

Dartmouth College

## Dartmouth Digital Commons

---

Computer Science Technical Reports

Computer Science

---

8-8-2003

### Formal Properties of Linear Memory Types

Heng Huang

*Dartmouth College*

Lea Wittie

*Dartmouth College*

Chris Hawblitzel

*Dartmouth College*

Follow this and additional works at: [https://digitalcommons.dartmouth.edu/cs\\_tr](https://digitalcommons.dartmouth.edu/cs_tr)



Part of the [Computer Sciences Commons](#)

---

#### Dartmouth Digital Commons Citation

Huang, Heng; Wittie, Lea; and Hawblitzel, Chris, "Formal Properties of Linear Memory Types" (2003).  
Computer Science Technical Report TR2003-468. [https://digitalcommons.dartmouth.edu/cs\\_tr/218](https://digitalcommons.dartmouth.edu/cs_tr/218)

This Technical Report is brought to you for free and open access by the Computer Science at Dartmouth Digital Commons. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Dartmouth Digital Commons. For more information, please contact [dartmouthdigitalcommons@groups.dartmouth.edu](mailto:dartmouthdigitalcommons@groups.dartmouth.edu).

# 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  $\lambda^{low}$ , 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  $\lambda^C$ , a CPS- and closure-converted polymorphic lambda calculus [3], to  $\lambda^{low}$ . This report supersedes an earlier report[2].

Formally stated, the theorems proved in this report are:

- **Type preservation:** If  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau$ ,  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$  and  $(M, e) \rightarrow (M', e')$ , then  $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$  and  $\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$ , where  $(\Phi'_e \supseteq \Phi_e)$ .
- **Type progress:** If  $(M, e)$  is closed and well-typed ( $C_{Me} \vdash (M, e : \tau)$  for some  $\tau$  and  $C_{Me} = \Psi_{Me}; \Phi_{Me}; \emptyset; \emptyset; B; \text{limit}$ ), then either  $e$  is a value or else there is some  $(M', e')$  so that  $(M, e) \rightarrow (M', e')$ .
- **Strong normalization:** If  $\Psi_{Me}; \Phi_{Me}; \emptyset; \emptyset; \text{true}; i \vdash (M, e : \tau)$ , then  $e$  must step to a value in a finite sequence of zero or more steps, without changing memory:  $(M, e) \rightarrow (M, e_1) \rightarrow \dots \rightarrow (M, e_n) \rightarrow (M, v)$ .

- **Type erasure:**

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

- **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  $\emptyset; \emptyset; \mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) RE, \{g : \tau_g(R, E)\}; \text{true}; \infty \vdash_L \mathcal{A}(e) RE g : \tau_{halt}$

## 1 Abstract syntax

This section defines the abstract syntax of  $\lambda^{low}$ . The letter  $x$  is used to indicate a value variable, while  $\alpha$  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

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

### kinds

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

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

### arithmetic

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

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

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

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

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

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

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

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

### types

$$\tau = \tau_1 \xrightarrow{\phi, \text{limit}} \tau_2 \mid \phi(\vec{\tau}) \mid \alpha \mid I \mid B \mid \lambda \alpha : K. \tau \mid \forall \alpha : K; B. \tau$$

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

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

## expressions

$$\begin{aligned} e = & i \mid b \mid x \mid e_1 e_2 \mid e\tau \mid \phi(\vec{e}) \mid \lambda x : \tau \xrightarrow{\phi, \text{limit}} e \mid e_1 \text{ op } e_2 \mid \neg e \\ & \mid \Lambda \alpha : K ; B.v \mid \text{let } \langle \vec{x} \rangle = e_1 \text{ in } e_2 \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \mid \text{if } B \text{ then } e_1 \text{ else } e_2 \\ & \mid \text{union}(b, \tau_1, \tau_2, e) \mid \text{pack}[\tau_1, e] \text{ as } \exists \alpha : K ; B.\tau_2 \mid \text{unpack } \alpha, x = e_1 \text{ in } e_2 \\ & \mid \text{case}(b, e) \mid \text{roll}[(\mu \alpha : K.\tau_0)\tau_1 \cdots \tau_n](e) \mid \text{unroll}(e) \mid \text{fix } x : \tau.v \mid \text{load}(e_{\text{ptr}}, e_{\text{Has}}) \\ & \mid \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v) \mid \text{coerce}(e) \mid \text{distinguish}(I_1, I_2, e_1, e_2) \mid \text{make\_eq}(\tau) \\ & \mid \text{apply\_eq}(\tau, e_1, e_2) \mid \text{new\_fun}(K) \mid \text{discard\_fun}(e) \\ & \mid \text{define\_fun}(e, \tau) \mid \text{in\_domain}(I_1, I_2, e_1, e_2) \mid \text{fact} \end{aligned}$$

## values

$$\begin{aligned} v = & i \mid b \mid \Lambda \alpha : K ; B.v \mid \text{pack}[\tau_1, v] \text{ as } \exists \alpha : K ; B.\tau_2 \\ & \mid \text{roll}[(\mu \alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) \mid \lambda x : \tau \xrightarrow{\phi, \text{limit}} e \mid \phi(\vec{v}) \mid \text{union}(b, \tau_1, \tau_2, v) \mid \text{fact} \end{aligned}$$

## untyped expressions

$$\begin{aligned} d = & i \mid b \mid x \mid d_1 d_2 \mid \langle \vec{d} \rangle \mid \lambda x \longrightarrow d \\ & \mid d_1 \text{ op } d_2 \mid \neg d \mid \text{let } \langle \vec{x} \rangle = d_1 \text{ in } d_2 \\ & \mid \text{if } d_1 \text{ then } d_2 \text{ else } d_3 \mid \text{fix } x.u \mid \text{load}(d_{\text{ptr}}, \langle \rangle) \mid \text{store}(d_{\text{ptr}}, \langle \rangle, d_v) \end{aligned}$$

## untyped values

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

## expressions for substitution

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

## environments

$$\begin{aligned} F &= F_1^{K_1} \mid F_2^{K_2} \mid F_3^{K_3} \mid \dots \\ M &= \{1 \mapsto v_1, \dots, n \mapsto v_n\} \\ \Psi &= \{1 \mapsto \tau_1, \dots, n \mapsto \tau_n\} \\ \Phi &= \{F_1^{K_1} \xrightarrow{\phi_1} \delta_1, \dots, F_n^{K_n} \xrightarrow{\phi_n} \delta_n\} \\ \delta &= \{0 \mapsto \tau_0, \dots, n \mapsto \tau_n\} \end{aligned}$$

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

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

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

### untyped environments

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

### judgments

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

$$B \vdash B_1 \doteq B_2$$

$$B \vdash I_1 \doteq I_2$$

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

$$C \vdash e : \tau$$

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

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

### abbreviations

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

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

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

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

$$\text{let } x = e_1 \text{ in } e_2 \triangleq \text{let } \langle x \rangle = \wedge \langle e_1 \rangle \text{ in } e_2$$

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

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

## 1.1 Notes on environments

We treat the environments  $\Psi, \Phi, \delta, \Delta, \Gamma$  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  $\Psi, \Phi, \delta, \Delta, \Gamma$  must be well-formed functions (and the definitions in this report apply only to well-formed functions):

$$\begin{aligned} (i \mapsto \tau_1 \in \Psi) \wedge (i \mapsto \tau_2 \in \Psi) &\Rightarrow \tau_1 = \tau_2 \\ (F^K \xrightarrow{\phi_1} \delta_1 \in \Phi) \wedge (F^K \xrightarrow{\phi_2} \delta_2 \in \Phi) &\Rightarrow (\delta_1 = \delta_2) \wedge (\phi_1 = \phi_2) \\ (i \mapsto \tau_1 \in \delta) \wedge (i \mapsto \tau_2 \in \delta) &\Rightarrow \tau_1 = \tau_2 \\ (\alpha \mapsto K_1 \in \Delta) \wedge (\alpha \mapsto K_2 \in \Delta) &\Rightarrow K_1 = K_2 \\ (x \mapsto \tau_1 \in \Gamma) \wedge (x \mapsto \tau_2 \in \Gamma) &\Rightarrow \tau_1 = \tau_2 \end{aligned}$$

For  $\Psi, \delta, \Delta, \Gamma$  we use the usual function application notation:  $\Gamma(x) = \tau \Leftrightarrow x \mapsto \tau \in \Gamma$ .

Each  $F_i^{K_i}$  in  $\Phi$  must suffice to type-check both  $\text{Gen}(F_i^{K_i}, I)$  expressions, which are linear, and other expression containing the type operator  $F_i^{K_i}$ , which may be nonlinear. The  $\text{Gen}(F_i^{K_i}, I)$  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:

$$\Psi = \Psi_1, \Psi_2 \Leftrightarrow (\Psi = \Psi_1 \cup \Psi_2) \wedge (\text{domain}(\Psi_1) \not\cap \text{domain}(\Psi_2))$$

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

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

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

### Environment extension

These add new elements to environments:

$$\begin{aligned} \Psi, i \mapsto \tau &\triangleq \Psi \cup \{i \mapsto \tau\}, \text{ where } i \notin \text{domain}(\Psi) \\ \Phi, F^K \xrightarrow{\phi} \delta &\triangleq \Phi \cup \{F^K \xrightarrow{\phi} \delta\}, \text{ where } F^K \notin \text{domain}(\Phi) \\ \Delta, \alpha \mapsto K &\triangleq \Delta \cup \{\alpha \mapsto K\}, \text{ where } \alpha \notin \text{domain}(\Delta) \\ \Gamma, x \mapsto \tau &\triangleq \Gamma \cup \{x \mapsto \tau\}, \text{ where } x \notin \text{domain}(\Gamma) \\ B_1, B_2 &\triangleq B_1 \wedge B_2 \\ (\Psi; \Phi; \Delta; \Gamma; B; \text{limit}), i \mapsto \tau &\triangleq (\Psi, i \mapsto \tau; \Phi; \Delta; \Gamma; B; \text{limit}) \\ (\Psi; \Phi; \Delta; \Gamma; B; \text{limit}), F^K \xrightarrow{\phi} \delta &\triangleq (\Psi; \Phi, F^K \xrightarrow{\phi} \delta; \Delta; \Gamma; B; \text{limit}) \\ (\Psi; \Phi; \Delta; \Gamma; B; \text{limit}), \alpha \mapsto K_\alpha &\triangleq (\Psi; \Phi; \Delta, \alpha \mapsto K_\alpha; \Gamma; B; \text{limit}), \\ \text{where } \alpha &\text{ does not appear anywhere in } (\Psi; \Phi; \Delta; \Gamma; B; \text{limit}). \\ (\Psi; \Phi; \Delta; \Gamma; B; \text{limit}), x \mapsto \tau &\triangleq (\Psi; \Phi; \Delta; \Gamma, x \mapsto \tau; B; \text{limit}), \end{aligned}$$

where  $x$  does not appear anywhere in  $(\Psi; \Phi; \Delta; \Gamma; B; \text{limit})$  and  $\Phi; \Delta \vdash \tau : \overset{\phi}{i}$ .  
 $(\Psi; \Phi; \Delta; \Gamma; B_1; \text{limit}), B_2 \triangleq (\Psi; \Phi; \Delta; \Gamma; B_1, B_2; \text{limit})$

## Nonlinear environments

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

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

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

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

## Environment subsets

As the abstract machine steps from one state to the next, the  $\Phi$  environment must grow to accomodate new type sequence allocations. Therefore, we define a subset operator  $\dot{\subseteq}$  for  $\Phi$  to indicate that the  $\Phi_2$  after a step is an extension of the  $\Phi_1$  before the step. To accomodate the  $\text{discard\_fun}(e)$  expression, this operator must be able to demote a linear mapping  $F^K \overset{\wedge}{\mapsto} \delta$  to a nonlinear mapping  $F^K \mapsto \delta$ :

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

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

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,  $\hat{\subseteq}$ :

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

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

where we define subset for  $\Gamma$  as:

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

where  $\Phi; \Delta \vdash \tau_1 : i_1, \dots, \Phi; \Delta \vdash \tau_n : i_n$ .

## 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.

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

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

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

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

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

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

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

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

$$\frac{\forall i. (\Phi; B \vdash \tau_i \equiv \tau'_i)}{\Phi; B \vdash T[\tau_1, \dots, \tau_n] \equiv T[\tau'_1, \dots, \tau'_n]}$$

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  $T[\tau_1, \dots, \tau_n]$  to indicate the type with the holes replaced by  $\tau_1, \dots, \tau_n$ , so that a single type equivalence rule covers all the cases. This is only for notational convenience.

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

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

$$\mid \text{Has}(\tau_1, \tau_2) \mid \text{Gen}(\tau_1, \tau_2) \mid \text{Eq}(\tau_1, \tau_2) \mid \text{InDomain}(\tau_1, \tau_2)$$

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

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

## 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.
- $\text{simplify}(i_1 + i_2) = i_3$ , where  $i_3$  is the sum of  $i_1$  and  $i_2$
- $\text{simplify}(i_1 - i_2) = i_3$ , where  $i_3$  is  $i_1$  minus  $i_2$



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

## 3.2 Rules

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$(E - MAKEEQ)make\_eq(\tau) \rightarrow fact$$

$$(E - APPLYEQ)apply\_eq(\tau, fact, v) \rightarrow v$$

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

$$(E - DISCARDFUN)discard\_fun(fact) \rightarrow \cdot \langle \rangle$$

$$(E - DEFINEFUN)define\_fun(fact, \tau) \rightarrow^\wedge \langle fact, fact, fact \rangle$$

Rather than writing each of the congruence rules separately ( $\frac{e_2 \rightarrow e'_2}{v_1 e_2 \rightarrow v_1 e'_2}$ , 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.

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

$$\mid \text{coerce}(e) \mid ee_2 \mid v_1 e \mid \phi(v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n) \mid \text{let } \vec{x} = e \text{ in } e_2$$

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

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

$$\mid \text{distinguish}(I_1, I_2, e, e_2) \mid \text{distinguish}(I_1, I_2, v_1, e) \mid \text{apply\_eq}(\tau, e, e_2) \mid \text{apply\_eq}(\tau, v_1, e)$$

$$\mid \text{discard\_fun}(e) \mid \text{define\_fun}(e, \tau) \mid \text{in\_domain}(I_1, I_2, e, e_2) \mid \text{in\_domain}(I_1, I_2, v_1, e)$$

$$\begin{aligned} & (\text{congruence rule}) \frac{(M, e) \rightarrow (M', e')}{(M, E[e]) \rightarrow (M', E[e'])} \\ & \frac{e \rightarrow e'}{(M, e) \rightarrow (M, e')} \end{aligned}$$

## 4 Type well-formedness

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

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

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

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

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

$$\begin{array}{c}
(K - BOP) \frac{\Phi; \Delta \vdash B_1 : \text{bool} \quad \Phi; \Delta \vdash B_2 : \text{bool}}{\Phi; \Delta \vdash B_1 \text{ bop } B_2 : \text{bool}} \\
(K - ANTI) \frac{\Phi; \Delta \vdash B : \text{bool}}{\Phi; \Delta \vdash \neg B : \text{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 : \text{bool} \quad \Phi; \Delta, \alpha : K \vdash \tau : i^\phi}{\Phi; \Delta \vdash \forall \alpha : K; B. \tau : i^\phi} \\
(K - SOME) \frac{\Phi; \Delta, \alpha : K \vdash B : \text{bool} \quad \Phi; \Delta, \alpha : K \vdash \tau : i^\phi}{\Phi; \Delta \vdash \exists \alpha : K; B. \tau : i^\phi} \\
(K - REC) \frac{\Phi; \Delta, \alpha : K \vdash \tau : K}{\Phi; \Delta \vdash \mu \alpha : K. \tau : K} \\
(K - ABS) \frac{\Phi; \Delta \vdash \tau_1 : i_1^{\phi_1} \quad \Phi; \Delta \vdash \tau_2 : i_2^{\phi_2} \quad (\Phi; \Delta \vdash \text{limit} : \text{int} \Rightarrow i = 0) \text{ or } (\text{limit} = \infty \Rightarrow i = 1)}{\Phi; \Delta \vdash \tau_1 \xrightarrow{\phi, \text{limit}} \tau_2 : i^\phi} \\
(K - NLTUPLE) \frac{\forall j. (\Phi; \Delta \vdash \tau_j : i_j^\cdot)}{\Phi; \Delta \vdash \langle \vec{\tau} \rangle : i^\cdot} (i = \sum_j i_j) \\
(K - LTUPLE) \frac{\forall j. (\Phi; \Delta \vdash \tau_j : i_j^{\phi_j})}{\Phi; \Delta \vdash \langle \vec{\tau} \rangle : i^\wedge} (i = \sum_j i_j) \\
(K - INT) \frac{\Phi; \Delta \vdash I : \text{int}}{\Phi; \Delta \vdash \text{Int}(I) : \dot{1}} \\
(K - BOOL) \frac{\Phi; \Delta \vdash B : \text{bool}}{\Phi; \Delta \vdash \text{Bool}(B) : \dot{1}} \\
(K - UNION) \frac{\Phi; \Delta \vdash B : \text{bool} \quad \Phi; \Delta \vdash \tau_1 : i^\phi \quad \Phi; \Delta \vdash \tau_2 : i^\phi}{\Phi; \Delta \vdash \text{Union}(B, \tau_1, \tau_2) : i^\phi} \\
(K - HAS) \frac{\Phi; \Delta \vdash I : \text{int} \quad \Phi; \Delta \vdash \tau : \dot{1}}{\Phi; \Delta \vdash \text{Has}(I, \tau) : \hat{0}} \\
(K - FUNGEN) \frac{\Phi; \Delta \vdash I : \text{int} \quad \Phi; \Delta \vdash \tau : \text{int} \rightarrow J}{\Phi; \Delta \vdash \text{Gen}(\tau, I) : \hat{0}} \\
(K - INDOMAIN) \frac{\Phi; \Delta \vdash I : \text{int} \quad \Phi; \Delta \vdash \tau : \text{int} \rightarrow J}{\Phi; \Delta \vdash \text{InDomain}(I, \tau) : \dot{0}} \\
(K - EQ) \frac{\Phi; \Delta \vdash \tau_1 : K \quad \Phi; \Delta \vdash \tau_2 : K}{\Phi; \Delta \vdash \text{Eq}(\tau_1, \tau_2) : \dot{0}} \\
(K - FUN) \Phi, \mathbb{F}^K \xrightarrow{\phi} \delta; \Delta \vdash \mathbb{F}^K : \text{int} \rightarrow K
\end{array}$$

## 4.1 Type checking rules

$$\begin{array}{c}
\begin{array}{c}
\Psi = \Psi_{\text{spare}}, \Psi_e \quad \Phi = \dot{\Phi}_{\text{spare}}, \Phi_e \\
\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M(i) : \Psi(i)) \\
\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau \\
(T - MEM) \frac{}{\Psi; \Phi; \Delta; \Gamma; B; \text{limit} \vdash (M, e) : \tau}
\end{array} \\
\\
(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} \\
\\
\begin{array}{c}
C \vdash e : \forall \alpha : K; B.\tau_1 \\
C \vdash \tau_2 : K \\
(T - TAPP) \frac{C \vdash [\alpha \mapsto \tau_2]B}{C \vdash e\tau_2 : [\alpha \mapsto \tau_2]\tau_1}
\end{array} \\
\\
(T - COERCE) \frac{\Psi; \Phi; \Delta; \Gamma; B; I_2 \vdash e : \tau \quad (\text{limit}_1 = \infty \wedge \tau : \dot{0}) \text{ or } (\text{limit}_1 = I_1 \wedge B \vdash I_1 > I_2)}{\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e) : \tau} \\
\\
(T - EQ) \frac{C \vdash e : \tau_1 \quad C \vdash \tau_1 \equiv \tau_2 \quad C \vdash \tau_1 : K_1 \quad C \vdash \tau_2 : K_1}{C \vdash e : \tau_2} \\
\\
(T - TUPLE) \frac{C = \dot{C}, C_1, \dots, C_n \quad \forall i. (C_i \vdash e_i : \tau_i) \quad C \vdash \phi(\vec{\tau}) : K}{C \vdash \phi(\vec{e}) : \phi(\vec{\tau})} \\
\\
(T - ABS) \frac{\dot{\Phi}; \Delta \vdash \tau_1 : K \quad \dot{\Psi}; \dot{\Phi}; \Delta; \Gamma, x : \tau_1; B; \text{limit}_2 \vdash e : \tau_2 \quad (\text{limit}_2 = \infty) \text{ or } (\dot{\Phi}; \Delta \vdash \text{limit}_2 : \text{int} \quad B \vdash \text{limit}_2 \geq 0)}{\dot{\Psi}; \dot{\Phi}; \Delta; \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{C_1, C_2 = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}_C \quad C_1 \vdash e_1 : \tau_a \overset{\phi, \text{limit}_f}{\longrightarrow} \tau_b \quad C_2 \vdash e_2 : \tau_a \quad (\text{limit}_C = \text{limit}_f = \infty) \text{ or } (B \vdash \text{limit}_f < \text{limit}_C) \quad \text{or } (\text{limit}_C = \infty, \text{limit}_f = I, \dot{\Phi}; \Delta \vdash \tau_b : \dot{0})}{C_1, C_2 \vdash e_1 e_2 : \tau_b} \\
\\
(T - LET) \frac{C_a \vdash e_a : \langle \vec{\tau} \rangle \quad C_b, \bar{x} : \vec{\tau} \vdash e_b : \tau_b}{C_a, C_b \vdash \text{let} \langle \vec{x} \rangle = e_a \text{ in } e_b : \tau_b} \\
\\
(T - FIX) \frac{C \vdash \tau : \dot{i} \quad C, x : \tau \vdash v : \tau}{C \vdash (\text{fix } x : \tau.v) : \tau} \\
\\
(T - PACK) \frac{C \vdash \tau_1 : K \quad C \vdash \exists \alpha : K; B.\tau_2 : \dot{i} \quad C \vdash e : [\alpha \mapsto \tau_1]\tau_2 \quad C \vdash [\alpha \mapsto \tau_1]B}{C \vdash \text{pack}[\tau_1, e] \text{ as } \exists \alpha : K; B.\tau_2 : \exists \alpha : K; B.\tau_2}
\end{array}$$

$$\begin{array}{c}
(T - UNPACK) \frac{C_1 \vdash e_1 : \exists \alpha : K; B.\tau_1 \quad C_2, \alpha : K, x : \tau_1, B \vdash e_2 : \tau_2}{C_1, C_2 \vdash \text{unpack } \alpha, x = e_1 \text{ in } e_2 : \tau_2} \\
(T - ROLL) \frac{\tau = (\mu \alpha : K.\tau_0)\tau_1 \cdots \tau_n \quad C \vdash \tau : \overset{\phi}{i}}{C \vdash e : ([\alpha \mapsto \mu \alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n}{C \vdash \text{roll}[\tau](e) : \tau} \\
(T - UNROLL) \frac{\tau = (\mu \alpha : K.\tau_0)\tau_1 \cdots \tau_n \quad C \vdash \tau : \overset{\phi}{i} \quad C \vdash e : \tau}{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 \text{fact} : \text{Has}(i, \tau) \\
(T - FACT2) \dot{C}, F^K \hat{\mapsto} \delta \vdash \text{fact} : \text{Gen}(F^K, i) \\
(T - FACT3) \dot{C} \vdash \text{fact} : \text{Eq}(\tau, \tau) \\
(T - FACT4) \frac{i \in \text{dom}(\delta)}{\dot{C}, F^K \mapsto \delta \vdash \text{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{C = C_1, C_2, C_3 = \Psi; \Phi; \Delta; \Gamma; B; \infty \quad C_2 \vdash e_{\text{Has}} : \text{Has}(I, \tau_1) \quad C_3 \vdash e_v : \tau_2 \quad C_1 \vdash e_{\text{ptr}} : \text{Int}(I) \quad C \vdash \tau_2 : \dot{1}}{C \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v) : \text{Has}(I, \tau_2)} \\
(T - DISTINGUISH) \frac{C_1, C_2 \vdash I_1 : \text{int} \quad C_1, C_2 \vdash I_2 : \text{int} \quad C_1 \vdash e_1 : \text{Has}(I_1, \tau_1) \quad C_2 \vdash e_2 : \text{Has}(I_2, \tau_2)}{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{C \vdash \tau_f : K \rightarrow J \quad C \vdash e_1 : \text{Eq}(\tau_a, \tau_b) \quad C \vdash \tau_a : K \quad C \vdash \tau_b : K \quad C \vdash e_2 : \tau_f \tau_a}{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} \rightarrow J.\text{Gen}(\alpha, 0) \\
(T - DISCARDFUN) \frac{C \vdash e : \text{Gen}(\tau, I)}{C \vdash \text{discard\_fun}(e) : \cdot \langle \cdot \rangle} \\
(T - DEFINEFUN) \frac{C \vdash e : \text{Gen}(\tau_f, I) \quad C \vdash \tau_f : \text{int} \rightarrow 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}
\end{array}$$

$$\begin{array}{c}
\frac{C_1, C_2 \vdash I_1 : \text{int} \quad C_1, C_2 \vdash I_2 : \text{int}}{C_1 \vdash e_1 : \text{InDomain}(I_1, \tau_f) \quad C_2 \vdash e_2 : \text{Gen}(\tau_f, I_2)} \\
(T - \text{INDOMAIN}) \frac{}{C_1, C_2 \vdash \text{in\_domain}(I_1, I_2, e_1, e_2) : \wedge \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2) \rangle} \\
(T - \text{INT}) \dot{C} \vdash i : \text{Int}(i) \\
(T - \text{BOOL}) \dot{C} \vdash b : \text{Bool}(b) \\
(T - \text{IOP}) \frac{C_1 \vdash e_1 : \text{Int}(I_1) \quad C_2 \vdash e_2 : \text{Int}(I_2)}{C_1, C_2 \vdash e_1 \text{ iop } e_2 : \text{Int}(I_1 \text{ iop } I_2)} \\
(T - \text{CMP}) \frac{C_1 \vdash e_1 : \text{Int}(I_1) \quad C_2 \vdash e_2 : \text{Int}(I_2)}{C_1, C_2 \vdash e_1 \text{ cmp } e_2 : \text{Bool}(I_1 \text{ cmp } I_2)} \\
(T - \text{BOP}) \frac{C_1 \vdash e_1 : \text{Bool}(B_1) \quad C_2 \vdash e_2 : \text{Bool}(B_2)}{C_1, C_2 \vdash e_1 \text{ bop } e_2 : \text{Bool}(B_1 \text{ bop } B_2)} \\
(T - \text{CBOOL}) \frac{C \vdash e : \text{Bool}(B)}{C \vdash \neg e : \text{Bool}(\neg B)} \\
(T - \text{IFE}) \frac{C_a \vdash e_1 : \text{Bool}(B) \quad C_b, B \vdash e_2 : \tau \quad C_b, \neg B \vdash e_3 : \tau}{C_a, C_b \vdash \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : \tau} \\
(T - \text{IFB}) \frac{C = \Psi; \Phi; \Delta; \Gamma; B_C; I \quad C \vdash B : \text{bool} \quad C, B \vdash e_1 : \tau \quad C, \neg B \vdash e_2 : \tau}{C \vdash \text{if } B \text{ then } e_1 \text{ else } e_2 : \tau} \\
(T - \text{UNION}) \frac{C \vdash \tau_1 : K \quad C \vdash \tau_2 : K \quad C \vdash e : \tau_i \quad (i = \begin{array}{l} 1 \text{ if } b \\ 2 \text{ if } \neg b \end{array})}{C \vdash \text{union}(b, \tau_1, \tau_2, e) : \text{Union}(b, \tau_1, \tau_2)} \\
(T - \text{CASE}) \frac{C \vdash e : \text{Union}(b, \tau_1, \tau_2)}{C \vdash \text{case}(b, e) : \tau_i \quad (i = \begin{array}{l} 1 \text{ if } b \\ 2 \text{ if } \neg b \end{array})}
\end{array}$$

## 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  $e = \dots e_1 \dots$ , and the derivation of  $C \vdash (M, e : \tau)$  contains the judgment  $C_1 \vdash e_1 : \tau_1$ , then we annotate  $e$  with  $\tau_1$  as  $e = \dots (e_1 : \tau_1) \dots$  when convenient. We annotate types with kinds in a similar fashion. These annotations guide the erasure rules.

