(\tau'[\gamma/\beta]) = \mathcal{A}_0(\tau'[\gamma/\beta])$. Using this with the type of ptr_pack_K tells us that $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$ptr\_pack_K\, R\, E\, (\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau'))\mathcal{A}_{tyarg}(\tau) : \mathcal{A}(\tau'[\tau/\beta])\, R\, E \rightarrow \mathcal{A}(\exists\gamma : K.(\tau'[\gamma/\beta]))\, R\, E.$

From this we conclude $\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R, E)\} \vdash_L$

$ptr\_pack_K\, R\, E\, (\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau'))\mathcal{A}_{tyarg}(\tau)x : \mathcal{A}(\exists\gamma : K.(\tau'[\gamma/\beta]))\, R\, E.$

And $\exists\gamma : K.(\tau'[\gamma/\beta]) = \exists\beta : K.\tau'$.

7. $\mathcal{A}(fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e)\, R\, E\, g = fix\, x : \forall\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\forall R' : K_R.\forall E' : \text{int}. \wedge \langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \rightarrow \tau_{halt}(R')$.

$\Lambda\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\Lambda R' : K_R.\Lambda E' : \text{int}.\lambda y : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \rightarrow \text{let}\,\langle g',\, \overrightarrow{x}\rangle = y\,\text{in}\,\mathcal{A}(e)\, R'\, E'\, g'$

From (TF-FIX) rule, $\overrightarrow{\alpha : K} \vdash_F \tau_i : K_i',\, \overrightarrow{\alpha : K};\, x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void},\, x_1 : \tau_1,\, \cdots,\, x_n : \tau_n \vdash_F e$

and $\quad \Delta; \Gamma \vdash_F\ fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e \quad : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void}$

By induction, $\mathcal{A}_{tyarg}(\overrightarrow{\alpha : K}),\, R' : K_R,\, E' : \text{int} \vdash_L \mathcal{A}(\tau_i)R'E' : 1$

$\mathcal{A}_{tyarg}(\overrightarrow{\alpha : K}),\, R' : K_R,\, E' : \text{int};\, x : \mathcal{A}(\forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void})R'E',\, x_1 : \mathcal{A}(\tau_1)R'E',\, \cdots,\, x_n : \mathcal{A}(\tau_n)R'E',$

$\{g : \tau_g(R',\, E')\} \vdash_L \mathcal{A}(e)\, R'\, E'\, g : \tau_{halt}$

by (T-VAR), $y : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \vdash_L y : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle$

By (T-LET) rule, $\mathcal{A}_{tyarg}(\overrightarrow{\alpha : K}),\, R' : K_R,\, E' : \text{int};\, x : \mathcal{A}(\forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void})R'E',\, y : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \vdash_L$

$\text{let}\,\langle g',\, \overrightarrow{x}\rangle = y\,\text{in}\,\mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$

By (T-ABS) rule, we obtain $\mathcal{A}_{tyarg}(\overrightarrow{\alpha : K}),\, R' : K_R,\, E' : \text{int};\, x : \mathcal{A}(\forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void})R'E' \vdash_L$

$\lambda y : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \rightarrow \text{let}\,\langle g',\, \overrightarrow{x}\rangle = y\,\text{in}\,\mathcal{A}(e)\, R'\, E'\, g' : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \rightarrow \tau_{halt}$

Because $\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)} \vdash_L \overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)},\, E' : \text{int} \vdash_L E' : \text{int}$, and $R' : K_R \vdash_L R' : K_R$,

by (T-TABS) rule, $\emptyset;\, x : \mathcal{A}(\forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void})R'E' \vdash_L$

$\Lambda\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\Lambda R' : K_R.\Lambda E' : \text{int}.\lambda y : \wedge\langle\tau_g(R', E'),\, \overrightarrow{\mathcal{A}(\tau)\, R'\, E'}\rangle \rightarrow \text{let}\,\langle g',\, \overrightarrow{x}\rangle = y\,\text{in}\,\mathcal{A}(e)\, R'\, E'\, g' :$

$\forall\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}.\forall R' : K_R.\forall E' : \text{int}. \wedge \langle \tau_g(R', E'), \overrightarrow{\mathcal{A}(\tau)\, R'\, E'} \rangle \to \tau_{halt}$

By (T-FIX) rule and weakening lemma, $\mathcal{A}_{tyarg}(\Delta),\ R' : K_R,\ E' : \text{int}; \mathcal{A}(\Gamma)R'E',\ \{g : \tau_g(R',\, E')\} \vdash_L$

$\mathcal{A}(fixcode\, x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e)\, R\, E\, g : \mathcal{A}(\forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \to \text{void})R'E'$

8. $\mathcal{A}(x(\overrightarrow{x}))\, R\, E\, g = x\, R\, E\, \wedge \langle g, \overrightarrow{x} \rangle$

From (TF-APP) rule, $\Delta; \Gamma \vdash_F x_i : \tau_i$, $\Delta; \Gamma \vdash_F x : (\tau_1, \cdots, \tau_n) \to \text{void}$,

and $\Delta; \Gamma \vdash_F x(\overrightarrow{x})$

By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L x_i : \mathcal{A}(\tau_i)RE$

$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L x : \forall R : K_R.\forall E : \text{int}. \wedge \langle \tau_g(R, E), \overrightarrow{\mathcal{A}(\tau)\, R\, E} \rangle \to \tau_{halt}$

By (T-APP) rule, we get $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L x\, R\, E\, \wedge \langle g, \overrightarrow{x} \rangle : \tau_{halt}$

9. $\mathcal{A}(\text{if0}(x,\, e_1,\, e_2))\, R\, E\, g = \text{unpack}\, \beta, x_1 =\ x\, \text{in}\, \text{if}\, x_1 = 0\, \text{then}\, \mathcal{A}(e_1)\, R\, E\, g\ \text{ else}\, \mathcal{A}(e_2)\, R\, E\, g$

From (TF-IF) rule, $\Delta; \Gamma \vdash_F x : \text{int}$, $\Delta; \Gamma \vdash_F e_1\ \ \Delta; \Gamma \vdash_F e_2$,

and $\Delta; \Gamma \vdash_F \text{if0}(x,\, e_1,\, e_2)$

By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g\}RE \vdash_L x : \tau_{int}$

$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L \mathcal{A}(e_1)\, R\, E\, g : \tau_{halt}$

$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L \mathcal{A}(e_2)\, R\, E\, g : \tau_{halt}$

Because $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int},\ \beta : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\},\ x_1 : Int(\beta) \vdash_L (x_1 = 0) : Bool(\beta = 0)$

From (T-IFE) rule , we know

$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int},\ \beta : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\},\ x_1 : Int(\beta) \vdash_L \text{if}\, x_1 = 0\, \text{then}\, \mathcal{A}(e_1)\, R\, E\, g\ \text{ else}\, \mathcal{A}(e_2)\, R\, E\, g : \tau_{halt}$

By (T-UNPACK) rule, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L \mathcal{A}(\text{if0}(x,\, e_1,\, e_2))\, R\, E\, g : \tau_{halt}$

10. $\mathcal{A}(\text{halt}[\tau]x)\, R\, E\, g = halt\, R\, E\, (\mathcal{A}(\tau)\, R\, E) \wedge < g,\ x >$,

From (TF-HALT) rule, we know $\Delta; \Gamma \vdash_F x : \tau$, and $\Delta; \Gamma \vdash_F \text{halt}[\tau]x$

By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L x : \mathcal{A}(\tau)\, R\, E$

Because $\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L halt : \forall R : K_R.\forall E : \text{int}.\forall \alpha :\dot{1}\ . \wedge < \tau_g(R, E),\ \alpha > \to \tau_{halt}$