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

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

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

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

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

$$(ER - APP) \text{erase}((e_1 : (\tau_1 \xrightarrow{\phi, \infty} \tau_2)) e_2) = \text{erase}(e_1) \text{erase}(e_2)$$

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

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

$$(ER - TUPLE) \text{erase}(\langle e_1, \dots, e_n \rangle : t : i) = \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle \text{ where } i > 0$$

$$(ER - TUPLEv0) \text{erase}(\langle v_1, \dots, v_n \rangle : t : 0) = \langle \rangle$$

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

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

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

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

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

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

$$(ER - LET) \text{erase}(\text{let } \langle \vec{x} \rangle = e_1 \text{ in } e_2) = \text{let } \langle \vec{x} \rangle = \text{erase}(e_1) \text{ in } \text{erase}(e_2)$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 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.

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

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

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

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

## 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  $C_1 \subseteq C'_1$ , then there is some  $C'_2$  such that  $C_2 \subseteq C'_2$  and  $C_1, C_2 \subseteq C'_1, C'_2$ .

If  $C = C_1, C_2$  and  $C_1 \subseteq C'_1$ , then there is some  $C'_2$  such that  $C_2 \subseteq C'_2$  and  $C_1, C_2 \subseteq C'_1, C'_2$ . Proof: if  $C_1 \subseteq C'_1$ , 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  $F^K \xrightarrow{\phi} \delta$ , it can add new elements to  $\delta$ . In this case, simply add the same elements to the same  $\delta$  in  $C_2$  (by the definition of splitting  $C_1, C_2$ , we know  $C_2$  contains the same  $\delta$ ) to get  $C'_2$ .
- It can change a  $F^K \xrightarrow{\hat{}} \delta$  to  $F^K \mapsto \delta$ . In this case,  $C_2$  must already contain  $F^K \mapsto \delta$ , so choose  $C'_2 = C_2$ .
- It can add a new  $F^K \xrightarrow{\phi} \delta$ , which appears neither in  $C_1$  nor  $C_2$ . In this case, add  $F^K \mapsto \delta$  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  $\tau_a$  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 ( $\text{limit}_C = \infty$ ,  $\text{limit}_f = I$ ,  $\Phi; \Delta \vdash \tau' : \hat{0}$ )).
3. (ABS) If  $C \vdash \lambda x : \tau_x \xrightarrow{\phi, \text{limit}_f} e : \tau_1 \xrightarrow{\text{limit}_f} \tau_2$ , then  $\tau_1 \equiv \tau_x$  and  $\Psi; \Phi; \Delta; \Gamma, x : \tau_x; B; \text{limit}_f \vdash e : \tau_2$ . ( $C \vdash \tau_1 :: K_1$ ,  $C \vdash \tau_x :: K_1$ ), ( $\text{limit}_f = \infty$ ) 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  $C \vdash \phi(\vec{e}) : \tau$ , then there is  $\phi(\vec{\tau'}) \equiv \tau$ , and  $C \vdash \phi(\vec{e}) : \phi(\vec{\tau'})$ ,  $\forall i. (C_i \vdash e_i : \tau'_i)$ ,  $C \vdash \phi(\vec{\tau'}) : K$ .
7. (PROJECT) If  $C \vdash \text{let } \langle \vec{x} \rangle = e_a \text{ in } e_b : \tau$ , then there is  $\tau_b \equiv \tau$ , and  $C \vdash \text{let } \langle \vec{x} \rangle = e_a \text{ in } e_b : \tau_b$ , and  $C_{e_a} \vdash e_a : \phi(\vec{\tau'})$ ,  $C_{e_b, \vec{x} : \tau'} \vdash e_b : \tau_b$ .
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)  $C \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v) : \tau$ , then there is  $\text{Has}(I, \tau_2) \equiv \tau$ , and  $C_{e_{\text{ptr}}} \vdash e_{\text{ptr}} : \text{Int}(I)$ ,  $C_{e_{\text{has}}} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$ ,  $C_{e_v} \vdash e_v : \tau_2$ ,  $C \vdash \tau_2 : \hat{1}$ .
10. (FIX)  $C \vdash \text{fix } x : \tau_x. v : \tau$ , then  $\tau \equiv \tau_x$  and  $C, x : \tau_x \vdash v : \tau$ ,  $C \vdash \tau : \hat{i}$
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  $C \vdash \tau : \hat{i}$ ,  $C \vdash e_0 : (\mu \alpha : K. \tau_0) \tau_1 \cdots \tau_n$
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  $C \vdash \tau : \hat{i}$ ,  $C \vdash e : ([\alpha \mapsto \mu \alpha : K. \tau_0] \tau_0) \tau_1 \cdots \tau_n$
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  $\tau \equiv \exists \alpha : \text{int} \rightarrow J. \text{Gen}(\alpha, 0)$
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  $C \vdash e : \text{Gen}(\tau_f, I)$ ,  $C \vdash \tau_f : \text{int} \rightarrow J$ ,  $C \vdash \tau_a : J$
18. (DISCARDFUN)  $C \vdash \text{discard\_fun}(e_0) : \tau'$ , then  $\tau' \equiv \cdot \langle \cdot \rangle$  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  $\tau_b$ ,  $\tau_f \tau_b \equiv \tau$  and  $C \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau_f \tau_b$ ,  $C \vdash \tau_f : K \rightarrow J$ ,  $C \vdash \tau_a : K$ ,  $C \vdash \tau_b : K$ ,  $C \vdash e_1 : \text{Eq}(\tau_a, \tau_b)$ ,  $C \vdash e_2 : \tau_f \tau_a$
22. (CASE)  $C \vdash \text{case}(b, e) : \tau$ , then  $\tau \equiv \tau_i$  ( $i = \begin{smallmatrix} 1 \text{ if } b \\ 2 \text{ if } \neg b \end{smallmatrix}$ ) 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  $C \vdash \tau'_1 : K$ ,  $C \vdash \tau'_2 : K$ ,  $C \vdash e : \tau_i$  ( $i = \begin{smallmatrix} 1 \text{ if } \tau_b \\ 2 \text{ if } \neg \tau_b \end{smallmatrix}$ )
24. (COERCE)  $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e) : \tau$ , then there is  $\tau' \equiv \tau$ , and  $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e) : \tau$ ,  $\Psi; \Phi; \Delta; \Gamma; B; I_2 \vdash e : \tau'$ , ( $(\text{limit}_1 = \infty \wedge \tau : 0)$  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  $\tau_n$ . For  $\tau_n$ , we will use (T-VAR) for  $C \vdash x : \tau_n$ . ( $\tau_1 \equiv \dots \equiv \tau_n \equiv \tau$ )

If using (T-VAR), then  $\tau_n$  is  $\tau$ .

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  $C \vdash e_1 e_2 : \tau$ , then there is some type  $\tau_a$  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  $(\text{limit}_C = \infty, \text{limit}_f = I, \Phi; \Delta \vdash \tau' : 0)$ ).

For  $C \vdash e_1 e_2 : \tau$ , we only have two type checking rules to use. (T-EQ) and (T-APP).

If using (T-EQ), then there will be  $\tau'$ . For  $\tau'$ , we will use (T-APP) for  $C \vdash e_1 e_2 : \tau'$ . ( $\tau \equiv \tau_1 \equiv \dots \equiv \tau_n \equiv \tau'$ )

If using (T-APP), then  $\tau'$  is  $\tau$ .

By (T-APP) rule, we obtain  $C \vdash e_1 e_2 : \tau'$ ,  $C_1 \vdash e_1 : \tau_a \xrightarrow{\text{limit}_f} \tau'$ , and  $C_2 \vdash e_2 : \tau_a$ , ( $(\text{limit}_C = \text{limit}_f = \infty)$  or  $(\text{limit}_f < \text{limit}_C)$  or  $(\text{limit}_C = \infty, \text{limit}_f = I, \Phi; \Delta \vdash \tau' : 0)$ ).

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

## 6.2.2 LEMMA [LIMIT-CHANGE]

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

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

By (T-INT) rule,

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

2.  $s = b$

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

By (T-BOOL) rule,

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

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.  $s = \text{fact}$

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

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

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

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

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

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

*case2* :  $\emptyset; \Phi; F^K \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{fact} : \text{Gen}(F^K, i)$

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

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

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

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

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

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 rules 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 rules 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  $\Phi; \Delta \vdash \tau_1 : K$ ,  $\Phi; \Delta \vdash \exists \alpha : K; B.\tau_2 : i$

$B_1 \vdash [\alpha \mapsto \tau_1]B$ ,  $\Psi; \Phi; \Delta; \Gamma; B_1; \text{limit}_1 \vdash v : [\alpha \mapsto \tau_1]\tau_2$

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  $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{roll}[\tau](v) : \tau$

$\Phi; \Delta \vdash \tau : i$ ,

$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash v : ([\alpha \mapsto \mu \alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$

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 rules 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:

$\Phi; \Delta \vdash \tau_1 : K$ ,  $\Phi; \Delta \vdash \tau_2 : K$ ,  $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash v : \tau_i$  ( $i = \begin{matrix} 1 \text{ if } b \\ 2 \text{ if } \neg b \end{matrix}$ )

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.  $s = \phi(\vec{v})$

$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \phi(\vec{v}) : \tau$

By Lemma (6.2.1.(6)), we have  $\phi(\vec{\tau}') \equiv \tau$ , and  $\forall i. (\Psi_i; \Phi_i; \Delta; \Gamma_i; B; \text{limit}_1 \vdash v_i : \tau'_i)$ ,  $\Phi; \Delta \vdash \phi(\vec{\tau}') : K$

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,

$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash \phi(\vec{v}) : \phi(\vec{\tau}')$

By (T-EQ) rule,  $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash \phi(\vec{v}) : \tau$

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

$\Psi; \Phi; \Delta; \Gamma, x : \tau; B; \text{limit}_1 \vdash v : \tau'$ ,  $\Phi; \Delta \vdash \tau' : i$

By induction,  $\Psi; \Phi; \Delta; \Gamma, x : \tau; B; \text{limit}_2 \vdash v : \tau'$

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

### 6.2.3 LEMMA [ARITHMETIC FORMS]

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

### 6.2.4 LEMMA [TYPE ENVIRONMENT SUBSTITUTION]

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

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

### 6.2.5 LEMMA [TYPE SUBSTITUTION 1]

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

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

Proof by induction on the kinding derivation.

1.  $\tau = \beta$

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

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

2.  $\tau = i$

then by (K-IVAR) rule,  $C, \overline{\alpha : K_\alpha} \vdash i : \text{int} \Rightarrow C \vdash i : \text{int}$   
 $[\overline{\alpha \mapsto \tau_\alpha}] \tau = i$ , thus  $C \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau : \text{int}$

3.  $\tau = \lambda \beta : K_a. \tau_0$  (In this case, we can rename  $\beta$ , then  $\alpha \neq \beta$ )

By (K-TABS) rule,  $C, \overline{\alpha : K_\alpha} \vdash \lambda \beta : K_a. \tau_0 : K_a \rightarrow K_b$ , and  $C, \beta : K_a, \overline{\alpha : K_\alpha} \vdash \tau_0 : K_b$

By induction, we obtain:

$(C, \beta : K_a) \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_0 : K_b$   
 $\Rightarrow C, \beta : K_a \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_0 : K_b$

By (K-TABS) rule,

$C \vdash \lambda \beta : K_a. [\overline{\alpha \mapsto \tau_\alpha}] \tau_0 : K_a \rightarrow K_b$   
 $\Rightarrow C \vdash [\overline{\alpha \mapsto \tau_\alpha}] \lambda \beta : K_a. \tau_0 : K_a \rightarrow K_b$

4.  $\tau = \tau_f \tau_a$

By (K-TAPP) rule,  $C, \overline{\alpha : K_\alpha} \vdash \tau_f \tau_a : K_b$ , and  $C, \overline{\alpha : K_\alpha} \vdash \tau_f : K_a \rightarrow K_b$ ,  $C, \overline{\alpha : K_\alpha} \vdash \tau_a : K_a$

By induction, we have

$C \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_f : K_a \rightarrow K_b$   
 $C \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_a : K_a$

By (K-APP) rule,

$C \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_f \tau_a : K_b$

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

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

6.  $\tau = F^K$

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

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

### 6.2.6 LEMMA [TYPE SUBSTITUTION 2]

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

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,  $[\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]B_1 \Rightarrow [\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]B_2$  for all remaining substitutions 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  $[\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]B$  is syntactically well-formed and  $\Phi; B \vdash \tau_1 \equiv \tau_2$ , and  $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$ , then  $[\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]\Phi; [\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]B \vdash [\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]\tau_1 \equiv [\overrightarrow{\alpha} \mapsto \overrightarrow{\tau_\alpha}]\tau_2$ .

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

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

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

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

### 6.2.8 LEMMA [TERM SUBSTITUTION]

If we make the following definitions and assumptions:

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

- $[\overline{\alpha \mapsto \tau_\alpha}]C = \Psi; \Phi; \Delta; [\overline{\alpha \mapsto \tau_\alpha}]\Gamma; [\overline{\alpha \mapsto \tau_\alpha}]B; [\overline{\alpha \mapsto \tau_\alpha}]\text{limit}_C$  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).
- $C, \overline{\alpha : K_\alpha}, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash e : \tau$
- $z_i \notin \text{domain}(\Gamma)$
- $[\overline{\alpha \mapsto \tau_\alpha}]C \vdash s_{x_i} : [\overline{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$  (where  $\tau_{x_i}$  are nonlinear types)
- $[\overline{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{y_i}) \vdash s_{y_i} : [\overline{\alpha \mapsto \tau_\alpha}]\tau_{y_i}$  (where  $\tau_{y_i}$  are linear types)
- $[\overline{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{z_i}) \vdash s_{z_i} : [\overline{\alpha \mapsto \tau_\alpha}]\tau_{z_i}$  (where  $\tau_{z_i}$  are linear types)
- $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$

then we can conclude the following:

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

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

1a.  $e = x_0$ , where  $x_0 \neq x_i, y_i, z_i$

$C, \overline{\alpha : K_\alpha}, \overline{x : \tau_x} \vdash x_0 : \tau$ . The  $\overline{y}$  must be empty, and  $\dot{C} = C = (C, \overline{C_{y_i}})$ , and  $C(x_0) = \tau$ .

$[s]e = x_0$

Since  $C(x_0) = \tau$ , we know  $([\overline{\alpha \mapsto \tau_\alpha}]C)(x_0) = [\overline{\alpha \mapsto \tau_\alpha}]\tau$ .

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

1b.  $e = x_i$

$\tau = \tau_{x_i}$  and  $C, \overline{\alpha : K_\alpha}, \overline{x : \tau_x} \vdash x_0 : \tau_{x_i}$ . The  $\overline{y}$  must be empty, and  $\dot{C} = C = (C, \overline{C_{y_i}})$ .

$[s]e = s_{x_i}$  and  $[\overline{\alpha \mapsto \tau_\alpha}]C \vdash s_{x_i} : [\overline{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$ .

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

2a.  $e = y_i$

$C, \overline{\alpha : K_\alpha}, \overline{x : \tau_x}, \overline{y_i : \tau_{y_i}} \vdash e : \tau_{y_i}$  and  $\tau = \tau_{y_i}$ . The  $\overline{y}$  must be empty except for  $y_i$ , and  $\dot{C} = C$  and  $\overline{C_{y_i}} = C_{y_i}$ .

$[s]e = s_{y_i}$  and  $[\overline{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{y_i}) \vdash s_{y_i} : [\overline{\alpha \mapsto \tau_\alpha}]\tau_{y_i}$ ,

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

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

3.  $e = e_1 e_2$

$C, \overline{\alpha : K_\alpha}, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash e_1 e_2 : \tau_2$

By Lemma (6.2.1.(2)), we have

$C_1, \overline{\alpha : K_\alpha}, \overline{x : \tau_x}, \overline{y' : \tau_{y'}} \vdash e_1 : \tau_1 \xrightarrow{f} \tau_2 \quad C_2, \overline{\alpha : K_\alpha}, \overline{x : \tau_x}, \overline{y'' : \tau_{y''}} \vdash e_2 : \tau_1$

By induction, we obtain:

$[\overline{\alpha \mapsto \tau_\alpha}](C_1, \overline{C_{y'_i}}) \vdash [\overline{\alpha \mapsto \tau_\alpha}, \overline{x \mapsto s_x}, \overline{y' \mapsto s_{y'}}, (\overline{z \mapsto s_z}, \overline{y'' \mapsto s_{y''}})]e_1 :$

$[\overline{\alpha \mapsto \tau_\alpha}](\tau_1 \xrightarrow{f} \tau_2) \Rightarrow [\overline{\alpha \mapsto \tau_\alpha}](C_1, \overline{C_{y'_i}}) \vdash [s]e_1 : [\overline{\alpha \mapsto \tau_\alpha}]\tau_1 \xrightarrow{f} [\overline{\alpha \mapsto \tau_\alpha}]\tau_2$

$[\overline{\alpha \mapsto \tau_\alpha}](C_2, \overline{C_{y''_i}}) \vdash [\overline{\alpha \mapsto \tau_\alpha}, \overline{x \mapsto s_x}, \overline{y'' \mapsto s_{y''}}, (\overline{z \mapsto s_z}, \overline{y' \mapsto s_{y'}})]e_2 : [\overline{\alpha \mapsto \tau_\alpha}]\tau_1$

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

By (T-APP) rule,

$$\begin{aligned} & [\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash [s]e_1([s]e_2) : [\overline{\alpha \mapsto \tau_\alpha}] \tau_2 \\ & [\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash [\overline{\alpha \mapsto \tau_\alpha}, \overline{x \mapsto s_x}, \overline{y \mapsto s_y}, \overline{z \mapsto s_z}]e : [\overline{\alpha \mapsto \tau_\alpha}] \tau \end{aligned}$$

$$4. e = \lambda w : \tau \xrightarrow{\phi, \text{limit}} e_0$$

By Lemma (6.2.1.(3)) and (T-EQ) rule, we have

$$C, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash \lambda w : \tau_w \xrightarrow{\phi, \text{limit}_2} e_0 : \tau_w \xrightarrow{\phi, \text{limit}_2} \tau_0$$

and  $C', w : \tau_w, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash e_0 : \tau_0$ ,  $\Phi; \Delta \vdash \tau_w : K$ , (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:

$$[\overline{\alpha \mapsto \tau_\alpha}] \overline{C'} \vdash s_{x_i} : [\overline{\alpha \mapsto \tau_\alpha}] \tau_{x_i} \text{ and } [\overline{\alpha \mapsto \tau_\alpha}] (\overline{C'}, \overline{C'_{y_i}}) \vdash s_{y_i} : [\overline{\alpha \mapsto \tau_\alpha}] \tau_{y_i} \text{ and } [\overline{\alpha \mapsto \tau_\alpha}] (\overline{C'}, \overline{C'_{z_i}}) \vdash s_{z_i} : [\overline{\alpha \mapsto \tau_\alpha}] \tau_{z_i}.$$

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  $\text{limit}_2$  in  $\overline{C'}, \overline{C'_{y_i}}, \overline{C'_{z_i}}$ )

If  $\tau_w$  is nonlinear, then by type substitution  $[\overline{\alpha \mapsto \tau_\alpha}] \tau_w$  is also nonlinear and so the Weakening Lemma says:

$$[\overline{\alpha \mapsto \tau_\alpha}] (\overline{C'}, w : \tau_w) \vdash s_{x_i} : [\overline{\alpha \mapsto \tau_\alpha}] \tau_{x_i} \text{ and } [\overline{\alpha \mapsto \tau_\alpha}] (\overline{C'}, \overline{C'_{y_i}}, w : \tau_w) \vdash s_{y_i} : [\overline{\alpha \mapsto \tau_\alpha}] \tau_{y_i} \text{ and } [\overline{\alpha \mapsto \tau_\alpha}] (\overline{C'}, \overline{C'_{z_i}}, w : \tau_w) \vdash s_{z_i} : [\overline{\alpha \mapsto \tau_\alpha}] \tau_{z_i}.$$

If  $\tau_w$  is linear, weakening is not needed.

By induction, we have:

$$[\overline{\alpha \mapsto \tau_\alpha}] (\overline{C'}, \overline{C'_{y_i}}), w : [\overline{\alpha \mapsto \tau_\alpha}] \tau_w \vdash [s]e_0 : [\overline{\alpha \mapsto \tau_\alpha}] \tau_0$$

By Type Substitution Lemma we know,  $[\overline{\alpha \mapsto \tau_\alpha}] (\Phi; \Delta) \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_w : K$

By (T-ABS) rule, we obtain:

$$\begin{aligned} & [\overline{\alpha \mapsto \tau_\alpha}] (C, \overline{C_{y_i}}) \vdash [s]e : [\overline{\alpha \mapsto \tau_\alpha}] \tau_w \xrightarrow{\phi, \text{limit}_2} [\overline{\alpha \mapsto \tau_\alpha}] \tau_0 \\ & [\overline{\alpha \mapsto \tau_\alpha}] (C, \overline{C_{y_i}}) \vdash [s]e : [\overline{\alpha \mapsto \tau_\alpha}] (\tau_w \xrightarrow{\phi, \text{limit}_2} \tau_0) \end{aligned}$$

$$5. e = \text{pack}[\tau_1, e_0] \text{ as } \exists \beta : K; B'. \tau_2$$

By Lemma (6.2.1.(14)) and (T-EQ) rule, we have

$$C, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash \text{pack}[\tau_1, e_0] \text{ as } \exists \beta : K; B'. \tau_2 : \exists \beta : K; B'. \tau_2, \text{ and}$$

$$C, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash \tau_1 : K \Rightarrow C, \alpha : K_\alpha \vdash \tau_1 : K,$$

$$C, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash [\beta \mapsto \tau_1] B' \Rightarrow C, \alpha : K_\alpha \vdash [\beta \mapsto \tau_1] B',$$

$$C, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash e_0 : [\beta \mapsto \tau_1] \tau_2,$$

$$C, \alpha : K_\alpha, \overline{x : \tau_x}, \overline{y : \tau_y} \vdash \exists \beta : K; B'. \tau_2 : i \Rightarrow C, \alpha : K_\alpha \vdash \exists \beta : K; B'. \tau_2 : i$$

By induction, we get:

$$[\overline{\alpha \mapsto \tau_\alpha}] (C, \overline{C_{y_i}}) \vdash [s]e_0 : [\overline{\alpha \mapsto \tau_\alpha}] [\beta \mapsto \tau_1] \tau_2$$

$$[\overline{\alpha \mapsto \tau_\alpha}] (C, \overline{C_{y_i}}) \vdash [s]e_0 : [\beta \mapsto \tau_1] ([\overline{\alpha \mapsto \tau_\alpha}] \tau_2)$$

By Type Substitution Lemma,

$$[\overline{\alpha \mapsto \tau_\alpha}] C \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_1 : K,$$

$$[\overline{\alpha \mapsto \tau_\alpha}] C \vdash [\overline{\alpha \mapsto \tau_\alpha}] (\exists \beta : K; B'. \tau_2) : i$$

$$\Rightarrow [\overline{\alpha \mapsto \tau_\alpha}] C \vdash \exists \beta : K; ([\overline{\alpha \mapsto \tau_\alpha}] B'). [\overline{\alpha \mapsto \tau_\alpha}] \tau_2 : i$$

$$C, \alpha : K_\alpha \vdash \exists \beta : K; B'. \tau_2 : i \Rightarrow C, \alpha : K_\alpha, \beta : K \vdash B' : \text{bool} \Rightarrow [\beta \mapsto \tau_1] C, \alpha : K_\alpha \vdash [\beta \mapsto \tau_1] B' : \text{bool} \Rightarrow$$

$$[\overline{\alpha \mapsto \tau_\alpha}] [\beta \mapsto \tau_1] C \vdash [\overline{\alpha \mapsto \tau_\alpha}] [\beta \mapsto \tau_1] B' : \text{bool} \text{ (this implies that } [\overline{\alpha \mapsto \tau_\alpha}] [\beta \mapsto \tau_1] B' \text{ is syntactically well-formed)}$$

$$\Rightarrow [\overline{\alpha \mapsto \tau_\alpha}] C \vdash [\beta \mapsto [\overline{\alpha \mapsto \tau_\alpha}] \tau_1] ([\overline{\alpha \mapsto \tau_\alpha}] B'),$$

By Weakening Lemma,

$$[\overline{\alpha \mapsto \tau_\alpha}] (C, \overline{C_{y_i}}) \vdash [\overline{\alpha \mapsto \tau_\alpha}] \tau_1 : K,$$



$[\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash [\beta \mapsto [\overline{\alpha \mapsto \tau_\alpha}]\tau_1]([\overline{\alpha \mapsto \tau_\alpha}]B')$ ,  
 $[\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash \exists\beta : K; ([\overline{\alpha \mapsto \tau_\alpha}]B').[\overline{\alpha \mapsto \tau_\alpha}]\tau_2 : i$   
 By (T-PACK) rule,  
 $[\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash \text{pack}[[\overline{\alpha \mapsto \tau_\alpha}]\tau_1, [s]e_0] \text{ as } \exists\beta : K; [\overline{\alpha \mapsto \tau_\alpha}]B'.[\overline{\alpha \mapsto \tau_\alpha}]\tau_2 :$   
 $\exists\beta : K; [\overline{\alpha \mapsto \tau_\alpha}]B'.[\overline{\alpha \mapsto \tau_\alpha}]\tau_2$   
 Therefore,  $[\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash$   
 $[s](\text{pack}[\tau_1, e_0] \text{ as } \exists\beta : K; B'.\tau_2) : [\overline{\alpha \mapsto \tau_\alpha}]\exists\beta : K; B'.\tau_2$

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

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

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

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

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

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

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

By induction, we obtain:

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

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

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

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

By (T-UNPACK) rule,  $[\overline{\alpha \mapsto \tau_\alpha}](C, \overline{C_{y_i}}) \vdash$

$\text{unpack } \beta, w = [s]e_1 \text{ in } [s]e_2 : [\overline{\alpha \mapsto \tau_\alpha}]\tau_2$

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

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

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

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

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

By induction,

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

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

By Type Substitution Lemma,

$[\overline{\alpha \mapsto \tau_\alpha}]C \vdash [\overline{\alpha \mapsto \tau_\alpha}]\tau : i$

By Weakening Lemma,

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

By (T-FIX) rule,

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

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

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

### 6.2.9 THEOREM [PRESERVATION]

If  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau$ ,  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$  and  $(M, e) \rightarrow (M', e')$ , then  $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$  and  $\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$ , 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*:  $e = \lambda x : \tau_1 \xrightarrow{\phi, \text{limit}} e_0$

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*:  $e = e_1 e_2$      $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash e : \tau$

From lemma 6.2.1.(2), there is  $\tau_1$  and

$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit}_C \vdash e_1 : \tau_1 \xrightarrow{\text{limit}_f} \tau'$ ,

( $\text{limit}_C = \text{limit}_f = \infty$ ) or ( $\text{limit}_f < \text{limit}_C$ ) or ( $\text{limit}_C = \infty, \text{limit}_f = I, \Phi_e; \Delta \vdash \tau' : \dot{0}$ )

$\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit}_C \vdash e_2 : \tau_1, \tau' \equiv \tau$

(*E - APPABS1*)

$e_1 = \lambda x : \tau_x \xrightarrow{\phi, I} e_{12}$      $e_2 = v_2$      $e' = \text{coerce}([x \mapsto v_2]e_{12})$

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 ( $\text{limit}_C = \infty, \Phi_e; \Delta \vdash \tau' : \dot{0}$ ),

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  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

Thus,  $\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M', e' : \tau)$

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

(*E - APPABS2*)

$e_1 = \lambda x : \tau_x \xrightarrow{\phi, \infty} e_{12}$      $e_2 = v_2$      $e' = [x \mapsto v_2]e_{12}$

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  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

Thus,  $\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit}_C \vdash (M', e' : \tau)$

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

5. *Case T – TAPP*:  $e = e_1\tau_2$       $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_1\tau_2 : \tau$

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*)

$e_1 = \Lambda \alpha : K'; B.v$ ,      $e' = [\alpha \mapsto \tau_2]v$ ,

Because  $\tau'_2 \equiv \tau_2$ , then  $e' = [\alpha \mapsto \tau'_2]v$

By Lemma 6.2.1.(5),  $K' = K$ ,      $\Psi_e; \Phi_e; \Delta, \alpha : K'; \Gamma; B; \text{limit} \vdash v : \tau'_1$

Thus,  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau'_2 : K'$ ;      $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_1 : \forall \alpha : K'; B.\tau'_1$

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  $\alpha$  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  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

6. *Case T – TUPLE*:  $e = \phi(\vec{e})$       $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \phi(\vec{e}) : \tau$

By Lemma 6.2.1.(6),  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \phi(\vec{e}) : \phi(\vec{\tau}')$ , and  $\phi(\vec{\tau}') \equiv \tau$ ,

and  $\forall i. (\Psi_{e_i}; \Phi_{e_i}; \Delta; \Gamma_{e_i}; B; \text{limit} \vdash e_i : \tau_i)$       $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \phi(\vec{\tau}') : K$

Suppose,  $e_k \rightarrow e'_k$  now.

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \phi(\vec{e}) : \phi(\vec{\tau}'))$ ,

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

and  $\Psi_{e_k}; \Phi_{e_k}; \Delta; \Gamma_{e_k}; B; \text{limit} \vdash e_k : \tau'_k$

Thus,  $\Psi'_{\text{spare}}, \Psi_{e_k}; \Phi'_{\text{spare}}, \Phi_{e_k}; \Delta; \Gamma_{e_k}; B; \text{limit} \vdash (M, e_k : \tau'_k)$

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

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

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

and  $\Psi'_{\text{spare}} = \Psi_{\text{spare}}, \Psi_{e_1}, \dots, \Psi_{e_{k-1}}, \Psi_{e_{k+1}}, \dots, \Psi_{e_n}$ ;  $\Phi'_{\text{spare}} = \Phi_{\text{spare}}, \Phi_{e_1}, \dots, \Phi_{e_{k-1}}, \Phi_{e_{k+1}}, \dots, \Phi_{e_n}$

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  $C = C_1, \dots, C_n$  and  $C'_k \supseteq C_k$ , by the split subset lemma, we can find a

$C'_1, \dots, C'_n = \Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} = C' \supseteq C$  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:

$\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \phi(\vec{e}') : \phi(\vec{\tau}')$

By (T-EQ), we obtain  $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \phi(\vec{e}') : \tau$

Using (T-MEM) rule,

$\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', \phi(\vec{e}') : \tau)$

7. *Case T – LET*:  $e = \text{let } \langle \vec{x} \rangle = e_a \text{ in } e_b$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \phi(\vec{e}) : \tau$

By Lemma 6.2.1.(7), there is  $\tau'_b \equiv \tau$ .      $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{let } \langle \vec{x} \rangle = e_a \text{ in } e_b : \tau'_b$

and  $\Psi_{e_a}; \Phi_{e_a}; \Delta; \Gamma_{e_a}; B; \text{limit} \vdash e_a : \phi(\vec{\tau}')$ ,      $\Psi_{e_b}; \Phi_{e_b}; \Delta; \Gamma_{e_b}, \vec{x} : \tau'; B; \text{limit} \vdash e_b : \tau'_b$

(*E-LET*)

$e_a = \phi\langle v_1, \dots, v_n \rangle, \quad \Psi_{e_a}; \Phi_{e_a}; \Delta; \Gamma_{e_a}; B; \text{limit} \vdash \phi\langle v_1, \dots, v_n \rangle : \phi\langle \tau'_1, \dots, \tau'_n \rangle,$

$e' = [x_1 \mapsto v_1, \dots, x_n \mapsto v_n]e_b$

By Lemma 6.2.1.(6), there is  $\phi\langle \tau''_1, \dots, \tau''_n \rangle \equiv \phi\langle \tau'_1, \dots, \tau'_n \rangle$

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  $\Psi_{e_b}; \Phi_{e_b}; \Delta; \Gamma_{e_b}, x : \tau'_i; B; \text{limit} \vdash e_b : \tau'_b$

By substitution lemma, we obtain

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash [x_1 \mapsto v_1, \dots, x_n \mapsto v_n]e_b : \tau'_b$

By (T-EQ),  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

8. *Case T-LOAD*:  $e = \text{load}(e_{\text{ptr}}, e_{\text{Has}})$

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

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  $e' = \wedge\langle M(i), \text{fact} \rangle$

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

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

$\{i \mapsto \tau'\}; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \wedge\langle \tau', \text{Has}(I, \tau') \rangle$

By (T-MEM) rule and Weakening Lemma,

$\emptyset; \Phi_e; \Delta; \Gamma; \text{true}; \text{limit} \vdash M(i) : \tau'$

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

By (T-TUPLE) rule,

$\{i \mapsto \tau'\}; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \wedge\langle M(i), \text{fact} \rangle : \wedge\langle \tau', \text{Has}(I, \tau') \rangle$

By (T-EQ) rule,

$\{i \mapsto \tau'\}; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

(*E-LOAD1*)  $e_{\text{ptr}} \rightarrow e'_{\text{ptr}}$

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

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

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

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

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

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

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

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

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

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

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  $C = C_{\text{ptr}}, C_{\text{Has}}$  and  $C'_{\text{ptr}} \supseteq C_{\text{ptr}}$ , by the split subset lemma, we can find a  $C'_{\text{ptr}}, C'_{\text{Has}} = \Psi'_e ; \Phi'_e ; \Delta ; \Gamma ; B ; \text{limit} = C' \supseteq C$  so that  $C'_{\text{Has}} \supseteq C_{\text{Has}}$ . We then use weakening to show  $C'_{\text{Has}} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$ . 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,

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

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

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

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

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

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

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

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

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

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

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

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

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  $C = C_{\text{ptr}}, C_{\text{Has}}$  and  $C'_{\text{Has}} \supseteq C_{\text{Has}}$ , by the split subset lemma, we can find a  $C'_{\text{ptr}}, C'_{\text{Has}} = \Psi'_e ; \Phi'_e ; \Delta ; \Gamma ; B ; \text{limit} = C' \supseteq C$  so that  $C'_{\text{ptr}} \supseteq C_{\text{ptr}}$ . We then use weakening to show  $C'_{\text{ptr}} \vdash v_{\text{ptr}} : \text{Int}(I)$ . 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,

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

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

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

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

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

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

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

(E-STORE)

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

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

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

$\emptyset ; \Phi_{e_v} ; \Delta ; \Gamma_{e_v} ; B ; \text{limit} \vdash e_v : \tau_2, (\Phi ; \Delta \vdash \tau_2 : \dot{1})$

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

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

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

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

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And  $M' = [i \mapsto v]M, \Psi' = [i \mapsto \tau_2]\Psi$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi'(i))$

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

$\Psi'_e = [i \mapsto \tau_2]\Psi_e; \Phi'_e = \Phi_e; M' = [i \mapsto v]M$

(E - STORE1)  $e_{\text{ptr}} \rightarrow e'_{\text{ptr}}$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v) : \text{Has}(I, \tau_2))$

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

Thus, by Lemma 6.2.1.(9) and (T-EQ) rule,

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

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

$(\Psi'_{\text{spare}} = \Psi_{\text{spare}}, \Psi_{e_{\text{Has}}}, \Psi_{e_v}; \Phi'_{\text{spare}} = \Phi_{\text{spare}}, \Phi_{e_{\text{Has}}}, \Phi_{e_v})$

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

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

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

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

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

$\Psi_{e_v}; \Phi_{e_v}; \Delta; \Gamma_{e_v}; B; \text{limit} \vdash e_v : \tau_2$

Given  $C = \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}$  and  $C = C_{\text{ptr}}, C_{\text{Has}}, C_v$  and  $C'_{\text{ptr}} \supseteq C_{\text{ptr}}$ , by the split subset lemma, we can find

a  $C'_{\text{ptr}}, C'_{\text{Has}}, C'_v = \Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} = C' \supseteq C$  so that  $C'_{\text{Has}} \supseteq C_{\text{Has}}$  and  $C'_v \supseteq C_v$ . We then use weakening to show  $C'_{\text{Has}} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$  and  $C'_v \vdash e_v : \tau_2$ . Then using the (T-STORE) rule, we have:

$\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v) : \text{Has}(I, \tau_2)$

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

BY (T-MEM) rule,

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

10. Case T - FIX:  $e = \text{fix } x : \tau_x.v$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \text{fix } x : \tau_x.v : \tau$

By Lemma 6.2.1.(10), there is  $\tau_x \equiv \tau$

and  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \text{fix } x : \tau_x.v : \tau$

$\Psi_e; \Phi_e; \Delta; \Gamma, x : \tau_x; B; \text{limit}_C \vdash v : \tau$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \tau \stackrel{\phi}{:} i$

(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  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

11. Case T - UNROLL:  $e = \text{unroll}(e_0)$   $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{unroll}(e_0) : \tau$

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  $\tau'' \equiv \tau' \equiv (\mu\beta : K.\tau_{\text{roll}})\tau_1 \cdots \tau_n$ , (so  $\tau_0 \equiv \tau_{\text{roll}}$ )

and  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_{\text{roll}} : K$   $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash v : ([\beta \mapsto \mu\beta : K.\tau_{\text{roll}}]_{\tau_{\text{roll}}})\tau_1 \cdots \tau_n$

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  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

12. *Case T – UNPACK*:  $e = \text{unpack } \alpha, x = e_1 \text{ in } e_2$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{unpack } \alpha, x = e_1 \text{ in } e_2 : \tau$

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*)

$e_1 = \text{pack}[\tau_0, v] \text{ as } \exists \beta : K'; \tau_B.\tau'_1$   $e' = [\alpha \mapsto \tau_0, x \mapsto v]e_2$

$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash \text{pack}[\tau_0, v] \text{ as } \exists \beta : K'; \tau_B.\tau'_1 : \exists \alpha : K; B'.\tau_1$

By Lemma 6.2.1.(14), then  $\tau'_1 \equiv \tau_1$ ,  $\tau_B \equiv B'$  and,  $K' \equiv K$ ,

$\exists \beta : K'; \tau_B.\tau'_1 \equiv \exists \alpha : K; B'.\tau_1$

$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash \tau_0 : K$   $B \vdash [\alpha \mapsto \tau_0]_{\tau_B}$

$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash v : [\alpha \mapsto \tau_0]_{\tau'_1}$

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  $\alpha$ ,

$[\alpha \mapsto \tau_0](\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}, x : \tau_1; B, B'; \text{limit}) \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$

$\Rightarrow \Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B, [\alpha \mapsto \tau_0]B'; \text{limit}, x : [\alpha \mapsto \tau_0]\tau_1 \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$

$\Rightarrow \Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit}, x : [\alpha \mapsto \tau_0]\tau_1 \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$

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) rule,  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

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  $e_1 = \text{fact}_1$ ,  $e_2 = \text{fact}_2$ ,  $e' = \wedge \langle \text{know}(I_1 \neq I_2), \text{fact}_1, \text{fact}_2 \rangle$

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)  $\Phi_e; \Delta \vdash \exists \alpha : \text{bool}; I_1 \neq I_2. \cdot \langle \rangle : 0$

by (T-TUPLE)  $\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \cdot \langle \rangle : \cdot \langle \rangle$

$\Rightarrow \emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \cdot \langle \rangle : [\alpha \mapsto \cdot \langle \rangle] \cdot \langle \rangle$ ,

and  $\Phi_e; \Delta \vdash [\alpha \mapsto \cdot \langle \rangle](I_1 \neq I_2)$

By (T-PACK) rule,

$\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{pack}[\text{true}, \cdot \langle \rangle] \text{ as } \exists \alpha : \text{bool}; I_1 \neq I_2. \cdot \langle \rangle : \exists \alpha : \text{bool}; I_1 \neq I_2. \cdot \langle \rangle$

Using abbreviations,  $\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{know}(I_1 \neq I_2) : \text{Know}(I_1 \neq I_2)$

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:

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \wedge \langle \text{know}(I_1 \neq I_2), \text{fact}_1, \text{fact}_2 \rangle : \wedge \langle \text{Know}(I_1 \neq I_2), \text{Has}(I_1, \tau_1), \text{Has}(I_2, \tau_2) \rangle$

Thus,  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

14. *Case T - NEWFUN*:  $e = \text{new\_fun}(J)$

$\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{new\_fun}(J) : \tau$

By Lemma 6.2.1.(16), then  $\exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0) \equiv \tau$

and  $\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{new\_fun}(J) : \exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0)$

(*E - NEWFUN*)

$e' = \text{pack}[F^J, \text{fact}] \text{ as } \exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0)$

By (T-FACT2) rule, we have  $\emptyset; \Phi_e, F^J \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{Gen}(F^J, 0)$

By (K-FUN) rule,  $\Phi_e, F^J \mapsto \text{fun}; \Delta \vdash F^J : \text{int} \rightarrow J$

Because  $\Phi_e; \Delta \vdash 0 : \text{int}$ , and  $\Phi_e, F^J \mapsto \text{fun}; \Delta, \alpha : \text{int} \rightarrow J \vdash \alpha : \text{int} \rightarrow J$

By (K-FUNGEN) rule,  $\Phi_e, F^J \mapsto \text{fun}; \Delta \vdash \text{Gen}(\alpha, 0) : 0$

By (K-SOME) rule, we obtain  $\Phi_e, F^J \mapsto \text{fun}; \Delta \vdash \exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0) : 0$

By (T-PACK) rule, we get

$\emptyset; \Phi_e, F^J \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \text{pack}[F^J, \text{fact}] \text{ as } \exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0) : \exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0)$

Thus,  $\emptyset; \Phi_e, F^J \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash e' : \exists \alpha : \text{int} \rightarrow J.\text{Gen}(\alpha, 0)$

By (T-EQ) rule,  $\emptyset; \Phi_e, F^J \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

We don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

Now  $\Phi'_e \supseteq \Phi_e$ ,  $\Psi'_e = \Psi_e$ ; from (T-NEWFUN) we know  $F^J$  is fresh.

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



15. *Case T – DEFINEFUN*:  $e = \text{define\_fun}(e_0, \tau_a)$ .

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{define\_fun}(e_0, \tau_a) : \tau$

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  $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Gen}(\tau_f, I)$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_f : \text{int} \rightarrow J$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau_a : J$

(*E – DEFINEFUN*)

$e_0 = \text{fact}$ , then  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Gen}(\tau_f, I)$

and  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \text{define\_fun}(e_0, \tau_a) : \tau$

$\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \tau_f : \text{int} \rightarrow J$

$\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \tau_a : J$

Now,  $e' = \wedge \langle \text{fact}, \text{fact}, \text{fact} \rangle, (\text{fun}' \supseteq \text{fun})$

By (T-FACT2) rule,  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}'; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{Gen}(\tau_f, I + 1)$

By (T-FACT3) rule,  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_f I, \tau_a)$

By weakening lemma,  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}'; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_f I, \tau_a)$

By (T-FACT4) rule,  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{InDomain}(I, \tau_f)$

By weakening lemma,  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}'; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{InDomain}(I, \tau_f)$

By (T-TUPLE) rule, we obtain  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}'; \Delta; \Gamma; B; \text{limit} \vdash$

$\wedge \langle \text{fact}, \text{fact}, \text{fact} \rangle : \wedge \langle \text{Gen}(\tau_f, I + 1), \text{Eq}(\tau_f I, \tau_a), \text{InDomain}(I, \tau_f) \rangle$

By (T-EQ) rule,  $\emptyset; \Phi_e, \tau_f \hat{\mapsto} \text{fun}'; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

Now  $\Phi'_e \supseteq \Phi_e, \tau_f \hat{\mapsto} \text{fun}; \Psi'_e = \Psi_e;$

By (T-MEM),  $\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$

16. *Case T – DISCARDFUN*:  $e = \text{discard\_fun}(e_0)$ .

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{discard\_fun}(e_0) : \tau'$

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  $\emptyset; \Phi_{e_0}, \tau \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Gen}(\tau, I)$

$\emptyset; \Phi_{e_0}, \tau \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \text{define\_fun}(e_0, \tau_a) : \tau$

Now  $e' = \cdot \langle \rangle$ , and by (T-TUPLE) rule  $C \vdash \cdot \langle \rangle : \cdot \langle \rangle$

Thus,  $\emptyset; \Phi_{e_0}, \tau \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \cdot \langle \rangle : \cdot \langle \rangle$

By (T-EQ) rule, we obtain  $\emptyset; \Phi_{e_0}, \tau \hat{\mapsto} \text{fun}; \Delta; \Gamma; B; \text{limit} \vdash \cdot \langle \rangle : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

We don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

Now  $\Phi'_e \supseteq \Phi_e, \Psi'_e = \Psi_e;$

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

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

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

(E – INDOMAIN)

$e_1 = \text{fact}_1, e_2 = \text{fact}_2$  then

$\emptyset; \Phi_{e_1}, \tau_f \mapsto \text{fun}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash e_1 : \text{InDomain}(I_1, \tau_f)$

$\emptyset; \Phi_{e_2}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_{e_2}; B; \text{limit} \vdash e_2 : \text{Gen}(\tau_f, I_2)$

$\emptyset; \Phi_{e_0}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash e : \tau$

$e' = \wedge \langle \text{know}(0 \leq I_1 \wedge I_1 < I_2), \text{fact} \rangle$

By (T-FACT2) rule, we get  $\emptyset; \Phi_{e_0}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash \text{fact} : \text{Gen}(\tau_f, I_2)$

From definition, we know  $\langle 0 \leq I_1 \wedge I_1 < I_2 \rangle = \text{pack}[\text{true}, \cdot \langle \rangle]$  as  $\exists \alpha : \text{bool}; 0 \leq I_1 \wedge I_1 < I_2. \cdot \langle \rangle$

Because by (K-BOOL)  $\Phi_{e_0}, \tau_f \mapsto \text{fun}; \Delta \vdash \text{true} : \text{bool}$ ,

and by (K-SOME)  $\Phi_{e_0}, \tau_f \mapsto \text{fun}; \Delta \vdash \exists \alpha : \text{bool}; 0 \leq I_1 \wedge I_1 < I_2. \cdot \langle \rangle : 0$

$\emptyset; \Phi_{e_0}, \tau_f \mapsto \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash \cdot \langle \rangle : [\alpha \mapsto \text{true}] \cdot \langle \rangle$ ,

and  $B \vdash [\alpha \mapsto \text{true}](0 \leq I_1 \wedge I_1 < I_2)$

By (T-PACK) rule, we obtain:

$\emptyset; \Phi_{e_0}, \tau_f \mapsto \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash \text{pack}[\text{true}, \cdot \langle \rangle] \text{ as } \exists \alpha : \text{bool}; 0 \leq I_1 \wedge I_1 < I_2. \cdot \langle \rangle : \exists \alpha : \text{bool}; 0 \leq I_1 \wedge I_1 < I_2. \cdot \langle \rangle$

By using abbreviations,  $\emptyset; \Phi_{e_0}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash \text{know}(0 \leq I_1 \wedge I_1 < I_2) : \text{Know}(0 \leq I_1 \wedge I_1 < I_2)$

By (T-TUPLE) rule, we obtain:

$\emptyset; \Phi_{e_0}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash \wedge \langle \text{know}(0 \leq I_1 \wedge I_1 < I_2), \text{fact} \rangle : \wedge \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2) \rangle$

Then,  $\emptyset; \Phi_{e_0}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash e' : \wedge \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2) \rangle$

By (T-EQ) rule,  $\emptyset; \Phi_{e_0}, \tau_f \hat{\mapsto} \text{fun}; \Delta; \Gamma_e; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

Here we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

18. *Case T – MAKEEQ*:  $e = \text{make\_eq}(\tau_0)$

$\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{make\_eq}(\tau_0) : \tau$

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,  $\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_0, \tau_0)$

Then  $\emptyset; \Phi_e; \Delta; \Gamma_e; B; \text{limit} \vdash e' : \text{Eq}(\tau_0, \tau_0)$

Thus,  $\emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

Because  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

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

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

19. *Case T – APPLYEQ*:  $e = \text{apply\_eq}(\tau_f, e_1, e_2)$

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau$$

By Lemma 6.2.1.(21), then  $\tau_f \tau_b \equiv \tau$

$$\text{and } \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{apply\_eq}(\tau_f, e_1, e_2) : \tau_f \tau_b$$

$$\Phi_e; \Delta \vdash \tau_f : K \rightarrow J \quad \Phi_e; \Delta \vdash \tau_a : K \quad \Phi_e; \Delta \vdash \tau_b : K$$

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_1 : \text{Eq}(\tau_a, \tau_b) \quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_2 : \tau_f \tau_a$$

(*E – APPLYEQ*)

$$e_1 = \text{fact}, e_2 = v, \text{ and } e' = v$$

$$\text{By (T-FACT3) rule, } \emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_1 : \text{Eq}(\tau_{e_1}, \tau_{e_1}),$$

$$\text{then } \tau_{e_1} \equiv \tau_a \equiv \tau_b, \text{ and } \emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e : \tau_f \tau_b,$$

$$\text{Thus, } \emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_2 : \tau_f \tau_a \Rightarrow \emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash v : \tau_f \tau_b$$

$$\Rightarrow \emptyset; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau_f \tau_b$$

$$\text{Because } \Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$$

And we don't change M, then  $M' = M$

$$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$$

$$\text{By (T-MEM) rule, } \Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$$

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

20. *Case T – CASE*:  $e = \text{case}(b, e_0)$

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{case}(b, e_0) : \tau$$

$$\text{By Lemma 6.2.1.(22), then } \tau \equiv \tau_i \quad (i = \begin{matrix} 1 \text{ if } b \\ 2 \text{ if } \neg b \end{matrix})$$

$$\text{and } \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_0 : \text{Union}(b, \tau_1, \tau_2)$$

(*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

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau'_1 : K, \quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau'_2 : K$$

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash v : \tau'_i \quad (i = \begin{matrix} 1 \text{ if } \tau_b \\ 2 \text{ if } \neg \tau_b \end{matrix})$$

$$\text{Thus, } \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau'_i \quad (i = \begin{matrix} 1 \text{ if } b \\ 2 \text{ if } \neg b \end{matrix})$$

$$\text{By (T-EQ) rule, } \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$$

$$\text{Because } \Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$$

And we don't change M, then  $M' = M$

$$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$$

$$\text{By (T-MEM) rule, } \Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$$

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

21. *Case T – COERCE*:  $e = \text{coerce}(e_0)$

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e_0) : \tau$$

By Lemma 6.2.1.(24), then there is  $\tau' \equiv \tau$

$$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e_0) : \tau'$$

$$\text{and } \Psi_e; \Phi_e; \Delta; \Gamma; B; I_2 \vdash e_0 : \tau'$$

(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  $\Psi_{\text{spare}}, \Psi_e; \Phi_{\text{spare}}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$

And we don't change M, then  $M' = M$

$\forall i \in \text{dom}(\Psi). (\emptyset; \Phi; \emptyset; \emptyset; \text{true}; \infty \vdash M'(i) : \Psi(i))$

By (T-MEM) rule,  $\Psi_{\text{spare}}, \Psi'_e; \Phi_{\text{spare}}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$

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

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:

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

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

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

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

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

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

$$(G) \quad \frac{\forall i. (\Phi; B \vdash \tau_i \Rightarrow \tau'_i)}{\Phi; B \vdash T[\tau_1, \dots, \tau_n] \Rightarrow T[\tau'_1, \dots, \tau'_n]}$$

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

6. case  $\tau_a = I_a \xrightarrow{D} \tau_b = I_b$  and  $\tau_a = I_a \xrightarrow{D} \tau_c = I_c$ . Then  $I_a \doteq I_b \doteq I_c$ , so pick  $\tau_d = I_a$ .
7. case  $\tau_a = B_a \xrightarrow{E} \tau_b = B_b$  and  $\tau_a = B_a \xrightarrow{E} \tau_c = B_c$ . Then  $B_a \doteq B_b \doteq B_c$ , so pick  $\tau_d = B_a$ .
8. case  $\tau_a = F^K I_a \xrightarrow{G} \tau_b = \delta(i_b)$ . If  $\tau_a \xrightarrow{G} \tau_c = \delta(i_c)$ , then  $i_b = i_c$ , so choose  $\tau_d = \tau_b = \tau_c$ . If  $\tau_a = F^K I_a \xrightarrow{G} \tau_c = F^K I_c$ , 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  $C \vdash \tau_a \xrightarrow{*} \tau_b$ , and  $C \vdash \tau_a \xrightarrow{*} \tau_c$ , then there is some  $\tau_d$  so that  $C \vdash \tau_b \xrightarrow{*} \tau_d$  and  $C \vdash \tau_c \xrightarrow{*} \tau_d$ . Standard proof by tiling single parallel steps to fill in the area between  $\tau_a \xrightarrow{*} \tau_b$  and  $\tau_a \xrightarrow{*} \tau_c$ .

### 6.3.3 CORROLARY [CONFLUENCE FOR EQUIVALENT TYPES]

If  $C \vdash \tau_b \equiv \tau_c$ , then there is some  $\tau_d$  so that  $C \vdash \tau_b \xrightarrow{*} \tau_d$  and  $C \vdash \tau_c \xrightarrow{*} \tau_d$ . 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:

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

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

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

$$\tau = \tau_1 \xrightarrow{\phi, \text{limit}} \tau_2 \mid \langle \vec{\tau} \rangle \mid \forall\alpha : K; B.\tau$$

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

$$\mid \text{Has}(I, \tau) \mid \text{Gen}(\tau, I) \mid \text{Eq}(\tau_1, \tau_2) \mid \text{InDomain}(I, \tau)$$

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  $\tau$  has a value shape, then  $\tau'$  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  $\tau_b$  and  $\tau_c$  have value shapes, there is some  $\tau_d$  so that  $C \vdash \tau_b \xrightarrow{*} \tau_d$  and  $C \vdash \tau_c \xrightarrow{*} \tau_d$ , and  $\tau_d$  has the same shape as  $\tau_b$  and  $\tau_c$ , which implies that  $\tau_b$  and  $\tau_c$  have the same shape.

### 6.3.5 LEMMA [CANONICAL FORMS]

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

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

7. If  $C \vdash v : \text{Bool}(B)$ , then  $v = b$ .
8. If  $C \vdash v : \text{Union}(b, \tau_1, \tau_2)$ , then  $v = \text{union}(b, \tau_1, \tau_2, v_0)$ .
9. If  $C \vdash v : \text{Has}(I, \tau)$  or  $C \vdash v : \text{Gen}(\tau, I)$  or  $C \vdash v : \text{Eq}(\tau_1, \tau_2)$  or  $C \vdash v : \text{InDomain}(I, \tau)$ , then  $v = \text{fact}$ .

*Proof:*

1. Only two type checking rules can prove  $C \vdash v : \tau_1 \xrightarrow{\phi, \text{limit}} \tau_2$ : (T-EQ) and (T-ABS).  
For (T-EQ) rule, there is some  $\tau' \equiv \tau_1 \xrightarrow{\phi, \text{limit}} \tau_2$ . By shape preservation,  $\tau' = \tau'_1 \xrightarrow{\phi, \text{limit}} \tau'_2$ . Use induction.  
By (T-ABS) rule, we know  $v = \lambda x : \tau_1 \xrightarrow{\phi, \text{limit}} e$  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  $\tau$  and  $C_{Me} = \Psi_{Me}; \Phi_{Me}; \emptyset; \emptyset; B; \text{limit}$ ), then either  $e$  is a value or else there is some  $(M', e')$  so that  $(M, e) \rightarrow (M', e')$ .