By (T-APP) rule, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L halt\, R\, E\, (\mathcal{A}(\tau)\, R\, E) \wedge < g,\ x >: \tau_{halt}$

11. $\mathcal{A}(\text{let}\, x = t\, \text{in}\, e)\, R\, E\, g = \text{let}\, x = \mathcal{A}(t)\, R\, E\, g\, \text{in}\, \mathcal{A}(e)\, R\, E\, g$

From (TF-SUB) rule, $\Delta; \Gamma \vdash_F t : \tau$, $\Delta; \Gamma,\ x : \tau \vdash_F e$, and $\Delta; \Gamma \vdash_F \text{let}\, x = t\, \text{in}\, e$

By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L \mathcal{A}(t) : \mathcal{A}(\tau)\, R\, E$

$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\},\ x : \mathcal{A}(\tau)\, R\, E\ \vdash_L \mathcal{A}(e) : \tau_{halt}$

By (T-LET), we get $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L \mathcal{A}(\text{let}\, x = t\, \text{in}\, e)\, R\, E\, g : \tau_{halt}$

12. $\mathcal{A}(\text{let}\, x_0 = \pi_i(x_1 : \langle \overrightarrow{\tau : K} \rangle)\, \text{in}\, e)\, R\, E\, g = \text{let}\, \langle g', x_0 \rangle = obj\_load_{\overrightarrow{K},i}\, R\, E\, \overrightarrow{\mathcal{A}_{tyarg}(\tau)}\ \wedge \langle g, x_1 \rangle\, \text{in}\, \mathcal{A}(e)\, R\, E\, g'$

From (TF-PROJECT), $\Delta; \Gamma \vdash_F x_1 : \langle \tau_1, \cdots, \tau_n \rangle$, $\Delta; \Gamma,\ x_0 : \tau_i \vdash_F e$,

and $\Delta; \Gamma \vdash_F \text{let}\, x_0 = \pi_i x_1\, \text{in}\, e$

By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L x_1 : \mathcal{A}(\langle \overrightarrow{\tau : K} \rangle)\, R\, E$

$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g' : \tau_g(R,\, E)\},\ x_0 : \mathcal{A}(\tau_i)\, R\, E\ \vdash_L \mathcal{A}(e)\, R\, E\, g' : \tau_{halt}$

Because $\emptyset; \emptyset \vdash_L obj\_load_{\overrightarrow{K},i}\ : \forall R : K_R.\forall E : \text{int}.\forall\overrightarrow{\alpha : \mathcal{A}_{tyarg}(K)}. \wedge \langle \tau_g(R, E),\ \mathcal{A}(\langle \overrightarrow{\alpha : K} \rangle)\, R\, E \rangle \to \wedge \langle \tau_g(R, E),\ \mathcal{A}(\alpha_i : K_i)\, R\, E \rangle$

By (T-APP) rule and lemma2, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L$

$obj\_load_{\overrightarrow{K},i}\, R\, E\, \overrightarrow{\mathcal{A}_{tyarg}(\tau)}\ \wedge \langle g, x_1 \rangle : \wedge \langle \tau_g(R, E),\ \mathcal{A}(\tau_i : K_i)\, R\, E \rangle$

By (T-LET) rule, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R,\, E)\} \vdash_L$

$\mathcal{A}(\text{let } x_0 = \pi_i(x_1 : \langle\overrightarrow{\tau : K}\rangle) \text{ in } e)\, R\, E\, g : \tau_{halt}$