Observe that by the rule for typing  $(M, e)$ , we know that  $\Psi_{Me} = \Psi_{\text{spare}}, \Psi$  and  $\Phi_{Me} = \Phi_{\text{spare}}, \Phi$  and  $C \vdash e : \tau$ , where  $C = \Psi; \Phi; \emptyset; \emptyset; B; \text{limit}$ . Now prove that  $(M, e)$  steps, by induction on the derivation of  $C \vdash e : \tau$  (holding  $M, \Psi_{Me}$ , and  $\Phi_{Me}$  fixed throughout the induction as  $C, e$ , and  $\tau$  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  $C_0 = \Psi_0; \Phi_0; \emptyset; \emptyset; B; \text{limit}_0$  and  $\Psi_{Me} = \Psi_{0\text{-spare}}, \Psi_0$   $\Phi_{Me} = \Phi_{0\text{-spare}}, \Phi_0$ , so by induction,  $(M, e_0) \rightarrow (M', e'_0)$ , so  $(M, e) = (M, E[e_0]) \rightarrow (M', E[e'_0])$ . 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.

$$\frac{C_1, C_2 = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}_C \quad C_1 \vdash v_1 : \tau_a \xrightarrow{\phi, \text{limit}_f} \tau_b \quad C_2 \vdash v_2 : \tau_a}{(\text{limit}_C = \text{limit}_f = \infty) \text{ or } (B \vdash \text{limit}_f < \text{limit}_C)}$$

2.  $e = v_1 v_2$ , where (T-APP)  $\frac{\text{or } (\text{limit}_C = \infty, \text{limit}_f = I, \Phi; \Delta \vdash \tau_b : 0^{\phi_b})}{C_1, C_2 \vdash v_1 v_2 : \tau_b}$ .

By canonical forms,  $v_1 = \lambda x : \tau_a \xrightarrow{\phi, \text{limit}_f} e_0$ , so  $e$  steps by (E-APPABS).

$$\frac{C \vdash v_1 : \forall \alpha : K; B. \tau_1 \quad C \vdash \tau_2 : K}{C \vdash [\alpha \mapsto \tau_2] B}$$

3.  $e = v_1 \tau_2$ , where (T-TAPP)  $\frac{C \vdash [\alpha \mapsto \tau_2] B}{C \vdash v_1 \tau_2 : [\alpha \mapsto \tau_2] \tau_1}$ .

By canonical forms,  $v_1 = \Lambda \alpha : K; B. v$ , so  $e$  steps by (E-TAPPABS).

$$C_1 \vdash v_1 : \exists \alpha : K; B. \tau_1$$

4.  $e = \text{unpack } \alpha, x = v_1 \text{ in } e_2$ , where (T-UNPACK)  $\frac{C_2, \alpha : K, x : \tau_1, B \vdash e_2 : \tau_2}{C_1, C_2 \vdash \text{unpack } \alpha, x = v_1 \text{ in } e_2 : \tau_2}$ .

By canonical forms,  $v_1 = \text{pack}[\tau_1, v]$  as  $\exists \alpha : K; B. \tau_2$ , so  $e$  steps by (E-UNPACK).

$$C_1, C_2 \vdash I_1 : \text{int} \quad C_1, C_2 \vdash I_2 : \text{int}$$

5.  $e = \text{in\_domain}(I_1, I_2, v_1, v_2)$ , where (T-INDOMAIN)  $\frac{C_1 \vdash v_1 : \text{InDomain}(I_1, \tau_f) \quad C_2 \vdash v_2 : \text{Gen}(\tau_f, I_2)}{C_1, C_2 \vdash \text{in\_domain}(I_1, I_2, v_1, v_2) : \wedge (\text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2))}$ .

By canonical forms,  $v_1 = \text{fact}$  and  $v_2 = \text{fact}$ , 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  $e_1 \text{ op } e_2$ ,  $\neg e$ ,  $\text{case}(b, e)$ ,  $\text{unroll}(e)$ ,  $\text{apply\_eq}(\tau, e_1, e_2)$ ,  $\text{discard\_fun}(e)$ ,  $\text{define\_fun}(e, \tau)$ ,  $\text{distinguish}(I_1, I_2, e_1, e_2)$ , and  $\text{in\_domain}(I_1, I_2, e_1, e_2)$  follow exactly the same pattern.

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

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

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

10.  $e = \text{if } B \text{ then } e_1 \text{ else } e_2$ , where  $(T - IFB) \frac{C \vdash B : \text{bool} \quad C, B \vdash e_1 : \tau \quad C, \neg B \vdash e_2 : \tau}{C \vdash \text{if } B \text{ then } e_1 \text{ else } e_2 : \tau}$ .  
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.  $e = \text{load}(v_{\text{ptr}}, v_{\text{Has}})$ , where  $(T - LOAD) \frac{C_1 \vdash v_{\text{ptr}} : \text{Int}(I) \quad C_2 \vdash v_{\text{Has}} : \text{Has}(I, \tau)}{C_1, C_2 \vdash \text{load}(v_{\text{ptr}}, v_{\text{Has}}) : \langle \tau, \text{Has}(I, \tau) \rangle}$ .  
By canonical forms,  $v_{\text{ptr}} = i_{\text{ptr}}$ , and  $v_{\text{Has}} = \text{fact}$ . By inversion,  $C_1 \vdash i_{\text{ptr}} : \text{Int}(i_{\text{ptr}})$  and  $C_2 = C'_2, i_{\text{has}} \mapsto \tau \vdash \text{fact} : \text{Has}(i_{\text{has}}, \tau)$ . Together, these imply  $i_{\text{ptr}} = i_{\text{has}}$ . Since  $i_{\text{has}} \mapsto \tau \in \Psi$  and memory  $M$  is well-typed under  $C_{Me} = \Psi_{Me}; \dots$ , where  $\Psi_{Me} = \Psi_{\text{spare}}, \Psi$ , we know  $i_{\text{has}} \mapsto \tau \in \Psi_{Me}$ , so  $M(i_{\text{has}})$  exists, so  $M(i_{\text{ptr}})$  exists, so  $e$  steps by (E-LOAD).

12.  $e = \text{store}(v_{\text{ptr}}, v_{\text{Has}}, v_v)$ , where  $(T - STORE) \frac{C_1 \vdash v_{\text{ptr}} : \text{Int}(I) \quad C \vdash \tau_2 : \dot{1}}{C \vdash \text{store}(v_{\text{ptr}}, v_{\text{Has}}, v_v) : \text{Has}(I, \tau_2)}$ .  
By canonical forms,  $v_{\text{ptr}} = i_{\text{ptr}}$ , and  $v_{\text{Has}} = \text{fact}$ , 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  $\Psi_{Me}; \Phi_{Me}; \emptyset; \emptyset; \text{true}; i \vdash (M, e : \tau)$ , 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:

$$\begin{aligned} \text{depth}(x) &= 0 \\ \text{depth}(\lambda x : \tau \xrightarrow{\phi} e) &= 0 \\ \text{depth}(\text{fix } x.v) &= 0 \\ \text{depth}(\text{let } x_1, \dots, x_n = e_1 \text{ in } e_2) &= \max(\text{depth}(e_1), 1 + \text{depth}(e_2)) \\ \text{depth}(\text{unpack } \alpha, x = e_1 \text{ in } e_2) &= \max(\text{depth}(e_1), 1 + \text{depth}(e_2)) \end{aligned}$$

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

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

Lemma 0:  $\text{depth}(v) = 0$ . 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:  $\text{depth}([x \mapsto v]e) \leq \text{depth}(e)$ . 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  $\Psi_{Me}; \Phi_{Me}; \emptyset; \emptyset; \text{true}; i \vdash (M, e : \tau)$  and  $\Psi; \Phi; \emptyset; \emptyset; \text{true}; i \vdash e : \tau$  and  $(M, e) \rightarrow (M', e')$  by a non-congruence rule, then  $M = M'$  and one of these three must be true:

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

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

*Proof :*

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

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

$\Psi; \Phi; \emptyset; \emptyset; \text{true}; i \vdash (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2 : \tau_b$ ,

$\Psi_1; \Phi_1; \emptyset; \emptyset; \text{true}; i_f \vdash e_1 : \tau_a \xrightarrow{f} \tau_b$ , and  $(i > i_f)$

$\Psi_2; \Phi_2; \emptyset; \emptyset; \text{true}; i \vdash v_2 : \tau_a$

By Limit-Change Lemma,  $\Psi_2; \Phi_2; \emptyset; \emptyset; \text{true}; i_f \vdash v_2 : \tau_a$

By Term Substitution Lemma, we obtain:

$\Psi; \Phi; \emptyset; \emptyset; \text{true}; i_f \vdash [x \mapsto v_2]e_1 : \tau_b$

$\Psi; \Phi; \emptyset; \emptyset; \text{true}; i_f \vdash e'' : \tau_b$

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.  $e = \text{let } x_1, \dots, x_n = e_1 \text{ in } e_2$  ( $e_1 = \phi\langle v_1, \dots, v_n \rangle$ )

$e' = [x_1 \mapsto v_1, \dots, x_n \mapsto v_n]e_2$

In Preservation Lemma we have proved,  $\Psi; \Phi; \emptyset; \emptyset; \text{true}; i \vdash e' : \tau$

From lemma 1, we know  $\text{depth}([x_1 \mapsto v_1, \dots, x_n \mapsto v_n]e_2) \leq e_2$

And  $\text{depth}(\text{let } x_1, \dots, x_n = e_1 \text{ in } e_2) = \max(\text{depth}(e_1), 1 + \text{depth}(e_2))$ ,

so  $\text{depth}(e') < \text{depth}(\text{let } x_1, \dots, x_n = e_1 \text{ in } e_2)$

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,  $\Psi; \Phi; \emptyset; \emptyset; \text{true}; i \vdash e' : \tau$

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,  $\Psi; \Phi; \emptyset; \emptyset; \text{true}; i \vdash e' : \tau$ .

$\text{size}(\text{load}(i, \text{fact})) = 1 + \text{size}(i) + \text{size}(\text{fact}) > \text{size}(\wedge\langle M(i), \text{fact} \rangle) = 1$ , 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.

$\text{size}(\text{coerce}(v)) = 1 + \text{size}(v) > \text{size}(v) = 1$

$\text{size}((\Lambda \alpha : K; B.v)\tau) = 1 + \text{size}(v) > \text{size}(v) = 1$

$\text{size}(\text{if } v_1 \text{ then } e_2 \text{ else } e_3) = 1 + \text{size}(v_1) + \text{size}(e_2) + \text{size}(e_3) > \text{size}(e_2) \text{ or } \text{size}(e_3)$

$\text{size}(\text{if } B \text{ then } e_2 \text{ else } e_3) = 1 + \text{size}(e_2) + \text{size}(e_3) > \text{size}(e_2) \text{ or } \text{size}(e_3)$

$\text{size}(\text{unroll}(\text{roll}[\tau](v))) = 1 + 1 + \text{size}(v) > \text{size}(v)$

$\text{size}(\text{fix } x : \tau.v) = 1 + \text{size}(v) > \text{size}([x \mapsto \text{fix } x : \tau.v]v) = 1$  (By Lemma 2)

$\text{size}(\text{case}(b, \text{union}(b, \tau_1, \tau_2, v))) = 1 + 1 + \text{size}(b) + \text{size}(v) + \text{size}(b) > \text{size}(v) = 1$

$\text{size}(\text{in\_domain}(I_1, I_2, \text{fact}, \text{fact})) = 1 + \text{size}(I_1) + \text{size}(I_2) + \text{size}(\text{fact}) + \text{size}(\text{fact}) > \text{size}(e') = 1$

$\text{size}(\text{distinguish}(I_1, I_2, \text{fact}, \text{fact})) = \text{size}(I_1) + \text{size}(I_2) + \text{size}(\text{fact}) + \text{size}(\text{fact}) > \text{size}(e') = 1$

$\text{size}(\text{make\_eq}(\tau)) > \text{size}(\text{fact}) = 1$

$\text{size}(\text{apply\_eq}(\tau, \text{fact}, v)) > \text{size}(v) = 1$

$\text{size}(\text{new\_fun}(K)) > \text{size}(e') = 1$

$\text{size}(\text{discard\_fun}(\text{fact})) > \text{size}(\cdot) = 1$

$\text{size}(\text{define\_fun}(\text{fact}, \tau)) > \text{size}(\wedge\langle \text{fact}, \text{fact}, \text{fact} \rangle) = 1$

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



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

## 7.2 THEOREM [STRONG NORMALIZATION]

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

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

*Proof :*

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

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

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

If  $e$  is 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  $\Psi_{Me} = \Psi_{\text{spare}}, \Psi$  and  $\Phi_{Me} = \Phi_{\text{spare}}, \Phi$  and  $C \vdash e : \tau$ , where  $C = \Psi; \Phi; \emptyset; \emptyset; B; i$ , and by inspection of the type checking rules, we conclude that there is some  $C_1 = \Psi_1; \Phi_1; \emptyset; \emptyset; B; i_1$ , 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  $\text{depth}(e_1) \leq \text{depth}(e)$ , and  $\text{size}(e_1) < \text{size}(e)$ . Therefore, either  $i_1 < i$ , or  $i_1 = i$  and  $\text{depth}(e_1) < \text{depth}(e)$ , or  $i_1 = i$  and  $\text{depth}(e_1) = \text{depth}(e)$  and  $\text{size}(e_1) < \text{size}(e)$ , 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  $i$ :  $\Psi'_{Me}; \Phi'_{Me}; \emptyset; \emptyset; \text{true}; i \vdash (M, e_c : \tau)$ .

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

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

1.  $e' = \text{coerce}(e'')$  and  $i' < i$ ,  $\Psi'; \Phi'; \emptyset; \emptyset; \text{true}; i' \vdash e'' : \tau$ . 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.  $\Psi'; \Phi'; \emptyset; \emptyset; \text{true}; i \vdash e' : \tau$ , and  $\text{depth}(e') < \text{depth}(e_c)$ . By induction on  $\text{depth}(e)$ , 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.  $\Psi'; \Phi'; \emptyset; \emptyset; \text{true}; i \vdash e' : \tau$ , and  $\text{depth}(e') = \text{depth}(e_c)$ , and  $\text{size}(e') < \text{size}(e_c)$ . By induction on  $\text{size}(e)$ , 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  $\tau$  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  $\text{erase}((M, e)) \xrightarrow{?} \text{erase}((M', e'))$ .
2. (Erasure-Progress-2) If  $\text{erase}(e)$  is a value then  $(M, e) \xrightarrow{*} (M', v)$ ,  $\text{erase}((M, e)) = \text{erase}((M', v))$ .
3. (Erasure-Progress-3) If  $\text{erase}((M, e)) \mapsto (L', d')$  then  $(M, e) \xrightarrow{+} (M', e')$ ,  $\text{erase}((M', e')) = (L', d')$ .

In the cases where  $e$  does not effect memory ( $M = M'$ ),  $(M, e) \mapsto (M', e')$  is abbreviated 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.  $\text{erase}(\phi(\vec{v})) = \langle \text{erase}(v_1), \text{erase}(v_2), \dots, \text{erase}(v_n) \rangle$  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  $\text{Kind } \overset{\phi}{0}$  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),  $i : \text{Int}(i) : \overset{\phi}{1}$ . This does not have  $\text{Kind } \overset{\phi}{0}$ .
- Case:  $b$ . By Inversion and (T-BOOL),  $b : \text{Bool}(b)$ . By (K-BOOL),  $b : \text{Bool}(b) : \overset{\phi}{1}$ . This does not have  $\text{Kind } \overset{\phi}{0}$ .
- 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:  $\text{pack}[\tau_1, v]$  as  $\exists\alpha : K; B.\tau_2$ . By Inversion and (T-PACK),  $\text{pack}[\tau_1, v]$  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),  $\text{erase}(\text{pack}[\tau_1 : K, v])$  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  $K = \overset{\phi}{0}$ ,  $\text{erase}(\text{pack}[\tau_1 : \overset{\phi}{0}, v])$  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),  $\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) : ((\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n)$  and  $v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$  and  $\tau_0 : K$ . By (K\_REC),  $(\mu\alpha : K.\tau_0) : K$ . Therefore,  $v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n : K$  and  $\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n](v) : ((\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n) : K$ . 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  $K = \overset{\phi}{0}$ ,  $\text{erase}(\text{roll}[(\mu\alpha : \overset{\phi}{0}.\tau_0)\tau_1 \cdots \tau_n](v) : ((\mu\alpha : \overset{\phi}{0}.\tau_0)\tau_1 \cdots \tau_n) : \overset{\phi}{0}) = \text{erase}(v([\alpha \mapsto \mu\alpha : \overset{\phi}{0}.\tau_0]\tau_0)\tau_1 \cdots \tau_n : \overset{\phi}{0}) = \langle \rangle$ .
- Case:  $\lambda x : \tau \xrightarrow{\phi, \text{limit}} e$ . By (T-ABS),  $\lambda x : \tau \xrightarrow{\phi, \text{limit}} e : \tau_1 \xrightarrow{\phi, \text{limit}} \tau_2$ . If  $\text{limit} = \infty$ , by (K-ABS),  $\tau_1 \xrightarrow{\phi, \infty} \tau_2 : \overset{\phi}{1}$ . This does not have  $\text{Kind } \overset{\phi}{0}$ . If  $\text{limit} = I$ , by (K-ABS),  $\tau_1 \xrightarrow{\phi, I} \tau_2 : \overset{\phi}{0}$  and by (ER-FUNI)  $\text{erase}(\lambda x : \tau \xrightarrow{\phi, I} e) = \langle \rangle$ .
- Case:  $\phi(\vec{v})$ . BY (ER-TUPLEv0),  $\text{erase}(\phi(\vec{v}) : \tau : \overset{\phi}{0}) = \langle \rangle$ .
- 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:  $\text{fact}$ . 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]

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

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  $\text{erase}((M, e)) \mapsto \text{erase}((M', e'))$  using the corresponding untyped (or typed) evaluation rule.**

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

*Proof :*

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

**8.2.2 Case:  $\text{erase}(e) = \text{erase}(e')$ .**

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

*Proof :*

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

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

1. ( $E - \text{ABSAPP1}$ ) If  $e \xrightarrow{(E-\text{ABSAPP1})} e'$  then  $e = (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2$  and  $e' = \text{coerce}([x \mapsto v_2]e_1)$ .
2. ( $E - \text{APPLYEQ}$ ) If  $e \xrightarrow{(E-\text{APPLYEQ})} e'$  then  $e = \text{apply\_eq}(\tau, \text{fact}, v)$  and  $e' = v$ .
3. ( $E - \text{INDOMAIN}$ ) If  $e \xrightarrow{(E-\text{INDOMAIN})} e'$  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  $e \xrightarrow{(E-DISTINGUISH)} e'$  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  $e \xrightarrow{(E-DISCARDFUN)} e'$  then  $e = \text{discard\_fun}(\text{fact})$  and  $e' = \cdot \langle \rangle$ .
6. ( $E - DEFINEFUN$ ) If  $e \xrightarrow{(E-DEFINEFUN)} e'$  then  $e = \text{define\_fun}(\text{fact}, \tau)$  and  $e' = \wedge \langle \text{fact}, \text{fact}, \text{fact} \rangle$ .

*Proof :*

( $E - DEFINEFUN$ ) If  $e \xrightarrow{(E-DEFINEFUN)} e'$  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: $\text{erase}((M, e)) \xrightarrow{?} \text{erase}((M', e'))$ using congruence.

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

*Proof :*

(*Congruence rule*) If  $(M, e) \xrightarrow{(congruence\ rule)E[e]=e\tau} (M', e')$  then  $e = e_1\tau$  and  $e' = e'_1\tau$  and  $(M, e_1) \mapsto (M', e'_1)$ .

By induction,  $\text{erase}((M, e_1)) \xrightarrow{?} \text{erase}((M', e'_1))$ .  $\text{erase}((M, e)) = \text{erase}((M, e_1\tau)) = (\text{erase}(M), \text{erase}(e_1\tau)) = (\text{erase}(M), \text{erase}(e_1))$  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  $(\text{erase}(M), \text{erase}(e_1)) \xrightarrow{?} (\text{erase}(M'), \text{erase}(e'_1))$  and  $\text{erase}((M, e)) \xrightarrow{?} \text{erase}((M', e'))$ .

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

1. ( $E - COERCE$ ) If  $e \xrightarrow{(E-COERCE)} e'$  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: $\text{erase}((M, e)) \xrightarrow{?, (congruence\ rule)E[e]=\text{let } x=e \text{ in } e_2} \text{erase}((M', e'))$ .

1. (*Congruence rule*)  $E[e] = \text{distinguish}(I_1, I_2, e, e_2) \mid \text{distinguish}(I_1, I_2, v_1, e) \mid \text{define\_fun}(e, \tau) \mid \text{in\_domain}(I_1, I_2, e, e_2) \mid \text{in\_domain}(I_1, I_2, v_1, e)$

*Proof :*

(*Congruence rule*) If  $(M, e) \xrightarrow{(congruence\ rule)E[e]=\text{distinguish}(I_1, I_2, e, e_2)} (M', e')$  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,  $\text{erase}((M, e_1)) \xrightarrow{?} \text{erase}((M', e'_1))$ .  $\text{erase}((M, e)) = \text{erase}((M, \text{distinguish}(I_1, I_2, e_1, e_2))) = (\text{erase}(M), \text{erase}(\text{distinguish}(I_1, I_2, e_1, e_2)))$  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  $(\text{erase}(M), \text{erase}(e_1)) \xrightarrow{?} (\text{erase}(M'), \text{erase}(e'_1))$  and  $(\text{erase}(M), \langle \text{erase}(e_1), \text{erase}(e_2) \rangle) \xrightarrow{?, (congruence\ rule)E[e]=\phi \langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (\text{erase}(M'), \langle \text{erase}(e'_1), \text{erase}(e_2) \rangle)$  so,  $(\text{erase}(M), \text{let } \langle x, y \rangle = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle) \xrightarrow{?, (congruence\ rule)E[e]=\text{let } x=e \text{ in } e_2} (\text{erase}(M'), \text{let } \langle x, y \rangle = \langle \text{erase}(e'_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle)$ .

### 8.2.7 Case: unpack.

1. ( $E-UNPACK$ ) If  $e \xrightarrow{(E-UNPACK)} e'$  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 :*

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

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

1. ( $E-FIX$ ) If  $e \xrightarrow{(E-FIX)} e'$  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  $e' = [x \mapsto \text{fix } x : t : \overset{\phi}{0}.v]v$ .  $\text{erase}(e) = \text{erase}(\text{fix } x : t : \overset{\phi}{0}.v) = \langle \rangle$  by (ER-FIX0).  $\text{erase}(e') = \text{erase}([x \mapsto \text{fix } x : t : \overset{\phi}{0}.v]v) = [x \mapsto \text{erase}(\text{fix } x : t : \overset{\phi}{0}.v)] \text{erase}(v)$  by Erase-Term-Substitution. This equals  $[x \mapsto \langle \rangle] \langle \rangle$  by (ER-FIX0) and Zero-Erase-Value. This equals  $\langle \rangle$ . Therefore,  $\text{erase}(e) = \text{erase}(e')$ .

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

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

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

$$c = i \mid b \mid c\tau \mid \phi(\vec{c}) : \vec{\tau} : \overset{\phi}{i} \text{ where } i > 0 \mid \phi(\vec{v}) : \vec{\tau} : \overset{\phi}{0} \mid \lambda x : \tau \xrightarrow{\phi, \text{limit}} e$$

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

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

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

### 8.3.1 LEMMA [C-ERASE-VALUE]

$\text{erase}(c) = u$ . Proof by induction on  $c$ .

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

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

$\text{erase}(c)$  is either a value or another  $\text{erase}(c)$ . In the case of  $\langle \overrightarrow{\text{erase}(c)} \rangle$ , each  $\text{erase}(c)$  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,  $\text{erase}(e) \neq u$ . Proof by induction on the expressions.

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

13. By (ER-CASE),  $\text{erase}(\text{case}(b, e))$  where  $e \neq c = \text{erase}(e)$  where  $e \neq c$ .
14. By (ER-ROLL),  $\text{erase}(\text{roll}[(\mu\alpha : K.\tau_0)\tau_1 \dots \tau_n](e))$  where  $e \neq c = \text{erase}(e)$  where  $e \neq c$ .
15. By (ER-UNROLL),  $\text{erase}(\text{unroll}(e))$  where  $e \neq c = \text{erase}(e)$  where  $e \neq c$ .
16. By (ER-FIX),  $\text{erase}(\text{fix } x : \tau : \overset{\phi}{i} . v \text{ where } i > 0) = \text{fix } x . \text{erase}(v)$ .
17. By (ER-LOAD),  $\text{erase}(\text{load}(e_{\text{ptr}}, e_{\text{Has}})) = \text{load}(\text{erase}(e_{\text{ptr}}), \text{erase}(e_{\text{Has}}))$ .
18. By (ER-STORE),  $\text{erase}(\text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v)) = \text{store}(\text{erase}(e_{\text{ptr}}), \text{erase}(e_{\text{Has}}), \text{erase}(e_v))$ .
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  $\text{csize}(c)$  as:

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



### 8.3.4 LEMMA [C-SIZE-DECREASES]

If  $C \vdash c : \tau$ , where  $C$  has an empty  $\Delta$  and  $\Gamma$ , and  $(M, c)$  steps, then there is some  $c'$  so that  $(M, c) \xrightarrow{\pm} (M, c')$  and  $\text{csize}(c) > \text{csize}(c')$ . 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.  $\text{coerce}(v) \xrightarrow{(E-COERCE)} v$ .  $\text{csize}(\text{coerce}(v)) > \text{csize}(v)$ .
2.  $(\Lambda\alpha : K; B.v)\tau \xrightarrow{(E-TAPPTABS)} [\alpha \mapsto \tau]v$ .  $\text{csize}((\Lambda\alpha : K; B.v)\tau) = 1$ . By Type-Substitution-Value,  $[\alpha \mapsto \tau]v$  is a value so  $\text{csize}([\alpha \mapsto \tau]v) = 0$ .
3.  $\text{unroll}(\text{roll}[\tau](v)) \xrightarrow{(E-UNROLL)} v$ .  $\text{csize}(\text{unroll}(\text{roll}[\tau](v))) > 0$ .
4.  $\text{fix } x : t : \overset{\phi}{0} . v \xrightarrow{(E-FIX)} [x \mapsto \text{fix } x : t : \overset{\phi}{0} . v]v$ . 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.  $\text{case}(b, \text{union}(b, \tau_1, \tau_2, v)) \xrightarrow{(E-CASE)} v$ .  $\text{csize}(\text{case}(b, \text{union}(b, \tau_1, \tau_2, v))) = 2 > 0$ .
6.  $\text{make\_eq}(\tau) \xrightarrow{(E-MAKEEQ)} \text{fact}$ .  $\text{csize}(\text{make\_eq}(\tau)) = 1 > 0$ .
7.  $\text{new\_fun}(K) \xrightarrow{(E-NEWFUN)} \text{pack}[F^K, \text{fact}] \text{ as } \exists\alpha : \text{int} \mapsto K.\text{FunGen}(\alpha, 0)$ , where  $F^K$  is fresh.  $\text{csize}(\text{new\_fun}(K)) = 1 > 0$ .
8.  $\text{discard\_fun}(\text{fact}) \xrightarrow{(E-DISCARDFUN)} \langle \rangle$ .  $\text{csize}(\text{discard\_fun}(\text{fact})) = 1 > 0$ .
9.  $c \xrightarrow{(congruence\ rule)c=E[c_1]} c'$ . 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.  $\text{coerce}(e_1) \xrightarrow{(congruence\ rule)e=E[e_1]} \text{coerce}(e'_1)$ . By inversion,  $e_1$  type-checks in an environment with  $\text{limit} = i$  for some  $i$ . Therefore we can use strong normalization to say  $(M, e_1) \xrightarrow{*} (M, v_1)$ . By congruence,  $(M, \text{coerce}(e_1)) \xrightarrow{*} (M, \text{coerce}(v_1))$ . In one additional step,  $(M, \text{coerce}(v_1)) \xrightarrow{(E-COERCE)} (M, v_1)$ . 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  $\Delta$  and  $\Gamma$ , then  $(M, c) \xrightarrow{*} (M, v)$ .