13.
case 1. $\mathcal{A}(\text{let } [\alpha : K, x_0 : \tau_0 : \text{primitive}] = \text{unpack } x_1 : \exists\alpha : K.\tau \text{ in } e)\, R\, E\, g = \text{unpack } \alpha,\ x_0 = x_1 \text{ in } \mathcal{A}(e)\, R\, E\, g$
From (TF-UNPACK) we know $\Delta; \Gamma \vdash_F x_1 : \exists\alpha : K.\tau,\ (\Delta, \alpha : K); (\Gamma, x_0 : \tau) \vdash_F e,$
and $\Delta; \Gamma \vdash_F \text{let } [\alpha, x_0] = \text{unpack } x_1 \text{ in } e$
By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L x_1 : \mathcal{A}(\exists\alpha : K.\tau)\, R\, E$
$\mathcal{A}_{tyarg}(\Delta, \alpha : K),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma, x_0 : \tau)RE,\ \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$
Because $\Delta; \Gamma \vdash_F \tau : \text{primitive}$,
$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L x_1 : \exists\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{prim}(\tau)$
$\mathcal{A}_{tyarg}(\Delta),\ \alpha : \mathcal{A}_{tyarg}(K),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\},\ x_0 : \mathcal{A}_{prim}(\tau)\, R\, E \vdash_L \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$
By (T-UNPACK) rule,
$\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L \text{unpack } \alpha,\ x_0 = x_1 \text{ in } \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$

case 2. $\mathcal{A}(\text{let } [\alpha : K, x_0 : \tau_0 : \text{ptr}] = \text{unpack } x_1 : \exists\alpha : K.\tau \text{ in } e)\, R\, E\, g =$
$\text{unpack } \alpha, x_0 = ptr\_unpack_K\, R\, E\, (\lambda\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau))\, x_1 \text{ in } \mathcal{A}(e)\, R\, E\, g$
From (TF-UNPACK) we know $\Delta; \Gamma \vdash_F x_1 : \exists\alpha : K.\tau,\ (\Delta, \alpha : K); (\Gamma, x_0 : \tau) \vdash_F e,$
and $\Delta; \Gamma \vdash_F \text{let } [\alpha, x_0] = \text{unpack } x_1 \text{ in } e$
By induction, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L x_1 : \mathcal{A}(\exists\alpha : K.\tau)\, R\, E$
$\mathcal{A}_{tyarg}(\Delta),\ \mathcal{A}_{tyarg}(\alpha) : \mathcal{A}_{tyarg}(K),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\},\ x_0 : \mathcal{A}(\tau)\, R\, E \vdash_L \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$
From definition we know $\emptyset; \emptyset \vdash_L$
$ptr\_unpack_K : \forall R : K_R.\forall E : \text{int}.\forall\alpha : \mathcal{A}_{tyarg}(K) \to \mathcal{A}_{tyarg}(\text{ptr}).\mathcal{A}(\exists\beta : K.(\alpha\,\beta))\, R\, E \to \exists\beta : \mathcal{A}_{tyarg}(K).(\mathcal{A}(\alpha\,\beta)\, R\, E)$
Lemma2 tells us that $(\lambda\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau))\mathcal{A}_{tyarg}(\alpha) \equiv \mathcal{A}_{tyarg}(\tau)[\alpha/\alpha] = \mathcal{A}_{tyarg}(\tau) = \mathcal{A}_0(\tau)$. Using this with the type of ptr_unpack_K tells us that $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L$
$ptr\_unpack_K\, R\, E\, (\lambda\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau)) : \mathcal{A}(\exists\alpha : K.\tau)\, R\, E \to \exists\alpha : \mathcal{A}_{tyarg}(K).(\mathcal{A}(\tau)\, R\, E).$
Thus, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L$
$ptr\_unpack_K\, R\, E\, (\lambda\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau))\, x_1 : \exists\alpha : \mathcal{A}_{tyarg}(K).(\mathcal{A}(\tau)\, R\, E)$
Because $\mathcal{A}_{tyarg}(\Delta),\ \mathcal{A}_{tyarg}(\alpha) : \mathcal{A}_{tyarg}(K),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\},\ x_0 : \mathcal{A}(\tau)\, R\, E \vdash_L \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$
By (T-UNPACK) rule, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L$
$\text{unpack } \alpha, x_0 = ptr\_unpack_K\, R\, E\, (\lambda\alpha : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau))\, x_1 \text{ in } \mathcal{A}(e)\, R\, E\, g : \tau_{halt}$