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

## 8.4 THEOREM [ERASURE-PROGRESS-2]

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

If  $\text{erase}(e)$  is a value, then by Erase-C-Value,  $e = c$ . By C-\*Step-Value,  $(M, e) \xrightarrow{*} (M, v)$ . By Preservation,  $e : \tau$  and  $v : \tau$ . Either  $e = v$  and  $\text{erase}((M, e)) = \text{erase}((M, v))$  or  $e \neq v$  and  $(M, e) \xrightarrow{\pm} (M, v)$ . If  $e \neq v$ ,  $e \mapsto e_1 \xrightarrow{\text{repeatedly}} e_n \mapsto v$ . By Erasure-Progress-1,  $\text{erase}((M, e)) \xrightarrow{?} \text{erase}((M, e_1)) \xrightarrow{?, \text{repeatedly}} \text{erase}((M, e_n)) \xrightarrow{?} \text{erase}((M, v))$ . Since  $\text{erase}(e)$  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  $\Delta$  and  $\Gamma$  and  $\text{erase}(e)$  is a non value, then  $\text{erase}((M, e)) \mapsto (L', d')$ . Proof by induction on the expressions.

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

$$\begin{aligned} e = i \mid b \mid \phi \langle \vec{v} \rangle : \tau : \dot{0} \mid \lambda x : \tau \xrightarrow{\phi, \text{limit}} e_1 \\ \mid \Lambda \alpha : K ; B.v \mid \text{fix } x : \tau : \dot{0} .v \mid \text{coerce}(e_1) \\ \mid \text{make\_eq}(\tau) \mid \text{new\_fun}(K) \mid \text{fact} \end{aligned}$$

2. Case:  $e = x$ . This won't typecheck in an empty environment.
3. Case:  $e = \text{if } B \text{ then } e_1 \text{ else } e_2$ . This only appears inside coercion functions and will be removed by erase of the coercion function so erase does not apply to this case.
4. Case:  $e = e_1 \tau \mid \text{union}(b, \tau_1, \tau_2, e_1) \mid \text{pack}[\tau_1, e_1] \text{ as } \exists \alpha : K ; B.\tau_2 \mid \text{case}(b, e) \mid \text{roll}[(\mu \alpha : K.\tau_0)\tau_1 \cdots \tau_n](e) \mid \text{unroll}(e)$ .  
The proofs for all of these are similar so we will only show one.  
 $e = e_1 \tau$ . By (ER-APPT),  $\text{erase}(e_1 \tau) = \text{erase}(e_1)$ . If  $e_1 = c$ , by C-Erase-Value  $\text{erase}(e_1) = u$  so  $\text{erase}(e) = u$ . If  $e_1 \neq c$ ,  $e_1$  is not a value. By Non-C-Erase-Non-Value  $\text{erase}(e_1) \neq u$ . By induction,  $\text{erase}((M, e_1)) \mapsto (L', d')$ . Therefore,  $\text{erase}((M, e)) \mapsto (L', d')$ .
5. Case:  $e = e_1 e_2$ .

If  $e_1 : (\tau_1 \xrightarrow{\phi, \infty} \tau_2)$ ,  $\text{erase}((M, e)) = (\text{erase}(M), \text{erase}(e_1) \text{erase}(e_2))$  by (ER-M) and (ER-APP).

If  $e_1$  is in the set  $c$ , then by C-\*Step-Value,  $(M, e_1) \xrightarrow{*} (M, v_1)$ .  $(\text{erase}(M), \text{erase}(e_1) \text{erase}(e_2)) \xrightarrow{*, (\text{congruence rule}) E[e]=e e_2} (\text{erase}(M), \text{erase}(v_1) \text{erase}(e_2))$ .

By Inversion and Canonical Forms,  $v_1 = \lambda x : \tau_1 \xrightarrow{\phi, \infty} e_{11}$ .  $\text{erase}(v_1) = \lambda x \longrightarrow \text{erase}(e_{11})$ . If  $\text{erase}(e_2)$  is a value, then  $(\text{erase}(M), (\lambda x \longrightarrow \text{erase}(e_{11})) \text{erase}(e_2)) \xrightarrow{(U-ABSAPP)} (\text{erase}(M), [x \mapsto \text{erase}(e_2)] \text{erase}(e_{11}))$ . If  $\text{erase}(e_2)$  is not a value, then by induction  $\text{erase}((M, e_2)) \mapsto (L', d_2)$ .  
 $(\text{erase}(M), \text{erase}(v_1) \text{erase}(e_2)) \xrightarrow{(\text{congruence rule}) E[e]=v e} (L', \text{erase}(v_1) d_2)$ .

If  $e_1$  is not in the set  $c$ , then by Not-C-Erase-Not-Value  $\text{erase}(e_1)$  is a non value. By induction  $\text{erase}((M, e_1)) \mapsto (L', d_1)$ .  $(\text{erase}(M), \text{erase}(e_1) \text{erase}(e_2)) \xrightarrow{(\text{congruence rule}) E[e]=e e_2} (L', d_1 \text{erase}(e_2))$ .

If  $e_1 : (\tau_1 \xrightarrow{\phi, I} \tau_2)$ ,  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle)$  by (ER-M) and (ER-APPI).

If  $\text{erase}(e_1)$  is a value, if  $\text{erase}(e_2)$  is a value, then let  $x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle \xrightarrow{(E-LET)} \langle \rangle$ . If  $\text{erase}(e_2)$  is not a value, then by induction  $\text{erase}((M, e_2)) \mapsto (L', d_2)$ .  $(\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle) \xrightarrow{(\text{congruence rule}) E[e]=\phi \langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L', \text{let } x = \langle \text{erase}(e_1), d_2 \rangle \text{ in } \langle \rangle)$ .

If  $\text{erase}(e_1)$  is not a value, then by induction  $\text{erase}((M, e_1)) \mapsto (L', d_1)$ .  $(\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle) \xrightarrow{(\text{congruence rule}) E[e]=\phi \langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L', \text{let } x = \langle d_1, \text{erase}(e_2) \rangle \text{ in } \langle \rangle)$ .

6. Case:  $e = \phi \langle \vec{e}_1 \rangle$ . If  $e = \phi \langle \vec{e}_1 \rangle : \vec{\tau}_1 : \dot{i}$  where  $i > 0$ , by (ER-TUPLE) and (ER-M)  $\text{erase}((M, e)) = (\text{erase}(M), \langle \text{erase}(e_{11}), \dots, \text{erase}(e_{1n}) \rangle)$ . If  $\text{erase}(e)$  is a non value then some  $\text{erase}(e_{1i})$  is a non value. By induction,  $\text{erase}((M, e_{1i})) \mapsto (L', d_{1i})$ .  $(\text{erase}(M), \langle \text{erase}(e_{11}), \dots, \text{erase}(e_{1n}) \rangle) \xrightarrow{(\text{congruence rule}) E[e]=\phi \langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L', \langle \text{erase}(e_{11}), \dots, d_{1i}, \dots, \text{erase}(e_{1n}) \rangle)$ . If  $e = \phi \langle \vec{e}_1 \rangle : \vec{\tau}_1 : \dot{0}$ , by (ER-TUPLEe0) and (ER-M)  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \phi \langle \text{erase}(e_1) \rangle \text{ in } \langle \rangle)$  where some  $\text{erase}(e_{1i})$  is not a value. By induction  $\text{erase}((M, e_{1i})) \mapsto (L', d_{1i})$ .  $(\text{erase}(M), \langle \text{erase}(e_{11}), \dots, \text{erase}(e_{1n}) \rangle) \xrightarrow{(\text{congruence rule}) E[e]=\phi \langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L', \langle \text{erase}(e_{11}), \dots, d_{1i}, \dots, \text{erase}(e_{1n}) \rangle)$ .
7. Case:  $e = e_1 \text{ op } e_2 \mid \neg e_1 \mid \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \mid \text{unpack } \alpha, x = e_1 \text{ in } e_2 \mid \text{load}(e_{\text{ptr}}, e_{\text{Has}}) \mid \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_v) \mid \text{distinguish}(I_1, I_2, e_1, e_2) \mid \text{apply\_eq}(\tau, e_1, e_2) \mid \text{discard\_fun}(e_1) \mid \text{define\_fun}(e_1, \tau) \mid \text{in\_domain}(I_1, I_2, e_1, e_2)$ .

*Proof :*

$e = \text{if } e_1 \text{ then } e_2 \text{ else } e_3$ . By (ER-IF) and (ER-M),  $\text{erase}((M, e)) = (\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3))$ .

If  $e_1$  is in the set  $c$ , then by (C-\*Step-Value)  $(M, e_1) \mapsto^* (M, v_1)$ . Since  $v_1 : \text{Bool}(B)$ ,  $v_1 = b$  by Canonical Forms.

So,  $\text{erase}(v_1) = b$  by (ER-b). If  $b = \text{true}$  then  $(\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \xrightarrow{(E-IF1)} (\text{erase}(M), \text{erase}(e_2))$ .

If  $b = \text{false}$  then  $(\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \xrightarrow{(E-IF2)} (\text{erase}(M), \text{erase}(e_3))$ .

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  $(\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \xrightarrow{(congruence\ rule)E[e]=\text{if } e \text{ then } e_2 \text{ else } e_3} (L', \text{if } d' \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3))$ .

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

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

## 8.5 THEOREM [ERASURE-PROGRESS-3]

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

If  $\text{erase}(e)$  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:

$$e = i \mid b \mid \Lambda \alpha : K ; B.v \mid \text{pack}[\tau_1, v] \text{ as } \exists \alpha : K ; B.\tau_2 \mid \lambda x : \tau \xrightarrow{\phi, \text{limit}} e_1 \\ \mid \text{roll}[(\mu \alpha : K.\tau_0)\tau_1 \dots \tau_n](v) \mid \phi(\vec{v}) \mid \text{union}(b, \tau_1, \tau_2, v) \mid \text{fact}$$

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

### 8.5.2 Case: Most of the non values:

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

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  $\text{erase}(e)$  is a value and Erasure-Progress-3 does not apply.
- If  $\text{erase}(e_1)$  is a non value, then  $\text{erase}(e)$  is a non value.  $\text{erase}((M, e_1)) \mapsto (L', d'_1)$  by Untyped-Non-Values-Step and  $(M, e_1) \overset{\pm}{\mapsto} (M', e'_1)$ ,  $\text{erase}((M', e'_1)) = (L', d'_1)$  by induction.  $(M, e) = (M, e_1 \tau) \xrightarrow{+, (congruence\ rule)E[e]=e\tau} (M', e'_1 \tau) = (M', e')$  so  $(M, e) \overset{\pm}{\mapsto} (M', e')$ .  $\text{erase}((M, e)) = (\text{erase}(M), \text{erase}(e)) = (\text{erase}(M), \text{erase}(e_1)) \mapsto (L', d'_1) = (L', d')$  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 $e = \text{unpack } \alpha, x = e_1 \text{ in } e_2$ .

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

- Let  $\text{erase}(e_1)$  be a value.

$(M, e_1) \xrightarrow{*} (M, v_1)$ , and  $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$  by Erasure-Progress-2.  $(M, e) = (M, \text{unpack } \alpha, x = e_1 \text{ in } e_2)$   
 $\xrightarrow{*, (\text{congruence rule}) E[e]=\text{unpack } \alpha, x=e \text{ in } e_2} (M, \text{unpack } \alpha, x = v_1 \text{ in } e_2)$ . By Canonical Forms,  $v_1 = \text{pack}[\tau_1, v_{11}] \text{ as } \exists \alpha_1 : K ; B.\tau_2$ . So  $(M, \text{unpack } \alpha, x = v_1 \text{ in } e_2) = (M, \text{unpack } \alpha, x = \text{pack}[\tau_1, v_{11}] \text{ as } \exists \alpha_1 : K ; B.\tau_2 \text{ in } e_2) \xrightarrow{(E-UNPACK)} (M, [\alpha \mapsto \tau_1, x \mapsto v_{11}]e_2) = (M, e')$ .  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \text{erase}(e_1) \text{ in } \text{erase}(e_2)) \xrightarrow{(E-LET)} (\text{erase}(M), [x \mapsto \text{erase}(e_1)] \text{erase}(e_2)) = (L', d')$ .  $\text{erase}(v_1) = \text{erase}(\text{pack}[\tau_1, v_{11}] \text{ as } \exists \alpha_1 : K ; B.\tau_2) = \text{erase}(v_{11})$  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  $(M, e_1) \xrightarrow{\perp} (M', e'_1)$ ,  $\text{erase}((M', e'_1)) = (L', d_1)$  by induction.  $(M, e) = (M, \text{unpack } \alpha, x = e_1 \text{ in } e_2) \xrightarrow{+, (\text{congruence rule}) E[e]=\text{unpack } \alpha, x=e \text{ in } e_2} (M', \text{unpack } \alpha, x = e'_1 \text{ in } e_2) = (M', e')$ .  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \text{erase}(e_1) \text{ in } \text{erase}(e_2)) \xrightarrow{(\text{congruence rule}) E[e]=\text{let } x=e \text{ in } e_2} (\text{erase}(M'), \text{let } x = d_1 \text{ in } \text{erase}(e_2)) = (L', d')$ .  $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\text{unpack } \alpha, x = e'_1 \text{ in } e_2)) = (\text{erase}(M'), \text{let } x = \text{erase}(e'_1) \text{ in } \text{erase}(e_2))$  by (ER-M) and (ER-UNPACK).  $(\text{erase}(M'), \text{let } x = \text{erase}(e'_1) \text{ in } \text{erase}(e_2)) = (L', \text{let } x = d_1 \text{ in } \text{erase}(e_2)) = (L', d')$ .

### 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,  $e_1 : (\tau_1 \xrightarrow{\phi, \text{limit}} \tau_2)$  has limit  $I$  or  $\infty$ .

- Let limit =  $\infty$  so,  $e_1 : (\tau_1 \xrightarrow{\phi, \infty} \tau_2)$ .  
 $\text{erase}(e) = \text{erase}(e_1 : (\tau_1 \xrightarrow{\phi, \infty} \tau_2) e_2) = \text{erase}(e_1) \text{erase}(e_2)$  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  $(M, e_1) \xrightarrow{\perp} (M', e'_1)$ ,  $\text{erase}((M', e'_1)) = (L', d_1)$  by induction.  $\text{erase}((M, e)) = (\text{erase}(M), \text{erase}(e_1) \text{erase}(e_2)) \xrightarrow{(\text{congruence rule}) E[e]=e e_2} (\text{erase}(M'), d_1 \text{erase}(e_2)) = (L', d')$ .  $(M, e) = (M, e_1 e_2) \xrightarrow{+, (\text{congruence rule}) E[e]=e e_2} (M', e'_1 e_2) = (M', e')$ .  $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(e'_1 : (\tau_1 \xrightarrow{\phi, \infty} \tau_2) e_2)) = (\text{erase}(M'), \text{erase}(e'_1) \text{erase}(e_2))$  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,  $(M, e_1) \xrightarrow{*} (M, v_1)$ . 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,  $(M, e_1) \xrightarrow{*} (M, v)$ . By Inversion and Canonical Forms,  $v = \lambda x : \tau \xrightarrow{\phi, \text{limit}} e_{11}$ . By Erasure-Progress-2  $\text{erase}(e_1) = \text{erase}(v)$  so  $\text{erase}(e_1) = \text{erase}(v) = \lambda x \longrightarrow \text{erase}(e_{11})$ .  $\text{erase}(e) = (\lambda x \longrightarrow \text{erase}(e_{11})) \text{erase}(e_2) \xrightarrow{(U-ABSAPP)} [x \mapsto \text{erase}(e_2)] \text{erase}(e_{11}) = d'$ .  $e = (\lambda x : \tau_1 \xrightarrow{\phi, \infty} e_{11}) e_2 \xrightarrow{(E-ABSAPP2)} [x \mapsto e_2] e_{11} = e'$ .  $\text{erase}(e') = \text{erase}([x \mapsto e_2] e_{11}) = [x \mapsto \text{erase}(e_2)] \text{erase}(e_{11}) = d'$  by Erase-Term-Substitution.

- Let  $\text{limit} = I$  so,  $e_1 : (\tau_1 \xrightarrow{\phi, I} \tau_2)$ .  
 $\text{erase}(e) = \text{erase}(e_1 : (\tau_1 \xrightarrow{\phi, I} \tau_2) e_2) = \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle$  by (ER-APPI).
- Let  $\text{erase}(e_1)$  and  $\text{erase}(e_2)$  be values.  
 $(M, e_1) \xrightarrow{*} (M, v_1)$ , and  $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$  by Erasure-Progress-2.  $(M, e_2) \xrightarrow{*} (M, v_2)$ , and  $\text{erase}((M, e_2)) = \text{erase}((M, v_2))$  by Erasure-Progress-2.  
 $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle) \xrightarrow{(E-LET)} (\text{erase}(M), \langle \rangle) = (L', d')$ . ( $L' = \text{erase}(M)$ ).  $(M, e) = (M, e_1 e_2) \xrightarrow{*, (\text{congruence rule}) E[e]=e e_2} (M, v_1 v_2)$   
 $\xrightarrow{*, (\text{congruence rule}) E[e]=v e} (M, v_1 v_2)$ . By Inversion and Canonical Forms,  $e_1 = \lambda x : \tau \xrightarrow{\phi, I} e_{11}$ .  $(M, v_1 v_2) = (M, (\lambda x : \tau \xrightarrow{\phi, I} e_{11}) v_2) \xrightarrow{(E-ABSAPP1)} (M, \text{coerce}([x \mapsto v_2]e_{11})) = (M, e')$ .  
 $\text{erase}((M, e')) = (\text{erase}(M) \text{erase}(\text{coerce}([x \mapsto v_2]e_{11}))) = (\text{erase}(M), \langle \rangle) = (L', d')$  by (ER-COERCE).
- Let  $\text{erase}(e_1)$  be a value and  $\text{erase}(e_2)$  be a non value.  
 $(M, e_1) \xrightarrow{*} (M, v_1)$ , 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  $(M, e_2) \xrightarrow{\dagger} (M'_2, e'_2)$ ,  $\text{erase}((M'_2, e'_2)) = (L'_2, d_2)$  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  $(\text{erase}(M), \langle \text{erase}(e_1), \text{erase}(e_2) \rangle)$   
 $\xrightarrow{(\text{congruence rule}) E[e]=\phi\langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L'_2, \langle \text{erase}(e_1), d_2 \rangle)$ ,  $(\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle)$   
 $\xrightarrow{(\text{congruence rule}) E[e]=\text{let } \vec{x}=e \text{ in } e_2} (L'_2, \text{let } x = \langle \text{erase}(e_1), d_2 \rangle \text{ in } \langle \rangle) = (L', d')$ . ( $L' = L'_2$ ).  
 $(M, e) = (M, e_1 e_2) \xrightarrow{*, (\text{congruence rule}) E[e]=e e_2} (M, v_1 e_2)$   
 $\xrightarrow{+, (\text{congruence rule}) E[e]=v e} (M'_2, v_1 e'_2) = (M', e')$ . ( $M' = M'_2$ ). 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  $(M, e_1) \xrightarrow{\dagger} (M', e'_1)$ ,  $\text{erase}((M', e'_1)) = (L', d_1)$  by induction.  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle)$ . Since  $(\text{erase}(M), \langle \text{erase}(e_1), \text{erase}(e_2) \rangle)$   
 $\xrightarrow{(\text{congruence rule}) E[e]=\phi\langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (\text{erase}(M'), \langle d_1, \text{erase}(e_2) \rangle)$ ,  $(\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle)$   
 $\xrightarrow{(\text{congruence rule}) E[e]=\text{let } \vec{x}=e \text{ in } e_2} (L', \text{let } x = \langle d_1, \text{erase}(e_2) \rangle \text{ in } \langle \rangle) = (L', d')$ .  $(M, e) = (M, e_1 e_2) \xrightarrow{+, (\text{congruence rule}) E[e]=e e_2} (M', e'_1 e_2) = (M', e')$ . By Preservation,  $e'_1 : (\tau_1 \xrightarrow{\phi, I} \tau_2)$ .  $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(e'_1 e_2)) = (\text{erase}(M'), \text{let } x = \langle \text{erase}(e'_1), \text{erase}(e_2) \rangle \text{ in } \langle \rangle) = (L', \text{let } x = \langle d_1, \text{erase}(e_2) \rangle \text{ in } \langle \rangle) = (L', d')$ .

### 8.5.6 Case: Let $e = \phi(\vec{e})$ .

- Let  $e = \phi\langle e_1, e_2, \dots, e_n \rangle : t : i$  where  $i > 0$   
 $\text{erase}(e) = \text{erase}(\phi\langle e_1, e_2, \dots, e_n \rangle : t : i) = \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle$  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  $(M, e_k) \xrightarrow{\dagger} (M', e'_k)$ ,  $\text{erase}((M', e'_k)) = (L', d_k)$  by induction.  
 $\text{erase}((M, e)) = (\text{erase}(M), \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle)$   
 $\xrightarrow{(\text{congruence rule}) E[e]=\phi\langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L', \langle \text{erase}(e_1), \dots, d_k, \dots, \text{erase}(e_n) \rangle) = (L', d')$ .  
 $(M, e) = (M, \phi\langle e_1, \dots, e_k, \dots, e_n \rangle : t : i) \xrightarrow{+, (\text{congruence rule}) E[e]=\phi\langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (M', \phi\langle e_1, \dots, e'_k, \dots, e_n \rangle : t : i) = (M', e')$ .  
 $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\phi\langle e_1, \dots, e_k, \dots, e_n \rangle : t : i)) = (\text{erase}(M'), \langle \text{erase}(e_1), \dots, \text{erase}(e_k), \dots, \text{erase}(e_n) \rangle) = (L', \langle \text{erase}(e_1), \dots, d_k, \dots, \text{erase}(e_n) \rangle) = (L', d')$  by (ER-M) and (ER-TUPLE).
- Let  $e = \phi\langle e_1, e_2, \dots, e_n \rangle : t : \dot{0}$   
 $\text{erase}(e) = \text{erase}(\phi\langle e_1, e_2, \dots, e_n \rangle : t : \dot{0}) = \text{let } x = \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle \text{ in } \langle \rangle$  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  $\text{erase}((M, e_k)) \mapsto (L', d_k)$  by Untyped-Non-Values-Step and  $(M, e_k) \xrightarrow{+} (M', e'_k)$ ,  $\text{erase}((M', e'_k)) = (L', d_k)$  by induction.  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle \text{ in } \langle \rangle)$ . Since

$$(\text{erase}(M), \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle) \xrightarrow{+, (\text{congruence rule})E[e]=\phi\langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (L', \langle \text{erase}(e_1), \dots, d_k, \dots, \text{erase}(e_n) \rangle)$$

$$(\text{erase}(M), \text{let } x = \langle \text{erase}(e_1), \dots, \text{erase}(e_n) \rangle \text{ in } \langle \rangle) \xrightarrow{+, (\text{congruence rule})E[e]=\text{let } \vec{x}=e \text{ in } e_2} (L', \text{let } x = \langle \text{erase}(e_2), \dots, d_k, \dots, \text{erase}(e_n) \rangle \text{ in } \langle \rangle) = (L', d')$$

$$\xrightarrow{+, (\text{congruence rule})E[e]=\phi\langle v_1, \dots, v_{k-1}, e, e_{k+1}, \dots, e_n \rangle} (M', \phi\langle e_1, e_2, \dots, e_n \rangle : t : \dot{0}) = (M', e')$$

$$\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\phi\langle e_1, \dots, e'_k, \dots, e_n \rangle : t : \dot{0})) = (\text{erase}(M'), \text{let } x = \langle \text{erase}(e_1), \dots, \text{erase}(e'_k), \dots, \text{erase}(e_n) \rangle \text{ in } \langle \rangle) = (L', \text{let } x = \langle \text{erase}(e_1), \dots, d_k, \dots, \text{erase}(e_n) \rangle \text{ in } \langle \rangle) = (L', d')$$

by (ER-M) and (ER-TUPLEe0).

### 8.5.7 Case: Let $e = e_1 \text{ op } e_2 \mid \text{let } x = e_1 \text{ in } e_2$ .

These cases are similar so we only show one.

Let  $e = e_1 \text{ op } e_2$ .

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

- Let  $\text{erase}(e_1)$  and  $\text{erase}(e_2)$  be values.

$(M, e_1) \xrightarrow{*} (M, v_1)$  and  $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$  by Erasure-Progress-2. Similarly,  $(M, e_2) \xrightarrow{*} (M, v_2)$  and  $\text{erase}((M, e_2)) = \text{erase}((M, v_2))$  by Erasure-Progress-2.  $(M, e) = (M, e_1 \text{ op } e_2) \xrightarrow{*, (\text{congruence rule})E[e]=e \text{ op } e_2} (M, v_1 \text{ op } v_2)$  and  $(M, v_1 \text{ op } v_2) \xrightarrow{*, (\text{congruence rule})E[e]=v \text{ op } e} (M, v_1 \text{ op } v_2)$ . By canonical forms,  $v_1$  and  $v_2$  are both integers for an integer op, or booleans for a boolean op, so that  $\text{simplify}(v_1 \text{ op } v_2)$  exists and

$$(M, v_1 \text{ op } v_2) \xrightarrow{(E-SIMPLIFY1)} (M, \text{simplify}(v_1 \text{ op } v_2)) = (M, e')$$

$\text{erase}((M, e)) = (\text{erase}(M), \text{erase}(e_1) \text{ op } \text{erase}(e_2)) \xrightarrow{(E-SIMPLIFY1)} ((\text{erase}(M), \text{simplify}(\text{erase}(e_1) \text{ op } \text{erase}(e_2))) = (L', d')$

$(\text{erase}(M) = L')$ .  $\text{erase}((M, e')) = (\text{erase}(M), \text{erase}(\text{simplify}(v_1 \text{ op } v_2)))$

$= (\text{erase}(M), \text{simplify}(\text{erase}(v_1) \text{ op } \text{erase}(v_2)))$  by (ER-M) and Erase-Simplify. This equals

$(\text{erase}(M), \text{simplify}(\text{erase}(v_1) \text{ op } \text{erase}(v_2))) = (\text{erase}(M), \text{simplify}(\text{erase}(e_1) \text{ op } \text{erase}(e_2))) = (L', d')$ .

- Let  $\text{erase}(e_1)$  be a value and  $\text{erase}(e_2)$  be a non value.

$(M, e_1) \xrightarrow{*} (M, v_1)$  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  $(M, e_2) \xrightarrow{+} (M'_2, e'_2)$  and  $\text{erase}((M'_2, e'_2)) = (L'_2, d_2)$  by induction.  $(M, e) = (M, e_1 \text{ op } e_2)$

$\xrightarrow{*, (\text{congruence rule})E[e]=e \text{ op } e_2} (M, v_1 \text{ op } e_2)$  and  $(M, v_1 \text{ op } e_2) \xrightarrow{+, (\text{congruence rule})E[e]=v \text{ op } e} (M'_2, v_1 \text{ op } e'_2) = (M', e')$ .

$(M'_2 = M')$ .  $\text{erase}((M, e)) = (\text{erase}(M), \text{erase}(e_1) \text{ op } \text{erase}(e_2)) \xrightarrow{(\text{congruence rule})E[e]=v \text{ op } e} (L'_2, \text{erase}(e_1) \text{ op } d_2) = (L', d')$ .  $(L'_2 = L')$ .  $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(v_1 \text{ op } e'_2)) = (\text{erase}(M'), \text{erase}(v_1) \text{ op } \text{erase}(e'_2))$  by (ER-M) and (ER-OP). This equals  $(\text{erase}(M'_2), \text{erase}(v_1) \text{ op } \text{erase}(e'_2)) = (L'_2, \text{erase}(e_1) \text{ op } d_2) = (L', d')$ .

- Let  $\text{erase}(e_1)$  be a non value.

$\text{erase}((M, e_1)) \mapsto (L', d_1)$  by Untyped-Non-Values-Step and  $(M, e_1) \xrightarrow{+} (M', e'_1)$ ,  $\text{erase}((M', e'_1)) = (L', d_1)$  by induction.  $(M, e) = (M, e_1 \text{ op } e_2) \xrightarrow{+, (\text{congruence rule})E[e]=e \text{ op } e_2} (M', e'_1 \text{ op } e_2) = (M', e')$ .  $\text{erase}((M, e)) =$

$(\text{erase}(M), \text{erase}(e_1) \text{ op } \text{erase}(e_2)) \xrightarrow{(\text{congruence rule})E[e]=e \text{ op } e_2} (L', d_1 \text{ op } \text{erase}(e_2)) = (L', d')$ .  $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(e'_1 \text{ op } e_2)) = (\text{erase}(M'), \text{erase}(e'_1) \text{ op } \text{erase}(e_2))$  by (ER-M) and (ER-OP). This equals  $(L', d_1 \text{ op } \text{erase}(e_2)) = (L', d')$ .

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

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

- Let  $\text{erase}(e_1)$  be a value.

$(M, e_1) \xrightarrow{*} (M, v_1)$  and  $\text{erase}((M, e_1)) = \text{erase}((M, v_1))$  by Erasure-Progress-2.

By canonical forms,  $v_1$  must be a boolean.

- Let  $v_1$  be true.

$$(M, e) = (M, \text{if } e_1 \text{ then } e_2 \text{ else } e_3) \xrightarrow{*, (\text{congruence rule}) E[e]=\text{if } e \text{ then } e_2 \text{ else } e_3} (M, \text{if } v_1 \text{ then } e_2 \text{ else } e_3) \text{ and}$$

$$(M, \text{if } v_1 \text{ then } e_2 \text{ else } e_3) \xrightarrow{(E-IF1)} (M, e_2) = (M, e'). \quad \text{erase}(v_1) = \text{erase}(\text{true}) = \text{true} \text{ so } \text{erase}(e_1) \text{ is true.}$$

$$\text{erase}((M, e)) = (\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \xrightarrow{(E-IF1)} (\text{erase}(M), \text{erase}(e_2)) = (L', d').$$

$$\text{erase}((M, e')) = (\text{erase}(M), \text{erase}(e_2)) = (L', d').$$

- Let  $v_1$  be false.

$$(M, e) = (M, \text{if } e_1 \text{ then } e_2 \text{ else } e_3) \xrightarrow{*, (\text{congruence rule}) E[e]=\text{if } e \text{ then } e_2 \text{ else } e_3} (M, \text{if } v_1 \text{ then } e_2 \text{ else } e_3) \text{ and}$$

$$(M, \text{if } v_1 \text{ then } e_2 \text{ else } e_3) \xrightarrow{(E-IF2)} (M, e_3) = (M', e'). \quad \text{erase}(v_1) = \text{erase}(\text{false}) = \text{false} \text{ so } \text{erase}(e_1) \text{ is false.}$$

$$\text{erase}((M, e)) = (\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \xrightarrow{(E-IF2)} (\text{erase}(M), \text{erase}(e_3)) = (L', d').$$

$$\text{erase}((M', e')) = (\text{erase}(M), \text{erase}(e_3)) = (L', d').$$

- Let  $\text{erase}(e_1)$  be a non value.

$$\text{erase}((M, e_1)) \mapsto (M', d_1) \text{ by Untyped-Non-Values-Step and } (M, e_1) \xrightarrow{\perp} (M', e'_1), \quad \text{erase}((M', e'_1)) = (L', d_1)$$

$$\text{by induction. } (M, e) = (M, \text{if } e_1 \text{ then } e_2 \text{ else } e_3) \xrightarrow{+, (\text{congruence rule}) E[e]=\text{if } e \text{ then } e_2 \text{ else } e_3} (M, \text{if } e'_1 \text{ then } e_2 \text{ else } e_3) =$$

$$(M', e'). \quad \text{erase}((M, e)) = (\text{erase}(M), \text{if } \text{erase}(e_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \xrightarrow{(\text{congruence rule}) E[e]=\text{if } e \text{ then } e_2 \text{ else } e_3}$$

$$(L', \text{if } d_1 \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) = (L', d'). \quad \text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\text{if } e'_1 \text{ then } e_2 \text{ else } e_3)) =$$

$$(\text{erase}(M'), \text{if } \text{erase}(e'_1) \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) \text{ by (ER-M) and (ER-IF). This equals}$$

$$(L', \text{if } d_1 \text{ then } \text{erase}(e_2) \text{ else } \text{erase}(e_3)) = (L', d').$$

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

This expression only occurs inside type coercion functions so  $\text{if } B \text{ then } e_1 \text{ else } e_2$  will be removed by an  $\text{erase}()$  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 $e = \text{fix } x : t : i . v$ .

- Let  $i > 0$

$$\text{erase}(e) = \text{erase}(\text{fix } x : t : i . v) = \text{fix } x. \text{erase}(v) \text{ by (ER-FIX). } e = \text{fix } x : t : i . v \xrightarrow{(E-FIX)} [x \mapsto \text{fix } x : t : i . v]$$

$$.v]v = e'. \quad \text{erase}(e) = \text{fix } x. \text{erase}(v) \xrightarrow{(U-FIX)} [x \mapsto \text{fix } x. \text{erase}(v)] \text{erase}(v) = d'. \quad \text{erase}(e') = \text{erase}([x \mapsto$$

$$\text{fix } x : t : i . v]v) = [x \mapsto \text{erase}(\text{fix } x : t : i . v)] \text{erase}(v) = d' \text{ by Erase-Term-Substitution. This equals } [x \mapsto$$

$$\text{fix } x. \text{erase}(v)]v = d'.$$

- Let  $i = 0$

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

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

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

Let  $(M, e) = (M, \text{load}(e_{ptr}, e_{Has}))$ .

$$\text{erase}((M, e)) = \text{erase}((M, \text{load}(e_{ptr}, e_{Has}))) = (\text{erase}(M), \text{erase}(\text{load}(e_{ptr}, e_{Has})))$$

$$= (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \text{ by (ER-M) and (ER-LOAD).}$$

- Let  $\text{erase}(e_{ptr})$  and  $\text{erase}(e_{Has})$  be values.

$$(M, e_{ptr}) \xrightarrow{*} (M, v_{ptr}) \text{ and } \text{erase}((M, e_{ptr})) = \text{erase}((M, v_{ptr})) \text{ by Erasure-Progress-2. } (M, e_{Has}) \xrightarrow{*} (M, v_{Has})$$

$$\text{and } \text{erase}((M, e_{Has})) = \text{erase}((M, v_{Has})) \text{ also by Erasure-Progress-2. By canonical forms, } v_{ptr} = i \text{ and}$$

$$v_{Has} = \text{fact}. \quad \text{erase}((M, e)) = (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \xrightarrow{(U-LOAD)} (\text{erase}(M), \langle M(i), \langle \rangle \rangle) =$$

$$(L', d'). (M, e) = (M, \text{load}(e_{ptr}, e_{Has}))$$

$$\xrightarrow{*, (\text{congruence rule}) E[e]=\text{load}(e, e_{Has})} (M, \text{load}(i, e_{Has})) \text{ and } (M, \text{load}(i, e_{Has})) \xrightarrow{*, (\text{congruence rule}) E[e]=\text{load}(v_{ptr}, e)}$$

$$(M, \text{load}(i, \text{fact})) \text{ and } (M, \text{load}(i, \text{fact})) \xrightarrow{(E-LOAD)} (M, \wedge \langle M(i), \text{fact} \rangle) = (M, e'). \quad \text{erase}((M, e'))$$

$$= \text{erase}((M, \wedge \langle M(i), \text{fact} \rangle)) = (\text{erase}(M), \langle M(i), \langle \rangle \rangle) = (L, d') \text{ by (ER-M), (ER-M2), (ER-i), and (ER-FACT).}$$

- Let  $\text{erase}(e_{ptr})$  be a value and  $\text{erase}(e_{Has})$  be a non value.  
 $(M, e_{ptr}) \xrightarrow{*} (M, v_{ptr})$  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  
 $(M, e_{Has}) \xrightarrow{\dagger} (M'_2, e'_{Has})$ ,  $\text{erase}((M'_2, e'_{Has})) = (L'_2, d_{Has})$  by induction.  
 $\text{erase}((M, e)) = (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \xrightarrow{(\text{congruence rule})E[e]=\text{load}(v_{ptr}, e)} (L'_2, \text{load}(\text{erase}(e_{ptr}), d_{Has})) = (L'_2, \text{load}(i, d_{Has})) = (L', d')$ .  $(L'_2 = L')$ .  $(M, e) = (M, \text{load}(e_{ptr}, e_{Has}))$   
 $\xrightarrow{*, (\text{congruence rule})E[e]=\text{load}(e, e_{Has})} (M, \text{load}(i, e_{Has}))$  and  $(M, \text{load}(i, e_{Has})) \xrightarrow{+, (\text{congruence rule})E[e]=\text{load}(v_{ptr}, e)} (M'_2, \text{load}(i, e'_{Has})) = (M', e')$ .  $(M'_2 = M')$ .  
 $\text{erase}((M', e')) = \text{erase}((M'_2, \text{load}(i, e'_{Has}))) = (\text{erase}(M'_2), \text{load}(i, \text{erase}(e'_{Has})))$  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  $(M, e_{ptr}) \xrightarrow{\dagger} (M', e'_{ptr})$ ,  $\text{erase}((M', e'_{ptr})) = (L', d_{ptr})$  by induction.  $e'_{ptr} = i$  so  $\text{erase}(e'_{ptr}) = \text{erase}(i) = i = d_{ptr}$ .  $\text{erase}((M, e)) = (\text{erase}(M), \text{load}(\text{erase}(e_{ptr}), \text{erase}(e_{Has}))) \xrightarrow{(\text{congruence rule})E[e]=\text{load}(e, e_{Has})} (L', \text{load}(d_{ptr}, \text{erase}(e_{Has}))) = (L', d')$ .  
 $(M, e) = (M, \text{load}(e_{ptr}, e_{Has})) \xrightarrow{+, (\text{congruence rule})E[e]=\text{load}(e, e_{Has})} (M', \text{load}(i, e_{Has})) = (M', e')$ .  $\text{erase}((M', e')) = \text{erase}((M', \text{load}(i, e_{Has}))) = (\text{erase}(M'), \text{load}(i, \text{erase}(e_{Has})))$  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 $e = \text{coerce}(e) \mid \text{make\_eq}(\tau) \mid \text{new\_fun}(K)$ .

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

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.  
 $(M, e_1) \xrightarrow{*} (M, v_1)$  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.  $\text{erase}((M, e_1)) = \text{erase}((M, \langle \rangle))$ .  $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \text{erase}(e_1) \text{ in } \langle \rangle) \xrightarrow{(E-LET)} (\text{erase}(M), \langle \rangle) = (L', d')$ .  $(M, e) = (M, \text{discard\_fun}(e_1)) \xrightarrow{*, (\text{congruence rule})E[e]=\text{discard\_fun}(e)} (M, \text{discard\_fun}(v_1)) \xrightarrow{(E-DISCARDFUN)} (M, \langle \rangle) = (M, e')$ .  $\text{erase}((M, e')) = (\text{erase}(M), \text{erase}(\langle \rangle)) = (\text{erase}(M), \langle \rangle) = (L', d')$  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  $(M, e_1) \xrightarrow{\dagger} (M', e'_1)$ ,  $\text{erase}((M', e'_1)) = (L', d_1)$  by induction.  $(M, e) = (M, \text{discard\_fun}(e_1)) \xrightarrow{+, (\text{congruence rule})E[e]=\text{discard\_fun}(e)} (M', \text{discard\_fun}(e'_1)) = (M', e')$ .  
 $\text{erase}((M, e)) = (\text{erase}(M), \text{let } x = \text{erase}(e_1) \text{ in } \langle \rangle) \xrightarrow{(\text{congruence rule})E[e]=\text{let } x=e \text{ in } e_2} (L', \text{let } x = d_1 \text{ in } \langle \rangle) = (L', d')$ .  
 $\text{erase}((M', e')) = (\text{erase}(M'), \text{erase}(\text{discard\_fun}(e'_1))) = (\text{erase}(M'), \text{let } x = \text{erase}(e'_1) \text{ in } \langle \rangle)$  by (ER-M) and (ER-DISCARDFUN). This equals  $(L', \text{let } x = d_1 \text{ in } \langle \rangle) = (L', d')$ .



## 9 Converting $\lambda^C$ to $\lambda^{low}$

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

$$\tau = \alpha \mid \text{int} \mid \forall[\overrightarrow{\alpha}].(\overrightarrow{\tau}) \rightarrow \text{void} \mid \langle \overrightarrow{\tau} \rangle \mid \exists \alpha. \tau$$

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

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

$$p = + \mid - \mid *$$

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

### 9.1 Converting $\lambda^C$ to $\lambda^{C_2}$

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

$$K = \text{ptr} \mid \text{primitive}$$

$$\tau = \alpha \mid \text{int} \mid \forall[\overrightarrow{\alpha} : \overrightarrow{K}].(\overrightarrow{\tau}) \rightarrow \text{void} \mid \langle \overrightarrow{\tau} \rangle \mid \exists \alpha : K. \tau$$

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

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

$$p = + \mid - \mid *$$

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

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

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

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

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

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

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

$$\text{(TD-APP)} \quad \frac{\Delta; \Gamma \vdash v : (\tau_1, \dots, \tau_n) \rightarrow \text{void} \quad \Delta; \Gamma \vdash v_i : \tau_i}{\Delta; \Gamma \vdash v(\overrightarrow{v})}$$

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

$$\begin{array}{l}
\text{(TD-HALT)} \quad \frac{\Delta; \Gamma \vdash v : \tau}{\Delta; \Gamma \vdash \text{halt}[\tau]v} \\
\text{(TD-SUB)} \quad \frac{\Delta; \Gamma \vdash v : \tau \quad \Delta; \Gamma, x : \tau \vdash e}{\Delta; \Gamma \vdash \text{let } x = v \text{ in } e} \quad (x \notin \Gamma) \\
\text{(TD-PROJECT)} \quad \frac{\Delta; \Gamma \vdash v : \langle \tau_1, \dots, \tau_n \rangle}{\Delta; \Gamma \vdash \pi_i v : \tau_i} \quad (1 \leq i \leq n) \\
\text{(TD-UNPACK)} \quad \frac{\Delta; \Gamma \vdash v : \exists \alpha : K. \tau \quad (\Delta, \alpha : K); (\Gamma, x : \tau) \vdash e}{\Delta; \Gamma \vdash \text{let } [\alpha, x] = \text{unpack } v \text{ in } e} \quad (x \notin \Gamma \wedge \alpha \notin \Delta) \\
\text{(TD-OP)} \quad \frac{\Delta; \Gamma \vdash v_1 : \text{int} \quad \Delta; \Gamma \vdash v_2 : \text{int} \quad \Delta; \Gamma, x : \text{int} \vdash e}{\Delta; \Gamma \vdash \text{let } x = v_1 p v_2 \text{ in } e} \quad (x \notin \Gamma) \\
\text{(TD-VAR)} \quad \Delta; \Gamma, x : \tau \vdash x : \tau \\
\text{(TD-INT)} \quad \Delta; \Gamma \vdash i : \text{int} \\
\text{(TD-TAPP)} \quad \frac{\Delta \vdash \tau_0 : K \quad \Delta; \Gamma \vdash v : \forall [\alpha : K, \overrightarrow{\beta : K'}]. (\overrightarrow{\tau}) \rightarrow \text{void}}{\Delta; \Gamma \vdash v[\tau_0] : (\forall [\beta : K']. (\overrightarrow{\tau}[\tau_0/\alpha]) \rightarrow \text{void})} \\
\text{(TD-TUPLE)} \quad \frac{\Delta; \Gamma \vdash v_i : \tau_i}{\Delta; \Gamma \vdash \langle \overrightarrow{v} \rangle : \langle \overrightarrow{\tau} \rangle} \quad (x_0 \notin \Gamma) \\
\text{(TD-PACK)} \quad \frac{\Delta \vdash \tau : K \quad \Delta; \Gamma \vdash v : \tau'[\tau/\alpha]}{\Delta; \Gamma \vdash \text{pack}[\tau, v] \text{ as } \exists \alpha : K. \tau' : \exists \alpha : K. \tau'} \\
\text{(TD-FIX)} \quad \frac{\overrightarrow{\alpha : K} \vdash \tau_i : K'_i \quad \overrightarrow{\alpha : K}; x : \forall [\overrightarrow{\alpha : K}]. (\overrightarrow{\tau}) \rightarrow \text{void}, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e}{\Delta; \Gamma \vdash \text{fixcode } x[\overrightarrow{\alpha : K}](\overrightarrow{x : \overrightarrow{\tau}}).e : \forall [\overrightarrow{\alpha : K}]. (\overrightarrow{\tau}) \rightarrow \text{void}} \\
\text{(TD-ENV)} \quad \frac{\Delta \vdash \tau_i : K_i}{\Delta \vdash \{x_1 \mapsto \tau_1, \dots, x_n \mapsto \tau_n\}}
\end{array}$$

The translation from  $\lambda^C$  to  $\lambda^{C^2}$  boxes all non-pointer types:

$$\mathcal{B}(\{\alpha_1, \dots, \alpha_n\}) = \{\alpha_1 \mapsto \text{ptr}, \dots, \alpha_n \mapsto \text{ptr}\}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**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.  $\tau = \alpha$

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

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

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

1b.  $\tau = \beta \neq \alpha$

$$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\beta) = \beta$$

$$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\beta)[\mathcal{B}(\tau')/\alpha] = \beta[\mathcal{B}(\tau')/\alpha] = \beta$$

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

2.  $\tau = \text{int}$

$$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\text{int}[\tau'/\alpha]) = \mathcal{B}(\text{int}) = \langle \text{int} \rangle$$

$$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\text{int})[\mathcal{B}(\tau')/\alpha] = \langle \text{int} \rangle[\mathcal{B}(\tau')/\alpha] = \langle \text{int} \rangle$$

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

3.  $\tau = \forall[\overrightarrow{\beta}].(\overrightarrow{\tau}) \rightarrow \text{void}$

$$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\forall[\overrightarrow{\beta}].(\overrightarrow{\tau}) \rightarrow \text{void})[\tau'/\alpha] = \mathcal{B}(\forall[\overrightarrow{\beta}].(\overrightarrow{\tau[\tau'/\alpha]}) \rightarrow \text{void})$$

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

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

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

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

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

4.  $\tau = \langle \overrightarrow{\tau} \rangle$

$$\mathcal{B}(\tau[\tau'/\alpha]) = \mathcal{B}(\langle \overrightarrow{\tau} \rangle[\tau'/\alpha]) = \mathcal{B}(\langle \overrightarrow{\tau[\tau'/\alpha]} \rangle) = \langle \overrightarrow{\mathcal{B}(\tau[\tau'/\alpha])} \rangle$$

$$\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha] = \mathcal{B}(\langle \overrightarrow{\tau} \rangle)[\mathcal{B}(\tau')/\alpha] = \langle \overrightarrow{\mathcal{B}(\tau)} \rangle[\mathcal{B}(\tau')/\alpha] = \langle \overrightarrow{\mathcal{B}(\tau)[\mathcal{B}(\tau')/\alpha]} \rangle$$

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

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

1.  $\tau = \alpha$ ,  $\mathcal{B}(\tau) = \alpha$

Because  $\Delta \vdash_C \tau$  ( $FTV(\tau) \subseteq \Delta$ ),

$\mathcal{B}(\Delta) = \Delta', \alpha : \text{ptr}$

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.  $\tau = \forall \vec{\alpha} . \vec{\tau} \rightarrow \text{void}$ ,  $B(\tau) = \langle \forall \vec{\alpha} : \text{ptr} . \vec{B}(\vec{\tau}) \rightarrow \text{void} \rangle$

$\Delta \vdash_C \forall \vec{\alpha} . \vec{\tau} \rightarrow \text{void}$  where  $\Delta, \vec{\alpha} \vdash_C \tau_i$ .

By induction,  $\mathcal{B}(\Delta, \vec{\alpha}) \vdash_C \mathcal{B}(\tau_i) : \text{ptr}$ .

By (KD-ALL),  $\mathcal{B}(\Delta) \vdash_C \langle \forall \vec{\alpha} : \text{ptr} . \vec{B}(\vec{\tau}) \rightarrow \text{void} \rangle : \text{primitive}$

By the (KD-TUPLE) rule, we know  $\mathcal{B}(\Delta) \vdash_{C_2} \langle \forall \vec{\alpha} : \text{ptr} . \vec{B}(\vec{\tau}) \rightarrow \text{void} \rangle : \text{ptr}$

Thus,  $\mathcal{B}(\Delta) \vdash_{C_2} B(\tau) : \text{ptr}$

4.  $\tau_0 = \langle \vec{\tau} \rangle$ ,  $B(\tau_0) = \langle \vec{B}(\vec{\tau}) \rangle$

$\Delta \vdash_C \langle \vec{\tau} \rangle$ , and  $\Delta \vdash_C \tau_i$

By induction,  $\mathcal{B}(\Delta) \vdash_{C_2} B(\tau_i) : K_i$

From the (KD-TUPLE) rule, we know  $\mathcal{B}(\Delta) \vdash_{C_2} \langle \vec{B}(\vec{\tau}) \rangle : \text{ptr}$

Thus,  $\mathcal{B}(\Delta) \vdash_{C_2} B(\tau_0) : \text{ptr}$

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.  $e = v(\vec{v})$ ,  $\mathcal{B}(e) = (\pi_1(\mathcal{B}(v)))\overrightarrow{(\mathcal{B}(v))}$   
 By the (TC-APP),  $\Delta; \Gamma \vdash_C v : \langle \tau_1, \dots, \tau_n \rangle \rightarrow \text{void}$  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.  $e = \text{halt}[\tau]v$ ,  $\mathcal{B}(e) = \text{halt}[\mathcal{B}(\tau)]\mathcal{B}(v)$   
 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),  $\Delta; \Gamma \vdash_C v : \langle \tau_1, \dots, \tau_n \rangle$ , 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.  $v = \langle \vec{v} \rangle$ ,  $\mathcal{B}(v) = \langle \overrightarrow{\mathcal{B}(v)} \rangle$

By (TC-TUPLE),  $\Delta; \Gamma \vdash_C \langle \vec{v} \rangle : \langle \vec{\tau} \rangle$ , 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),  $\Delta; \Gamma \vdash_C v : \forall[\alpha, \vec{\beta}]. \vec{\tau}' \rightarrow \text{void}$ , and  $\Delta \vdash_C \tau$

By induction,  $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \langle \forall[\alpha : \text{ptr}, \vec{\beta} : \text{ptr}]. \overrightarrow{\mathcal{B}(\tau')} \rightarrow \text{void} \rangle$ ,  
and  $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr}$

By (TD-PROJECT),  $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} (\pi_1 \mathcal{B}(v)) : \forall[\alpha : \text{ptr}, \vec{\beta} : \text{ptr}]. \overrightarrow{\mathcal{B}(\tau')} \rightarrow \text{void}$

By (TD-TAPP), we obtain

$\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \pi_1(\mathcal{B}(v))[\mathcal{B}(\tau)] : \overrightarrow{\forall \beta : \text{ptr}. \mathcal{B}(\tau')}[\mathcal{B}(\tau)/\alpha] \rightarrow \text{void}$

By the Lemma above,  $\overrightarrow{\forall \beta : \text{ptr}. \mathcal{B}(\tau')}[\mathcal{B}(\tau)/\alpha] \rightarrow \text{void} = \overrightarrow{\forall \beta : \text{ptr}. \mathcal{B}([\tau/\alpha]\tau')} \rightarrow \text{void}$

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

Thus by (TD-TUPLE),  $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle (\pi_1(\mathcal{B}(v)))[\mathcal{B}(\tau)] \rangle : \mathcal{B}(\overrightarrow{\forall \beta}. [\tau/\alpha]\tau' \rightarrow \text{void})$

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

From (TC-FIX),  $\vec{\alpha} \vdash_C \tau_i$ ,  $\vec{\alpha}; x : \forall[\vec{\alpha}]. (\vec{\tau}) \rightarrow \text{void}$ ,  $x_1 : \tau_1, \dots, x_n : \tau_n \vdash_C e$

By induction,  $\vec{\alpha} : \text{ptr} \vdash_{C_2} \mathcal{B}(\tau_i) : \text{ptr}$ ,

$\vec{\alpha} : \text{ptr}; x : \langle \forall[\vec{\alpha} : \text{ptr}]. (\mathcal{B}(\vec{\tau})) \rightarrow \text{void} \rangle$ ,  $x_1 : \mathcal{B}(\tau_1), \dots, x_n : \mathcal{B}(\tau_n) \vdash_{C_2} \mathcal{B}(e)$

we know  $x_0 : \forall[\vec{\alpha} : \text{ptr}]. (\mathcal{B}(\vec{\tau})) \rightarrow \text{void} \vdash_{C_2} \langle x_0 \rangle : \langle \forall[\vec{\alpha} : \text{ptr}]. (\mathcal{B}(\vec{\tau})) \rightarrow \text{void} \rangle$

By weakening lemma,  $\vec{\alpha} : \text{ptr}; x : \langle \forall[\vec{\alpha} : \text{ptr}]. (\mathcal{B}(\vec{\tau})) \rightarrow \text{void} \rangle$ ,  $x_1 : \mathcal{B}(\tau_1), \dots, x_n : \mathcal{B}(\tau_n)$ ,  $x_0 : \forall[\vec{\alpha} : \text{ptr}]. (\vec{\tau}) \rightarrow \text{void} \vdash_{C_2} \mathcal{B}(e)$

By (TD-SUB),  $\vec{\alpha} : \text{ptr}; x_1 : \mathcal{B}(\tau_1), \dots, x_n : \mathcal{B}(\tau_n)$ ,  $x_0 : \forall[\vec{\alpha} : \text{ptr}]. (\vec{\tau}) \rightarrow \text{void} \vdash_{C_2} \text{let } x = \langle x_0 \rangle \text{ in } \mathcal{B}(e)$

By (TD-FIX),  $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \text{fixcode } x_0[\vec{\alpha} : \text{ptr}](x : \mathcal{B}(\vec{\tau})).\text{let } x = \langle x_0 \rangle \text{ in } \mathcal{B}(e) : \forall[\vec{\alpha} : \text{ptr}]. (\vec{\tau}) \rightarrow \text{void}$

By (TD-TUPLE),  $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \langle \text{fixcode } x_0[\vec{\alpha} : \text{ptr}](x : \mathcal{B}(\vec{\tau})).\text{let } x = \langle x_0 \rangle \text{ in } \mathcal{B}(e) \rangle : \langle \forall[\vec{\alpha} : \text{ptr}]. (\vec{\tau}) \rightarrow \text{void} \rangle$

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

From (TC-PACK),  $\Delta \vdash_C \tau$ ,  $\Delta; \Gamma \vdash_C v : \tau'[\tau/\alpha]$

By induction, we get  $\mathcal{B}(\Delta) \vdash_{C_2} \mathcal{B}(\tau) : \text{ptr}$ ,  $\mathcal{B}(\Delta); \mathcal{B}(\Gamma) \vdash_{C_2} \mathcal{B}(v) : \mathcal{B}(\tau'[\tau/\alpha])$

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 $\lambda^{C_2}$ to $\lambda^{flat}$

Translating  $\lambda^{C_2}$  to  $\lambda^{low}$  requires some effort to break up nested tuple allocations, so we define a “flattened” variant of  $\lambda^{C_2}$  called  $\lambda^{flat}$  to factor out this effort ahead of time.  $\lambda^{flat}$  is strictly a subset of  $\lambda^{C_2}$ : they share the same kind

and type systems, and any  $\lambda^{flat}$  expression is also a  $\lambda^{C_2}$  expression:

$K = \text{primitive} \mid \text{ptr}$

$\tau = \alpha \mid \text{int} \mid \forall[\overrightarrow{\alpha} : \overrightarrow{K}].(\overrightarrow{\tau}) \rightarrow \text{void} \mid \langle \overrightarrow{\tau} \rangle \mid \exists\alpha : K.\tau$

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

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

$p = + \mid - \mid *$

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

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

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

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

(KF-TUPLE) 
$$\frac{\Delta \vdash \tau_i : K_i}{\Delta \vdash \langle \overrightarrow{\tau} \rangle : \text{ptr}}$$

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

(TF-APP) 
$$\frac{\Delta; \Gamma \vdash x : (\tau_1, \dots, \tau_n) \rightarrow \text{void} \quad \Delta; \Gamma \vdash x_i : \tau_i}{\Delta; \Gamma \vdash x(\overrightarrow{x})}$$

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

(TF-HALT) 
$$\frac{\Delta; \Gamma \vdash x : \tau}{\Delta; \Gamma \vdash \text{halt}[\tau]x}$$

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

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

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

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

(TF-OP) 
$$\frac{\Delta; \Gamma \vdash x_1 : \text{int} \quad \Delta; \Gamma \vdash x_2 : \text{int}}{\Delta; \Gamma \vdash x_1 p x_2 : \text{int}}$$

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

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

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

$$\begin{array}{l}
\text{(TF-PACK)} \quad \frac{\Delta \vdash \tau : K \quad \Delta; \Gamma \vdash x : \tau'[\tau/\alpha]}{\Delta; \Gamma \vdash \text{pack}[\tau, x] \text{ as } \exists \alpha : K. \tau' : \exists \alpha : K. \tau'} \\
\text{(TF-FIX)} \quad \frac{\overrightarrow{\alpha} : \overrightarrow{K} \vdash \tau_i : K'_i \quad \overrightarrow{\alpha} : \overrightarrow{K}; x : \forall[\overrightarrow{\alpha} : \overrightarrow{K}].(\overrightarrow{\tau}) \rightarrow \text{void}, x_1 : \tau_1, \dots, x_n : \tau_n \vdash e}{\Delta; \Gamma \vdash \text{fixcode } x[\overrightarrow{\alpha} : \overrightarrow{K}](\overrightarrow{x} : \overrightarrow{\tau}).e : \forall[\overrightarrow{\alpha} : \overrightarrow{K}].(\overrightarrow{\tau}) \rightarrow \text{void}} \\
\text{(TF-ENV)} \quad \frac{\Delta \vdash \tau_i : K_i}{\Delta \vdash \{x_1 \mapsto \tau_1, \dots, x_n \mapsto \tau_n\}}
\end{array}$$

To aid the conversion, define a declaration  $d$  as:

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

Say that (let  $d_1, \dots, d_n$  in  $e$ ) is short for (let  $d_1$  in ... let  $d_n$  in  $e$ ).