14. $\mathcal{A}(\text{let } x_0 = \langle\overrightarrow{x : \tau_x}\rangle \text{ in } e)\, R\, E\, g = \text{unpack } E',\ z = alloc_{\overrightarrow{K},n}\, R\, E\, \overrightarrow{\mathcal{A}_{tyarg}(\tau_x)}\, \overrightarrow{\mathcal{A}_{tyarg}(\tau_y)} \wedge \langle g, \overrightarrow{x}, \overrightarrow{y}\rangle \text{ in}$
$\text{let } \langle g', x_0, \overrightarrow{y}\rangle = z \text{ in } \mathcal{A}(e)\, R\, E'\, g';$ where $\{y_1, \cdots, y_n\} = \{y : \tau_y : \text{ptr} \in FV(e)\} - \{x_0\}$
From (TF-VEC) rule, we know $\Delta; \Gamma \vdash_F x_i : \tau_i,\ \Delta; \Gamma,\ x_0 : \langle\tau_1, \cdots, \tau_n\rangle \vdash_F e,$
$\Delta; \Gamma \vdash_F \text{let } x_0 = \langle\overrightarrow{x}\rangle \text{ in } e$
By induction, we get $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L x_i : \mathcal{A}(\tau_i)\, R\, E$,
and $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E' : \text{int}; \mathcal{A}(\Gamma)RE',\ \{g' : \tau_g(R, E')\},\ x_0 : \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E' \vdash_L \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$
From definition, $\emptyset; \emptyset \vdash_L alloc_{\overrightarrow{K},n} : \forall R : K_R.\forall E : \text{int}.\overrightarrow{\forall\alpha : \mathcal{A}_{tyarg}(K)}.\forall\beta_1 : \mathcal{A}_{tyarg}(\text{ptr}).\cdots$
$\forall\beta_n : \mathcal{A}_{tyarg}(\text{ptr}).\wedge\langle\tau_g(R, E),\ \overrightarrow{\mathcal{A}(\alpha : K)\, R\, E},\ \overrightarrow{\mathcal{A}(\beta : \text{ptr})\, R\, E}\rangle \to \exists E'.\wedge\langle\tau_g(R, E'),\ \mathcal{A}(\langle\overrightarrow{\alpha : K}\rangle)\, R\, E',\ \overrightarrow{\mathcal{A}(\beta : \text{ptr})\, R\, E'}\rangle$
Using lemma2, $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E : \text{int}; \mathcal{A}(\Gamma)RE,\ \{g : \tau_g(R, E)\} \vdash_L alloc_{\overrightarrow{K},n} R\, E\, \overrightarrow{\mathcal{A}_{tyarg}(\tau_x)}\, \overrightarrow{\mathcal{A}_{tyarg}(\tau_y)} \wedge \langle g, \overrightarrow{x}, \overrightarrow{y}\rangle$:
$\exists E'. \wedge \langle\tau_g(R, E'),\ \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E',\ \overrightarrow{\mathcal{A}(\tau_y)\, R\, E'}\rangle$
We know $\mathcal{A}_{tyarg}(\Delta),\ R : K_R,\ E' : \text{int}; \mathcal{A}(\Gamma)RE',\ \{g' : \tau_g(R, E')\},\ x_0 : \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E' \vdash_L \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$.
Say that $\Gamma = \Gamma_{ptr} \cup \Gamma_{ptrextra} \cup \Gamma_{prim}$ where $\Gamma_{ptr}, \Gamma_{ptrextra}, \Gamma_{prim}$ have disjoint domains and:
$\Gamma_{ptr} = \{y_1 \mapsto \tau_{y_1} : \text{ptr}, \ldots, y_n \mapsto \tau_{y_n} : \text{ptr}\}$
$\Gamma_{ptrextra} = \{y'_1 \mapsto \tau_{y'_1} : \text{ptr}, \ldots, y'_{n'} \mapsto \tau_{y'_{n'}} : \text{ptr}\}$
$\Gamma_{prim} = \{z \mapsto \tau_{z_1} : \text{prim}, \ldots, z_m \mapsto \tau_{z_m} : \text{prim}\}$