Also define abbreviations for typing declarations and typing terms prefixed by sequences of declarations:

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

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

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

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

The conversion from  $\lambda^{C_2}$  to  $\lambda^{flat}$  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.

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

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

$$\begin{aligned}
\mathcal{F}(v[\tau]) &= \overrightarrow{d}, x = t, x[\tau] \\
\text{where } \mathcal{F}(v) &= \overrightarrow{d}, t
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}(\langle \overrightarrow{v} \rangle) &= \overrightarrow{d}, \overrightarrow{x} = \overrightarrow{t}, x_0 = \langle \overrightarrow{x} \rangle, x_0 \\
\text{where } \mathcal{F}(v_i) &= \overrightarrow{d}_i, t_i
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}(\pi_i v) &= \overrightarrow{d}, x_1 = t, x_2 = \pi_i x_1, x_2 \\
\text{where } \mathcal{F}(v) &= \overrightarrow{d}, t
\end{aligned}$$

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

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

$$\begin{aligned}
\mathcal{F}(v(\overrightarrow{v})) &= \text{let } \overrightarrow{d}, \overrightarrow{d}_1, \dots, \overrightarrow{d}_n, x = t, \overrightarrow{x} = \overrightarrow{t} \text{ in } x(\overrightarrow{x}) \\
\text{where } \mathcal{F}(v_i) &= \overrightarrow{d}_i, t_i, \mathcal{F}(v) = \overrightarrow{d}, t
\end{aligned}$$

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



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

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

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

$\mathcal{F}(\text{let } x_0 = v_1 p v_2 \text{ in } e) = \text{let } \vec{d}_1, \vec{d}_2, x_1 = t_1, x_2 = t_2 \text{ in } \text{let } x_0 = x_1 p x_2 \text{ in } \mathcal{F}(e)$   
 where  $\mathcal{F}(v_1) = \vec{d}_1, t_1, \mathcal{F}(v_2) = \vec{d}_2, t_2$

$\mathcal{F}(\text{let } [\alpha, x] = \text{unpack } v \text{ in } e) = \text{let } \vec{d}, x_0 = t \text{ in } \text{let } [\alpha, x] = \text{unpack } x_0 \text{ in } \mathcal{F}(e)$   
 where  $\mathcal{F}(v) = \vec{d}, t$

### 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.  $\tau$  and  $\Gamma$

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.  $v_0 = v[\tau_0], \mathcal{F}(v[\tau_0]) = \vec{d}, x = t, x[\tau_0]$ , where  $\mathcal{F}(v) = \vec{d}, t$

From (TD-TAPP), we know  $\Delta \vdash_{C_2} \tau_0 : K$ ,

and  $\Delta; \Gamma \vdash_{C_2} v : \forall[\alpha : K, \beta : K'] . (\vec{\tau}) \rightarrow \text{void}$

By induction,  $\Delta \vdash_F \tau_0 : K$ , and  $\Delta; \Gamma \vdash_F \vec{d}, t : \forall[\alpha : K, \beta : K'] . (\vec{\tau}) \rightarrow \text{void}$

By (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, x = t, x : \forall[\alpha : K, \beta : K'] . (\vec{\tau}) \rightarrow \text{void}$

By (TF-TAPP),  $\Delta; \Gamma \vdash_F \vec{d}, x = t, x[\tau_0] : (\forall[\beta : K'] . (\vec{\tau}[\tau_0/\alpha]) \rightarrow \text{void})$

5.  $v = \langle \vec{v} \rangle, \mathcal{F}(\langle \vec{v} \rangle) = \vec{d}, \overrightarrow{x = t}, x_0 = \langle \vec{x} \rangle, x_0; \mathcal{F}(v_i) = \vec{d}_i, t_i$

From (TD-TAPP), we know  $\Delta; \Gamma \vdash_{C_2} v_i : \tau_i$ .

By induction,  $\Delta; \Gamma \vdash_F \vec{d}_i, t_i : \tau_i$ .

By weakening,  $\Delta; \Gamma \vdash_F \vec{d}, t_i : \tau_i$ .

By weakening and (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, \overrightarrow{x = t}, x_i : \tau_i$ .

By (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, \overrightarrow{x = t}, x_0 = \langle \vec{x} \rangle, x_0 : \langle \vec{\tau} \rangle$

6.  $v = \text{pack}[\tau, v] \text{ as } \exists \alpha : K . \tau'$ ,

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

From (TD-PACK), we know  $\Delta \vdash_{C_2} \tau : K$ ,  $\Delta; \Gamma \vdash_{C_2} v : \tau'[\tau/\alpha]$ .

By induction,  $\Delta \vdash_F \tau : K$ ,  $\Delta; \Gamma \vdash_F \vec{d}, t : \tau'[\tau/\alpha]$ .

By (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, x = t, x : \tau'[\tau/\alpha]$ .

By (TF-PACK),  $\Delta; \Gamma \vdash_F \vec{d}, x = t, \text{pack}[\tau, x] \text{ as } \exists \alpha : K. \tau' : \exists \alpha : K. \tau'$

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

From (TD-FIX),  $\overrightarrow{\alpha : K} \vdash_{C_2} \tau_i : K'_i$ ,

and  $\overrightarrow{\alpha : K}; x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void}, x_1 : \tau_1, \dots, x_n : \tau_n \vdash_{C_2} e$

By induction, we get  $\overrightarrow{\alpha : K} \vdash_F \tau_i : K'_i$ ,

and  $\overrightarrow{\alpha : K}; x : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void}, x_1 : \tau_1, \dots, x_n : \tau_n \vdash_F \mathcal{F}(e)$

By (TF-FIX) rule,  $\Delta; \Gamma \vdash_F \text{fixcode } x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).\mathcal{F}(e) : \forall[\overrightarrow{\alpha : K}].(\overrightarrow{\tau}) \rightarrow \text{void}$

8.  $e = v(\vec{v})$ ,  $\mathcal{F}(e) = \text{let } \vec{d}, \vec{d}_1, \dots, \vec{d}_n, x = t, \vec{x} = \vec{t} \text{ in } x(\vec{x})$ ;  $\mathcal{F}(v_i) = \vec{d}_i, t_i$ ,  $\mathcal{F}(v) = \vec{d}, t$

From (TD-APP), we can get  $\Delta; \Gamma \vdash_{C_2} v : (\tau_1, \dots, \tau_n) \rightarrow \text{void}$   $\Delta; \Gamma \vdash_{C_2} v_i : \tau_i$ .

By induction,  $\Delta; \Gamma \vdash_F \vec{d}, t : (\tau_1, \dots, \tau_n) \rightarrow \text{void}$   $\Delta; \Gamma \vdash_F \vec{d}_i, t_i : \tau_i$ .

By weakening,  $\Delta; \Gamma \vdash_F \vec{d}, \vec{d}_1, \dots, \vec{d}_n, t : (\tau_1, \dots, \tau_n) \rightarrow \text{void}$   $\Delta; \Gamma \vdash_F \vec{d}, \vec{d}_1, \dots, \vec{d}_n, t_i : \tau_i$ .

By weakening and (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, \vec{d}_1, \dots, \vec{d}_n, x = t, \vec{x} = \vec{t}, x : (\tau_1, \dots, \tau_n) \rightarrow \text{void}$

and  $\Delta; \Gamma \vdash_F \vec{d}, \vec{d}_1, \dots, \vec{d}_n, x = t, \vec{x} = \vec{t}, x : \tau_i$ .

By (TF-LET/TF-PROJECT/TF-TUPLE) and (TF-APP),  $\Delta; \Gamma \vdash_F \text{let } \vec{d}, \vec{d}_1, \dots, \vec{d}_n, x = t, \vec{x} = \vec{t} \text{ in } x(\vec{x})$ .

9.  $e = \text{if0}(v, e_1, e_2)$ ,  $\mathcal{F}(e) = \text{let } \vec{d}, x = t \text{ in if0}(x, \mathcal{F}(e_1), \mathcal{F}(e_2))$ ;  $\mathcal{F}(v) = \vec{d}, t$

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,  $\Delta; \Gamma \vdash_F \vec{d}, t : \text{int}$ ,  $\Delta; \Gamma \vdash_F \mathcal{F}(e_1)$ ,  $\Delta; \Gamma \vdash_F \mathcal{F}(e_2)$ .

By (TF-VAR) and weakening,  $\Delta; \Gamma \vdash_F \vec{d}, x = t, x : \text{int}$ .

By weakening, (TF-LET/TF-PROJECT/TF-TUPLE), and (TF-IF),  $\Delta; \Gamma \vdash_F \text{let } \vec{d}, x = t \text{ in if0}(x, \mathcal{F}(e_1), \mathcal{F}(e_2))$

10.  $e = \text{halt}[\tau]v$ ,  $\mathcal{F}(\text{halt}[\tau]v) = \text{let } \vec{d}, x = t \text{ in halt}[\tau]x$ ;  $\mathcal{F}(v) = \vec{d}, t$

From (TD-HALT), we know  $\Delta; \Gamma \vdash_{C_2} v : \tau$ .

By induction,  $\Delta; \Gamma \vdash_F \vec{d}, t : \tau$ .

By (TF-VAR) and weakening,  $\Delta; \Gamma \vdash_F \vec{d}, x = t, x : \tau$ .

By (TF-LET/TF-PROJECT/TF-TUPLE) and (TF-HALT),  $\Delta; \Gamma \vdash_F \text{let } \vec{d}, x = t \text{ in halt}[\tau]x$

11.  $e = \text{let } x = v \text{ in } e$ ,  $\mathcal{F}(\text{let } x = v \text{ in } e) = \text{let } \vec{d} \text{ in let } x = t \text{ in } \mathcal{F}(e)$ ;  $\mathcal{F}(v) = \vec{d}, t$

From (TD-SUB) rule,  $\Delta; \Gamma \vdash_{C_2} v : \tau$ ,  $\Delta; \Gamma, x : \tau \vdash_{C_2} e$ .

By induction,  $\Delta; \Gamma \vdash_{C_2} \vec{d}, t : \tau$ ,  $\Delta; \Gamma, x : \tau \vdash_{C_2} \mathcal{F}(e)$ .

By weakening and (TF-LET/TF-PROJECT/TF-TUPLE),  $\Delta; \Gamma \vdash \mathcal{F}(\text{let } x = v \text{ in } e)$

12.  $v_0 = \pi_i v$ ,  $\mathcal{F}(v_0) = \vec{d}, x_1 = t, x_2 = \pi_i x_1, x_2$ ;  $\mathcal{F}(v) = \vec{d}, t$

From (TD-PROJECT) rule,  $\Delta; \Gamma \vdash_{C_2} v : \langle \tau_1, \dots, \tau_n \rangle$ .

By induction,  $\Delta; \Gamma \vdash_F \vec{d}, t : \langle \tau_1, \dots, \tau_n \rangle$ .

By (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, x_1 = t, x_1 : \langle \tau_1, \dots, \tau_n \rangle$ .

By (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}, x_1 = t, x_2 = \pi_i x_1, x_2 : \tau_i$ .

13.  $e_0 = \text{let } x_0 = v_1 p v_2 \text{ in } e$ ,  $\mathcal{F}(e_0) = \text{let } \vec{d}_1, \vec{d}_2, x_1 = t_1, x_2 = t_2 \text{ in let } x_0 = x_1 p x_2 \text{ in } \mathcal{F}(e)$ ,  $\mathcal{F}(v_1) = \vec{d}_1, t_1$ ,  $\mathcal{F}(v_2) = \vec{d}_2, t_2$

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,  $\Delta; \Gamma \vdash_F \vec{d}_1, t_1 : \text{int}$   $\Delta; \Gamma \vdash_F \vec{d}_2, t_2 : \text{int}$

$\Delta; \Gamma, x_0 : \text{int} \vdash_F \mathcal{F}(e)$ .

By weakening,  $\Delta; \Gamma \vdash_F \vec{d}_1, \vec{d}_2, t_1 : \text{int} \quad \Delta; \Gamma \vdash_F \vec{d}_1, \vec{d}_2, t_2 : \text{int}$  .

By weakening and (TF-VAR),  $\Delta; \Gamma \vdash_F \vec{d}_1, \vec{d}_2, x_1 = t_1, x_2 = t_2, x_1 : \text{int} \quad \Delta; \Gamma \vdash_F \vec{d}_1, \vec{d}_2, x_1 = t_1, x_2 = t_2, x_2 : \text{int}$  .

By (TF-LET/TF-PROJECT/TF-TUPLE), weakening, and (TF-OP),  $\Delta; \Gamma \vdash_F \mathcal{F}(e_0)$ .

14.  $e_0 = \text{let } [\alpha, x] = \text{unpack } v \text{ in } e, \mathcal{F}(e_0) = \text{let } \vec{d}, x_0 = t \text{ in let } [\alpha, x] = \text{unpack } x_0 \text{ in } \mathcal{F}(e); \mathcal{F}(v) = \vec{d}, t$

From (TD-UNPACK),  $\Delta; \Gamma \vdash v : \exists \alpha : K. \tau, (\Delta, \alpha : K); (\Gamma, x : \tau) \vdash e$

By induction,  $\Delta; \Gamma \vdash \vec{d}, t : \exists \alpha : K. \tau, (\Delta, \alpha : K); (\Gamma, x : \tau) \vdash \mathcal{F}(e)$

By weakening,  $\Delta; \Gamma \vdash \vec{d}, x_0 = t, x_0 : \exists \alpha : K. \tau$  .

By (TF-LET/TF-PROJECT/TF-TUPLE), weakening, and (TF-UNPACK),  $\Delta; \Gamma \vdash \mathcal{F}(e_0)$

### 9.3 Converting $\lambda^{flat}$ to $\lambda^{low}$

#### Kind conversion

$K_R = \text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow \dot{1}$

$\mathcal{A}(\text{primitive}) = \dot{1}$

$\mathcal{A}(\text{ptr}) = \dot{1}$

$\mathcal{A}_{tyarg}(\text{primitive}) = \dot{1}$

$\mathcal{A}_{tyarg}(\text{ptr}) = \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow \dot{0}$

#### 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.

$\mathcal{A}(\tau : \text{primitive}) = \lambda R : K_R. \lambda E : \text{int}. \mathcal{A}_{prim}(\tau)$

$\mathcal{A}_{tyarg}(\tau : \text{primitive}) = \mathcal{A}_{prim}(\tau)$

$\mathcal{A}(\tau : \text{ptr}) = \lambda R : K_R. \lambda E : \text{int}. \exists L : \text{int}; L \geq 0. (\text{GcPtr}(\mathcal{A}_{ptr}(\tau) L R) E)$

$\mathcal{A}_{tyarg}(\tau : \text{ptr}) = \mathcal{A}_0(\tau : \text{ptr})$

$\mathcal{A}_{ptr}(\tau) = \mu \alpha : \mathcal{A}_{tyarg}(\text{ptr}). \lambda L : \text{int}. \lambda R : K_R. \lambda E : \text{int}. \lambda I : \text{int}. \cdot \langle \mathcal{A}_0(\tau) L R E I, \text{GcCoerce } L R (\alpha L R) \rangle$

$\mathcal{A}_0(\alpha) = \alpha$

$\mathcal{A}_0(\exists \alpha : K. \tau) = \lambda L : \text{int}. \lambda R : K_R. \lambda E : \text{int}. \lambda I : \text{int}. \exists \alpha : \mathcal{A}_{tyarg}(K); L \geq 1. (\mathcal{A}_{ptr}(\tau) (L - 1) R E I)$

$\mathcal{A}_0(\langle \tau_1 : K_1, \dots, \tau_n : K_n \rangle) = \lambda L : \text{int}. \lambda R : K_R. \lambda E : \text{int}. \lambda I : \text{int}.$

$\cdot \langle \exists F : \text{int} \rightarrow \text{int} \rightarrow \dot{1} \cdot \langle \text{Eq}(F 0, \mathcal{A}(\tau_1) R), \dots, \text{Eq}(F (n - 1), \mathcal{A}(\tau_n) R), (\text{GcWord } R E I (\text{GcHdr } R F \text{Tag } n)) \rangle, \langle \text{GcWord } R E (I + 1) (\mathcal{A}(\tau_1) R), \dots, \text{GcWord } R E (I + n) ((\mathcal{A}(\tau_n) R)) \rangle \rangle \rangle$

where  $\text{Tag} = (\sum_{i=1}^n 2^{i-1} * \text{tag}(K_i))$ , where  $\text{tag}(\text{prim}) = 0$  and  $\text{tag}(\text{ptr}) = 1$

$\mathcal{A}_{prim}(\exists \alpha : K. \tau) = \exists \alpha : \mathcal{A}_{tyarg}(K). \mathcal{A}_{prim}(\tau)$

$\mathcal{A}_{prim}(\alpha) = \alpha$

$\mathcal{A}_{prim}(\text{int}) = \tau_{int}$

$\mathcal{A}_{prim}(\forall [\langle \alpha : K \rangle]. \langle \vec{\tau} \rangle \rightarrow \text{void}) = \forall \langle \alpha : \mathcal{A}_{tyarg}(K) \rangle. \forall R : K_R. \forall E : \text{int}. \wedge \langle \tau_g(R, E), \langle \mathcal{A}(\tau) R E \rangle \rangle \rightarrow \tau_{halt}$

$\tau_{int} = \exists I : \text{int}. \text{Int}(I)$

### Environment conversion:

$$\mathcal{A}_{tyarg}(\{\alpha_1 \mapsto K_1, \dots, \alpha_n \mapsto K_n\}) = \{\alpha_1 \mapsto \mathcal{A}_{tyarg}(K_1), \dots, \alpha_n \mapsto \mathcal{A}_{tyarg}(K_n)\}$$

$$\mathcal{A}(\{x_1 \mapsto \tau_1, \dots, x_n \mapsto \tau_n\}) RE = \{x_1 \mapsto (\mathcal{A}(\tau_1) RE), \dots, x_n \mapsto (\mathcal{A}(\tau_n) RE)\}$$

### Expression conversion:

$$\mathcal{A}(x) RE g = x$$

$$\mathcal{A}(i) RE g = \text{pack}(i, i) \text{ as } \tau_{int}$$

$$\mathcal{A}(x_1 p x_2) RE g = OP_p \cdot \langle x_1, x_2 \rangle,$$

$$\mathcal{A}(x[\tau]) RE g = x[\mathcal{A}_{tyarg}(\tau)]$$

$$\mathcal{A}(\text{pack}[\tau, x] \text{ as } \exists\beta : K.\tau' : \text{primitive}) RE g = \text{pack } \mathcal{A}_{tyarg}(\tau), x \text{ as } \exists\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}(\tau') RE$$

$$\mathcal{A}(\text{pack}[\tau, x] \text{ as } \exists\beta : K.\tau' : \text{ptr}) RE g = \text{ptr\_pack}_K RE (\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau')) \mathcal{A}_{tyarg}(\tau)x$$

$$\mathcal{A}(\text{fixcode } x[\overrightarrow{\alpha : K}](\overrightarrow{x : \tau}).e) RE g = \text{fix } x : \overrightarrow{\forall\alpha : \mathcal{A}_{tyarg}(K).\forall R' : K_R.\forall E' : \text{int}.\wedge \langle \tau_g(R', E'), \overrightarrow{\mathcal{A}(\tau) R' E'} \rangle} \rightarrow \tau_{halt}(R').$$

$$\overrightarrow{\Lambda\alpha : \mathcal{A}_{tyarg}(K).\Lambda R' : K_R.\Lambda E' : \text{int}.\lambda y : \wedge \langle \tau_g(R', E'), \overrightarrow{\mathcal{A}(\tau) R' E'} \rangle} \rightarrow \text{let } \langle g', \overrightarrow{x} \rangle = y \text{ in } \mathcal{A}(e) R' E' g'$$

$$\mathcal{A}(x(\overrightarrow{x})) RE g = x RE \wedge \langle g, \overrightarrow{x} \rangle$$

$$\mathcal{A}(\text{if0}(x, e_1, e_2)) RE g = \text{unpack } \beta, x_1 = x \text{ in if } x_1 = 0 \text{ then } \mathcal{A}(e_1) RE g \text{ else } \mathcal{A}(e_2) RE g$$

$$\mathcal{A}(\text{halt}[\tau]x) RE g = \text{halt } RE (\mathcal{A}(\tau) RE) \wedge \langle g, x \rangle$$

$$\mathcal{A}(\text{let } x = t \text{ in } e) RE g = \text{let } x = \mathcal{A}(t) RE g \text{ in } \mathcal{A}(e) RE g$$

$$\mathcal{A}(\text{let } x_0 = \pi_i(x_1 : \overrightarrow{\tau : K}) \text{ in } e) RE g = \text{let } \langle g', x_0 \rangle = \text{obj\_load}_{\overrightarrow{K}, i} RE \overrightarrow{\mathcal{A}_{tyarg}(\tau)} \wedge \langle g, x_1 \rangle \text{ in } \mathcal{A}(e) RE g'$$

$$\mathcal{A}(\text{let } [\alpha : K, x_0 : \tau_0 : \text{primitive}] = \text{unpack } x_1 : \exists\beta : K.\tau \text{ in } e) RE g = \text{unpack } \alpha, x_0 = x_1 \text{ in } \mathcal{A}(e) RE g$$

$$\mathcal{A}(\text{let } [\alpha : K, x_0 : \tau_0 : \text{ptr}] = \text{unpack } x_1 : \exists\beta : K.\tau \text{ in } e) RE g =$$

$$\text{unpack } \alpha, x_0 = \text{ptr\_unpack}_K RE (\lambda\beta : \mathcal{A}_{tyarg}(K).\mathcal{A}_{tyarg}(\tau)) x_1 \text{ in } \mathcal{A}(e) RE g$$

$$\mathcal{A}(\text{let } x_0 = \langle \overrightarrow{x} : \overrightarrow{\tau_x} \rangle \text{ in } e) RE g = \text{unpack } E', z = \text{alloc}_{\overrightarrow{K}, n} RE \overrightarrow{\mathcal{A}_{tyarg}(\tau_x)} \overrightarrow{\mathcal{A}_{tyarg}(\tau_y)} \wedge \langle g, \overrightarrow{x}, \overrightarrow{y} \rangle \text{ in}$$

$$\text{let } \langle g', x_0, \overrightarrow{y} \rangle = z \text{ in } \mathcal{A}(e) RE' g'; \quad \text{where } \{y_1, \dots, y_n\} = \{y : \tau_y : \text{ptr} \in FV(e)\} - \{x_0\}$$

### Types of $\lambda^{low}$ Run-time System Functions:

To simplify the forms of the types below, we treat type application  $(\alpha\beta)$  as if it were a  $\lambda^{flat}$  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).

$$K_W = \text{int} \rightarrow 1$$

$$K_A = \text{int} \rightarrow \text{int} \rightarrow 0$$

$$K_F = \text{int} \rightarrow \text{int} \rightarrow 1$$

$$K_R = \text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow 1$$

$$\tau_{int} = \exists I : \text{int}. \text{Int}(I)$$

$$\emptyset; R : K_R, E : \text{int} \vdash_L \tau_g(R, E) : K_g, \quad (K_g = \hat{i}_g)$$

$$\emptyset; \emptyset \vdash_L \tau_{halt} : K_{halt}, \quad (K_{halt} = \hat{i}_{halt})$$

$$\emptyset; \emptyset \vdash_L GcWord : K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow K_W \rightarrow \dot{0}$$

$$\emptyset; \emptyset \vdash_L GcPtr : K_A \rightarrow \text{int} \rightarrow \dot{1}$$

$$\emptyset; \emptyset \vdash_L GcCoerce : \text{int} \rightarrow K_R \rightarrow K_A \rightarrow \dot{0}$$

$$\emptyset; \emptyset \vdash_L GcHdr : K_R \rightarrow K_F \rightarrow \text{int} \rightarrow \text{int} \rightarrow \text{int} \rightarrow \dot{1}$$

$$\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L halt : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \dot{1} . \wedge \langle \tau_g(R, E), \alpha \rangle \rightarrow \tau_{halt}$$

$$\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L OP_p : \cdot \langle \tau_{int}, \tau_{int} \rangle \rightarrow \tau_{int}$$

$$\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L$$

$$ptr\_pack_K : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \mathcal{A}_{tyarg}(K) \rightarrow \mathcal{A}_{tyarg}(\text{ptr}). \forall \beta : \mathcal{A}_{tyarg}(K). \mathcal{A}(\alpha \beta) R E \rightarrow \mathcal{A}(\exists \gamma : K. (\alpha \gamma)) R E$$

$$\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L$$

$$ptr\_unpack_K : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \mathcal{A}_{tyarg}(K) \rightarrow \mathcal{A}_{tyarg}(\text{ptr}). \mathcal{A}(\exists \beta : K. (\alpha \beta)) R E \rightarrow \exists \beta : \mathcal{A}_{tyarg}(K). (\mathcal{A}(\alpha \beta) R E)$$

$$\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L$$

$$obj\_load_{\vec{K}, i} : \forall R : K_R. \forall E : \text{int}. \overrightarrow{\forall \alpha : \mathcal{A}_{tyarg}(K)}. \wedge \langle \tau_g(R, E), \mathcal{A}(\overrightarrow{\langle \alpha : K \rangle}) R E \rangle \rightarrow \wedge \langle \tau_g(R, E), \mathcal{A}(\alpha_i : K_i) R E \rangle$$

$$\emptyset; \emptyset; \emptyset; \emptyset; true; \infty \vdash_L$$

$$alloc_{\vec{K}, n} : \forall R : K_R. \forall E : \text{int}. \overrightarrow{\forall \alpha : \mathcal{A}_{tyarg}(K)}. \forall \beta_1 : \mathcal{A}_{tyarg}(\text{ptr}). \dots$$

$$\forall \beta_n : \mathcal{A}_{tyarg}(\text{ptr}). \wedge \langle \tau_g(R, E), \overrightarrow{\mathcal{A}(\alpha : K) R E}, \overrightarrow{\mathcal{A}(\beta : \text{ptr}) R E} \rangle \rightarrow \exists E'. \wedge \langle \tau_g(R, E'), \mathcal{A}(\overrightarrow{\langle \alpha : K \rangle}) R E', \overrightarrow{\mathcal{A}(\beta : \text{ptr}) R E'} \rangle$$

**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  $\tau$ . For each  $\tau$ , 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.

$$\begin{aligned} 1. & \mathcal{A}(\tau : \text{primitive})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] \\ &= (\lambda R : K_R. \lambda E : \text{int}. \mathcal{A}_{prim}(\tau))[\mathcal{A}_{tyarg}(\tau_0)/\alpha] \\ &= \lambda R : K_R. \lambda E : \text{int}. (\mathcal{A}_{prim}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha]) \\ &= \lambda R : K_R. \lambda E : \text{int}. \mathcal{A}_{prim}((\tau[\tau_0/\alpha]) : \text{primitive}) \text{ by the } \mathcal{A}_{prim}(\tau) \text{ case} \\ &= \mathcal{A}(\tau[\tau_0/\alpha]) \end{aligned}$$

$$\begin{aligned} 2. & \mathcal{A}(\tau : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha] \\ &= (\lambda R : K_R. \lambda E : \text{int}. \exists L : \text{int}; L \geq 0. (GcPtr(\mathcal{A}_{ptr}(\tau) L R) E))[\mathcal{A}_{tyarg}(\tau_0)/\alpha] \\ &= \lambda R : K_R. \lambda E : \text{int}. \exists L : \text{int}; L \geq 0. (GcPtr((\mathcal{A}_{ptr}(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha]) L R) E) \\ &= \lambda R : K_R. \lambda E : \text{int}. \exists L : \text{int}; L \geq 0. (GcPtr(\mathcal{A}_{ptr}((\tau[\tau_0/\alpha]) : \text{ptr}) L R) E) \text{ by the } \mathcal{A}_{ptr}(\tau) \text{ case} \\ &= \mathcal{A}(\tau[\tau_0/\alpha]) \end{aligned}$$

$$\begin{aligned} 3. & \mathcal{A}_{ptr}(\tau : \text{ptr})[\mathcal{A}_{tyarg}(\tau_0)/\alpha'] \\ &= (\mu \alpha : \mathcal{A}_{tyarg}(\text{ptr}). \lambda L : \text{int}. \lambda R : K_R. \lambda E : \text{int}. \lambda I : \text{int}. \cdot \langle \mathcal{A}_0(\tau) L R E I, GcCoerce L R (\alpha L R) \rangle)[\mathcal{A}_{tyarg}(\tau_0)/\alpha'] \\ &= \mu \alpha : \mathcal{A}_{tyarg}(\text{ptr}). \lambda L : \text{int}. \lambda R : K_R. \lambda E : \text{int}. \lambda I : \text{int}. \cdot \langle (\mathcal{A}_0(\tau)[\mathcal{A}_{tyarg}(\tau_0)/\alpha']) L R E I, GcCoerce L R (\alpha L R) \rangle \end{aligned}$$

$= \mu\alpha : \mathcal{A}_{tyarg}(\text{ptr}).\lambda L : \text{int}.\lambda R : K_R.\lambda E : \text{int}.\lambda I : \text{int}.\cdot(\mathcal{A}_0(\tau[\tau_0/\alpha]) L R E I, GcCoerce L R(\alpha L R))$  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  $\alpha$  and  $\tau_0$  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  $\alpha$  and  $\tau_0$  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  $\tau$ . For each  $\tau$ , 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 :  $\mathcal{A}(\tau : \text{primitive}) = \lambda R : \text{int} . \lambda E : \text{int} . \mathcal{A}_{\text{prim}}(\tau)$

By the  $\mathcal{A}_{\text{prim}}(\tau)$  case,  $\mathcal{A}_{\text{tyarg}}(\Delta) \vdash_L \mathcal{A}_{\text{prim}}(\tau) : \dot{1}$ .