All free pointer variables in e are in Γ_{ptr}, so no free variables of e appear in $\Gamma_{ptrextra}$. It's easy to show that e typechecks without $\Gamma_{ptrextra}$ (by induction on the typing derivation), and that each free pointer variable y_i in e must appear in Γ (also by induction on the typing derivation):

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E' : \text{int}; \mathcal{A}(\Gamma_{ptr} \cup \Gamma_{prim})RE',\, \{g' : \tau_g(R,\, E')\},\, x_0 : \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E' \vdash_L \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$.

Then by (T-LET) and (T-VAR):

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E' : \text{int}; \mathcal{A}(\Gamma_{prim})RE',\, z : \wedge\langle\tau_g(R, E'),\, \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E',\, \overrightarrow{\mathcal{A}(\tau_y)\, R\, E'}\rangle \vdash_L$
$\text{let}\, \langle g', x_0, \overrightarrow{y}\rangle = z\, \text{in}\, \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$

For primitive types, $\mathcal{A}(\tau : \text{primitive}) = \lambda R : K_R.\lambda E : \text{int}.\mathcal{A}_{prim}(\tau)$, so $\mathcal{A}(\tau : \text{primitive})\, R\, E' = \mathcal{A}(\tau : \text{primitive})\, R\, E$. This means that $\mathcal{A}(\Gamma_{prim})\, R\, E' = \mathcal{A}(\Gamma_{prim})\, R\, E$, which we can use after weaking the environment with E:

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int},\, E' : \text{int}; \mathcal{A}(\Gamma_{prim})RE,\, z : \wedge\langle\tau_g(R, E'),\, \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E',\, \overrightarrow{\mathcal{A}(\tau_y)\, R\, E'}\rangle \vdash_L$
$\text{let}\, \langle g', x_0, \overrightarrow{y}\rangle = z\, \text{in}\, \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$

We can weaken Γ_{prim} to the superset Γ:

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int},\, E' : \text{int}; \mathcal{A}(\Gamma)RE,\, z : \wedge\langle\tau_g(R, E'),\, \mathcal{A}(\langle\overrightarrow{\tau_x}\rangle)\, R\, E',\, \overrightarrow{\mathcal{A}(\tau_y)\, R\, E'}\rangle \vdash_L$
$\text{let}\, \langle g', x_0, \overrightarrow{y}\rangle = z\, \text{in}\, \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$

By (T-UNPACK), we get

$\mathcal{A}_{tyarg}(\Delta),\, R : K_R,\, E : \text{int}; \mathcal{A}(\Gamma)RE,\, \{g : \tau_g(R,\, E)\} \vdash_L$
$\text{unpack}\, E',\, z = alloc_{\overrightarrow{K},n} R\, E\, \overrightarrow{\mathcal{A}_{tyarg}(\tau_x)}\, \overrightarrow{\mathcal{A}_{tyarg}(\tau_y)} \wedge \langle g,\, \overrightarrow{x},\, \overrightarrow{y}\rangle\, \text{in}\, \text{let}\, \langle g', x_0, \overrightarrow{y}\rangle = z\, \text{in}\, \mathcal{A}(e)\, R\, E'\, g' : \tau_{halt}$

References

[1] Chris Hawblitzel, Edward Wei, Heng Huang, Lea Wittie, and Eric Krupski. Low-level linear memory management (submitted for publication). 2003.

[2] Heng Huang and Chris Hawblitzel. Proofs of soundness and strong normalization for linear memory types. In *Dartmouth Technical Report TR2002-437*, 2002.

[3] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From system f to typed assembly language. volume 21, pages 527–568. ACM Press, 1999.

[4] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[5] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.