So  $\mathcal{A}_{\text{tyarg}}(\Delta) \vdash_L \mathcal{A}(\tau : \text{ptr}) : K_R \rightarrow \text{int} \rightarrow \dot{1}$ .

case 3 :  $\mathcal{A}_{\text{ptr}}(\tau : \text{ptr}) = \mu \alpha : \mathcal{A}_{\text{tyarg}}(\text{ptr}) . \lambda L : \text{int} . \lambda R : K_R . \lambda E : \text{int} . \lambda I : \text{int} . \langle \mathcal{A}_0(\tau) L R E I, GcCoerce L R(\alpha L R) \rangle$

By induction,  $\mathcal{A}_{\text{tyarg}}(\Delta) \vdash_L \mathcal{A}_0(\tau : \text{ptr}) : \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow 0$ .

By the kinding rules,  $\mathcal{A}_{\text{tyarg}}(\Delta) \vdash_L \mathcal{A}_{\text{ptr}}(\tau) : \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow 0$ .

cases 4,5,6: We show several cases for  $\mathcal{A}_0(\tau)$  and  $\mathcal{A}_{\text{prim}}(\tau)$ . The proofs for the other cases are similar.

case 4 :

$\mathcal{A}_{\text{prim}}((\exists \alpha : K . \tau) : \text{primitive}) = \exists \alpha : \mathcal{A}_{\text{tyarg}}(K) . \mathcal{A}_{\text{prim}}(\tau)$

By induction,  $\mathcal{A}_{\text{tyarg}}(\Delta, \alpha : K) \vdash_L \mathcal{A}_{\text{prim}}(\tau : \text{primitive}) : \dot{1}$ .

So  $\mathcal{A}_{\text{tyarg}}(\Delta), \alpha : \mathcal{A}_{\text{tyarg}}(K) \vdash_L \mathcal{A}_{\text{prim}}(\tau) : \dot{1}$ .

From (K-SOME) rule,  $\mathcal{A}_{\text{tyarg}}(\Delta) \vdash_L \mathcal{A}_{\text{prim}}(\exists \alpha : K . \tau) : \dot{1}$

case 5 :

$\mathcal{A}_{\text{prim}}(\alpha : \text{primitive}) = \alpha$

From (KF-TVAR) rule,  $\Delta', \alpha : \text{primitive} \vdash_F \alpha : \text{primitive}$

By (K-TVAR) rule,  $\mathcal{A}_{\text{tyarg}}(\Delta'), \alpha : \mathcal{A}_{\text{tyarg}}(\text{primitive}) \vdash_L \alpha : \mathcal{A}_{\text{tyarg}}(\text{primitive})$

Thus,  $\mathcal{A}_{\text{tyarg}}(\Delta', \alpha : \text{primitive}) \vdash_L \alpha : \dot{1}$

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,  $\mathcal{A}_{\text{tyarg}}(\Delta'), \alpha : \mathcal{A}_{\text{tyarg}}(\text{ptr}) \vdash_L \alpha : \mathcal{A}_{\text{tyarg}}(\text{ptr})$

Thus,  $\mathcal{A}_{\text{tyarg}}(\Delta', \alpha : \text{ptr}) \vdash_L \alpha : \text{int} \rightarrow K_R \rightarrow \text{int} \rightarrow \text{int} \rightarrow 0$

2.  $\mathcal{A}(x) R E g = x$

By (T-VAR),  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\}, x : \mathcal{A}(\tau) R E \vdash_L x : \mathcal{A}(\tau) R E$

So  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma, x : \tau) R E, \{g : \tau_g(R, E)\} \vdash_L x : \mathcal{A}(\tau) R E$

3.  $\mathcal{A}(i) R E g = \text{pack}(i, i)$  as  $\exists I : \text{int} . \text{Int}(I)$

From (TF-INT),  $\Delta; \Gamma \vdash_F i : \text{int}$

Because by (K-IVAR)  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L i : \text{int}$ ,

by (T-PACK),  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \text{pack}(i, i)$  as  $\exists I : \text{int} . \text{Int}(I) : \exists I : \text{int} . \text{Int}(I)$

Thus,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(i) R E g : \mathcal{A}(\text{int}) R E$

4.  $\mathcal{A}(x_1 p x_2) R E g = O P_p \cdot \langle x_1, x_2 \rangle, (\Delta'; \Gamma' \vdash_L O P_p : \langle \tau_{\text{int}}, \tau_{\text{int}} \rangle \rightarrow \tau_{\text{int}})$

From (TF-OP) rule,  $\Delta; \Gamma \vdash_F x_1 : \text{int}$ , and  $\Delta; \Gamma \vdash_F x_2 : \text{int}$

By induction,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x_1 : \tau_{\text{int}}$

$\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x_2 : \tau_{\text{int}}$

Because  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L O P_p : \langle \tau_{\text{int}}, \tau_{\text{int}} \rangle \rightarrow \tau_{\text{int}}$

By (T-APP) rule,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(x_1 p x_2) R E g : \tau_{\text{int}}$

5.  $\mathcal{A}(x[\tau_0]) R E g = x[\mathcal{A}_{\text{tyarg}}(\tau_0)]$

From (TF-TAPP) rule, we know  $\Delta \vdash_F \tau_0 : K$ ,  $\Delta; \Gamma \vdash_F x : \forall[\alpha : K, \beta : K'] . (\vec{\tau}) \rightarrow \text{void}$

By induction,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int} \vdash_L \mathcal{A}_{\text{tyarg}}(\tau_0) : \mathcal{A}_{\text{tyarg}}(K)$

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$   
 $x : \forall \alpha : \mathcal{A}_{tyarg}(K), \beta : \mathcal{A}_{tyarg}(K'). \forall R : K_R. \forall E : \text{int}. \wedge \langle \tau_g(R, E), \overline{\mathcal{A}(\tau)} \rangle \rightarrow \tau_{halt}$   
 By (T-TAPP) rule,  $\overline{\mathcal{A}_{tyarg}(\Delta)}, R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$   
 $x[\mathcal{A}_{tyarg}(\tau_0)] : \forall \beta : \mathcal{A}_{tyarg}(K'). \forall R : K_R. \forall E : \text{int}. (\wedge \langle \tau_g(R, E), \overline{\mathcal{A}(\tau)} \rangle \rightarrow \tau_{halt})[\mathcal{A}_{tyarg}(\tau_0)/\alpha]$   
 Using Lemma2,  $\overline{\mathcal{A}_{tyarg}(\Delta)}, R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$   
 $x[\mathcal{A}_{tyarg}(\tau_0)] : \forall \beta : \mathcal{A}_{tyarg}(K'). \forall R : K_R. \forall E : \text{int}. \wedge \langle \tau_g(R, E), \overline{\mathcal{A}(\tau[\tau_0/\alpha])} \rangle \rightarrow \tau_{halt}$   
 Thus,  $\overline{\mathcal{A}_{tyarg}(\Delta)}, R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$   
 $\mathcal{A}(x[\tau_0]) R E g : \mathcal{A}(\forall \beta : K'). (\overline{\tau}[\tau_0/\alpha] \rightarrow \text{void}) R E$

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,  $\Delta \vdash_F \tau : K, \Delta; \Gamma \vdash_F x : \tau'[\tau/\beta]$

and  $\Delta; \Gamma \vdash_F \text{pack}[\tau, x] \text{ as } \exists \beta : K. \tau' : \exists \beta : K. \tau'$

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.*  $\mathcal{A}(\text{pack}[\tau, x] \text{ as } \exists \beta : K. \tau' : \text{ptr}) R E g = \text{ptr\_pack}_K R E (\lambda \beta : \mathcal{A}_{tyarg}(K). \mathcal{A}_{tyarg}(\tau')) \mathcal{A}_{tyarg}(\tau) x$

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  $\text{ptr\_pack}_K$  tells us that  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$

$\text{ptr\_pack}_K R E (\lambda \beta : \mathcal{A}_{tyarg}(K). \mathcal{A}_{tyarg}(\tau')) \mathcal{A}_{tyarg}(\tau) : \mathcal{A}(\tau'[\tau/\beta]) R E \rightarrow \mathcal{A}(\exists \gamma : K. (\tau'[\gamma/\beta])) R E$ .

From this we conclude  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$

$\text{ptr\_pack}_K R E (\lambda \beta : \mathcal{A}_{tyarg}(K). \mathcal{A}_{tyarg}(\tau')) \mathcal{A}_{tyarg}(\tau) x : \mathcal{A}(\exists \gamma : K. (\tau'[\gamma/\beta])) R E$ .

And  $\exists \gamma : K. (\tau'[\gamma/\beta]) = \exists \beta : K. \tau'$ .

7.  $\mathcal{A}(\text{fixcode } x[\overline{\alpha : K}](\overline{x : \tau}). e) R E g = \text{fix } x : \forall \alpha : \mathcal{A}_{tyarg}(K). \forall R' : K_R. \forall E' : \text{int}. \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \rightarrow \tau_{halt}(R')$ .

$\Lambda \alpha : \mathcal{A}_{tyarg}(K). \Lambda R' : K_R. \Lambda E' : \text{int}. \lambda y : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \rightarrow \text{let } \langle g', \overline{x} \rangle = y \text{ in } \mathcal{A}(e) R' E' g'$

From (TF-FIX) rule,  $\overline{\alpha : K} \vdash_F \tau_i : K'_i, \overline{\alpha : K}; x : \forall [\alpha : K]. (\overline{\tau}) \rightarrow \text{void}, x_1 : \tau_1, \dots, x_n : \tau_n \vdash_F e$

and  $\Delta; \Gamma \vdash_F \text{fixcode } x[\overline{\alpha : K}](\overline{x : \tau}). e : \forall [\alpha : K]. (\overline{\tau}) \rightarrow \text{void}$

By induction,  $\mathcal{A}_{tyarg}(\overline{\alpha : K}), R' : K_R, E' : \text{int} \vdash_L \mathcal{A}(\tau_i) R' E' : \mathbb{1}$

$\mathcal{A}_{tyarg}(\overline{\alpha : K}), R' : K_R, E' : \text{int}; x : \mathcal{A}(\forall [\alpha : K]. (\overline{\tau}) \rightarrow \text{void}) R' E', x_1 : \mathcal{A}(\tau_1) R' E', \dots, x_n : \mathcal{A}(\tau_n) R' E',$

$\{g : \tau_g(R', E')\} \vdash_L \mathcal{A}(e) R' E' g : \tau_{halt}$

by (T-VAR),  $y : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \vdash_L y : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle$

By (T-LET) rule,  $\mathcal{A}_{tyarg}(\overline{\alpha : K}), R' : K_R, E' : \text{int}; x : \mathcal{A}(\forall [\alpha : K]. (\overline{\tau}) \rightarrow \text{void}) R' E', y : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \vdash_L$

$\text{let } \langle g', \overline{x} \rangle = y \text{ in } \mathcal{A}(e) R' E' g' : \tau_{halt}$

By (T-ABS) rule, we obtain  $\mathcal{A}_{tyarg}(\overline{\alpha : K}), R' : K_R, E' : \text{int}; x : \mathcal{A}(\forall [\alpha : K]. (\overline{\tau}) \rightarrow \text{void}) R' E' \vdash_L$

$\lambda y : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \rightarrow \text{let } \langle g', \overline{x} \rangle = y \text{ in } \mathcal{A}(e) R' E' g' : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \rightarrow \tau_{halt}$

Because  $\overline{\alpha : \mathcal{A}_{tyarg}(K)} \vdash_L \overline{\alpha : \mathcal{A}_{tyarg}(K)}, E' : \text{int} \vdash_L E' : \text{int}$ , and  $R' : K_R \vdash_L R' : K_R$ ,

by (T-TABS) rule,  $\emptyset; x : \mathcal{A}(\forall [\alpha : K]. (\overline{\tau}) \rightarrow \text{void}) R' E' \vdash_L$

$\Lambda \alpha : \mathcal{A}_{tyarg}(K). \Lambda R' : K_R. \Lambda E' : \text{int}. \lambda y : \wedge \langle \tau_g(R', E'), \overline{\mathcal{A}(\tau) R' E'} \rangle \rightarrow \text{let } \langle g', \overline{x} \rangle = y \text{ in } \mathcal{A}(e) R' E' g' :$



$\overrightarrow{\forall \alpha : \mathcal{A}_{tyarg}(K). \forall R' : K_R. \forall E' : \text{int}. \wedge \langle \tau_g(R', E'), \overrightarrow{\mathcal{A}(\tau) R' E'} \rangle \rightarrow \tau_{halt}}$   
 By (T-FIX) rule and weakening lemma,  $\overrightarrow{\mathcal{A}_{tyarg}(\Delta), R' : K_R, E' : \text{int}; \mathcal{A}(\Gamma) R' E', \{g : \tau_g(R', E')\}} \vdash_L$   
 $\mathcal{A}(\text{fixcode } x[\alpha : K](x : \overrightarrow{\tau}). e) R E g : \mathcal{A}(\overrightarrow{\forall [\alpha : K]. (\overrightarrow{\tau}) \rightarrow \text{void}}) R' E'$

8.  $\mathcal{A}(x(\overrightarrow{x})) R E g = x R E \wedge \langle g, \overrightarrow{x} \rangle$

From (TF-APP) rule,  $\Delta; \Gamma \vdash_F x_i : \tau_i, \Delta; \Gamma \vdash_F x : (\tau_1, \dots, \tau_n) \rightarrow \text{void}$ ,  
 and  $\Delta; \Gamma \vdash_F x(\overrightarrow{x})$

By induction,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x_i : \mathcal{A}(\tau_i) R E$

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x : \forall R : K_R. \forall E : \text{int}. \wedge \langle \tau_g(R, E), \overrightarrow{\mathcal{A}(\tau) R E} \rangle \rightarrow \tau_{halt}$

By (T-APP) rule, we get  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x R E \wedge \langle g, \overrightarrow{x} \rangle : \tau_{halt}$

9.  $\mathcal{A}(\text{if0}(x, e_1, e_2)) R E g = \text{unpack } \beta, x_1 = x \text{ in if } x_1 = 0 \text{ then } \mathcal{A}(e_1) R E g \text{ else } \mathcal{A}(e_2) R E g$

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,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g\} R E \vdash_L x : \tau_{int}$

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(e_1) R E g : \tau_{halt}$

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(e_2) R E g : \tau_{halt}$

Because  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}, \beta : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\}, x_1 : \text{Int}(\beta) \vdash_L (x_1 = 0) : \text{Bool}(\beta = 0)$

From (T-IFE) rule, we know

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}, \beta : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\}, x_1 : \text{Int}(\beta) \vdash_L \text{if } x_1 = 0 \text{ then } \mathcal{A}(e_1) R E g \text{ else } \mathcal{A}(e_2) R E g :$

$\tau_{halt}$

By (T-UNPACK) rule,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(\text{if0}(x, e_1, e_2)) R E g : \tau_{halt}$

10.  $\mathcal{A}(\text{halt}[\tau]x) R E g = \text{halt } R E (\mathcal{A}(\tau) R E) \wedge \langle g, x \rangle$ ,

From (TF-HALT) rule, we know  $\Delta; \Gamma \vdash_F x : \tau$ , and  $\Delta; \Gamma \vdash_F \text{halt}[\tau]x$

By induction,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x : \mathcal{A}(\tau) R E$

Because  $\emptyset; \emptyset; \emptyset; \emptyset; \text{true}; \infty \vdash_L \text{halt} : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \text{int}. \wedge \langle \tau_g(R, E), \alpha \rangle \rightarrow \tau_{halt}$

By (T-APP) rule,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \text{halt } R E (\mathcal{A}(\tau) R E) \wedge \langle g, x \rangle : \tau_{halt}$

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,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(t) : \mathcal{A}(\tau) R E$

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\}, x : \mathcal{A}(\tau) R E \vdash_L \mathcal{A}(e) : \tau_{halt}$

By (T-LET), we get  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(\text{let } x = t \text{ in } e) R E g : \tau_{halt}$

12.  $\mathcal{A}(\text{let } x_0 = \pi_i(x_1 : \overrightarrow{\langle \tau : K \rangle}) \text{ in } e) R E g = \text{let } \langle g', x_0 \rangle = \text{obj\_load}_{\overrightarrow{K}, i} R E \overrightarrow{\mathcal{A}_{tyarg}(\tau)} \wedge \langle g, x_1 \rangle \text{ in } \mathcal{A}(e) R E g'$

From (TF-PROJECT),  $\Delta; \Gamma \vdash_F x_1 : \langle \tau_1, \dots, \tau_n \rangle, \Delta; \Gamma, x_0 : \tau_i \vdash_F e$ ,

and  $\Delta; \Gamma \vdash_F \text{let } x_0 = \pi_i x_1 \text{ in } e$

By induction,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L x_1 : \mathcal{A}(\overrightarrow{\langle \tau : K \rangle}) R E$

$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g' : \tau_g(R, E)\}, x_0 : \mathcal{A}(\tau_i) R E \vdash_L \mathcal{A}(e) R E g' : \tau_{halt}$

Because  $\emptyset; \emptyset \vdash_L \text{obj\_load}_{\overrightarrow{K}, i} : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \mathcal{A}_{tyarg}(K). \wedge \langle \tau_g(R, E), \mathcal{A}(\overrightarrow{\langle \alpha : K \rangle}) R E \rangle \rightarrow \wedge \langle \tau_g(R, E), \mathcal{A}(\alpha_i :$

$K_i) R E \rangle$

By (T-APP) rule and lemma2,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L$

$\text{obj\_load}_{\overrightarrow{K}, i} R E \overrightarrow{\mathcal{A}_{tyarg}(\tau)} \wedge \langle g, x_1 \rangle : \wedge \langle \tau_g(R, E), \mathcal{A}(\tau_i : K_i) R E \rangle$

By (T-LET) rule,  $\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) R E, \{g : \tau_g(R, E)\} \vdash_L$

$\mathcal{A}(\text{let } x_0 = \pi_i(x_1 : \overrightarrow{\langle \tau : K \rangle}) \text{ in } e) \text{ RE } g : \tau_{\text{halt}}$

13.

*case 1.*  $\mathcal{A}(\text{let } [\alpha : K, x_0 : \tau_0 : \text{primitive}] = \text{unpack } x_1 : \exists \alpha : K. \tau \text{ in } e) \text{ RE } g = \text{unpack } \alpha, x_0 = x_1 \text{ in } \mathcal{A}(e) \text{ RE } g$

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,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L x_1 : \mathcal{A}(\exists \alpha : K. \tau) \text{ RE}$

$\mathcal{A}_{\text{tyarg}}(\Delta, \alpha : K), R : K_R, E : \text{int}; \mathcal{A}(\Gamma, x_0 : \tau) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L \mathcal{A}(e) \text{ RE } g : \tau_{\text{halt}}$

Because  $\Delta; \Gamma \vdash_F \tau : \text{primitive},$

$\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L x_1 : \exists \alpha : \mathcal{A}_{\text{tyarg}}(K). \mathcal{A}_{\text{prim}}(\tau)$

$\mathcal{A}_{\text{tyarg}}(\Delta), \alpha : \mathcal{A}_{\text{tyarg}}(K), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\}, x_0 : \mathcal{A}_{\text{prim}}(\tau) \text{ RE} \vdash_L \mathcal{A}(e) \text{ RE } g : \tau_{\text{halt}}$

By (T-UNPACK) rule,

$\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L \text{unpack } \alpha, x_0 = x_1 \text{ in } \mathcal{A}(e) \text{ RE } g : \tau_{\text{halt}}$

*case 2.*  $\mathcal{A}(\text{let } [\alpha : K, x_0 : \tau_0 : \text{ptr}] = \text{unpack } x_1 : \exists \alpha : K. \tau \text{ in } e) \text{ RE } g =$

$\text{unpack } \alpha, x_0 = \text{ptr\_unpack}_K \text{ RE } (\lambda \alpha : \mathcal{A}_{\text{tyarg}}(K). \mathcal{A}_{\text{tyarg}}(\tau)) x_1 \text{ in } \mathcal{A}(e) \text{ RE } g$

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,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L x_1 : \mathcal{A}(\exists \alpha : K. \tau) \text{ RE}$

$\mathcal{A}_{\text{tyarg}}(\Delta), \mathcal{A}_{\text{tyarg}}(\alpha) : \mathcal{A}_{\text{tyarg}}(K), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\}, x_0 : \mathcal{A}(\tau) \text{ RE} \vdash_L \mathcal{A}(e) \text{ RE } g : \tau_{\text{halt}}$

From definition we know  $\emptyset; \emptyset \vdash_L$

$\text{ptr\_unpack}_K : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \mathcal{A}_{\text{tyarg}}(K) \rightarrow \mathcal{A}_{\text{tyarg}}(\text{ptr}). \mathcal{A}(\exists \beta : K. (\alpha \beta)) \text{ RE} \rightarrow \exists \beta : \mathcal{A}_{\text{tyarg}}(K). (\mathcal{A}(\alpha \beta) \text{ RE})$

Lemma2 tells us that  $(\lambda \alpha : \mathcal{A}_{\text{tyarg}}(K). \mathcal{A}_{\text{tyarg}}(\tau)) \mathcal{A}_{\text{tyarg}}(\alpha) \equiv \mathcal{A}_{\text{tyarg}}(\tau)[\alpha/\alpha] = \mathcal{A}_{\text{tyarg}}(\tau) = \mathcal{A}_0(\tau)$ . Using this

with the type of  $\text{ptr\_unpack}_K$  tells us that  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L$

$\text{ptr\_unpack}_K \text{ RE } (\lambda \alpha : \mathcal{A}_{\text{tyarg}}(K). \mathcal{A}_{\text{tyarg}}(\tau)) : \mathcal{A}(\exists \alpha : K. \tau) \text{ RE} \rightarrow \exists \alpha : \mathcal{A}_{\text{tyarg}}(K). (\mathcal{A}(\tau) \text{ RE})$ .

Thus,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L$

$\text{ptr\_unpack}_K \text{ RE } (\lambda \alpha : \mathcal{A}_{\text{tyarg}}(K). \mathcal{A}_{\text{tyarg}}(\tau)) x_1 : \exists \alpha : \mathcal{A}_{\text{tyarg}}(K). (\mathcal{A}(\tau) \text{ RE})$

Because  $\mathcal{A}_{\text{tyarg}}(\Delta), \mathcal{A}_{\text{tyarg}}(\alpha) : \mathcal{A}_{\text{tyarg}}(K), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\}, x_0 : \mathcal{A}(\tau) \text{ RE} \vdash_L \mathcal{A}(e) \text{ RE } g :$

$\tau_{\text{halt}}$

By (T-UNPACK) rule,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L$

$\text{unpack } \alpha, x_0 = \text{ptr\_unpack}_K \text{ RE } (\lambda \alpha : \mathcal{A}_{\text{tyarg}}(K). \mathcal{A}_{\text{tyarg}}(\tau)) x_1 \text{ in } \mathcal{A}(e) \text{ RE } g : \tau_{\text{halt}}$

14.  $\mathcal{A}(\text{let } x_0 = \overrightarrow{\langle x : \tau_x \rangle} \text{ in } e) \text{ RE } g = \text{unpack } E', z = \text{alloc}_{\overrightarrow{K}, n} \text{ RE } \overrightarrow{\mathcal{A}_{\text{tyarg}}(\tau_x)} \overrightarrow{\mathcal{A}_{\text{tyarg}}(\tau_y)} \wedge \langle g, \overrightarrow{x}, \overrightarrow{y} \rangle \text{ in}$

$\text{let } \langle g', x_0, \overrightarrow{y} \rangle = z \text{ in } \mathcal{A}(e) \text{ RE } g'; \text{ where } \{y_1, \dots, y_n\} = \{y : \tau_y : \text{ptr} \in \text{FV}(e)\} - \{x_0\}$

From (TF-VEC) rule, we know  $\Delta; \Gamma \vdash_F x_i : \tau_i, \Delta; \Gamma, x_0 : \langle \tau_1, \dots, \tau_n \rangle \vdash_F e,$

$\Delta; \Gamma \vdash_F \text{let } x_0 = \overrightarrow{\langle x \rangle} \text{ in } e$

By induction, we get  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L x_i : \mathcal{A}(\tau_i) \text{ RE},$

and  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E' : \text{int}; \mathcal{A}(\Gamma) \text{ RE}', \{g' : \tau_g(R, E')\}, x_0 : \mathcal{A}(\overrightarrow{\langle \tau_x \rangle}) \text{ RE}' \vdash_L \mathcal{A}(e) \text{ RE } g' : \tau_{\text{halt}}$

From definition,  $\emptyset; \emptyset \vdash_L \text{alloc}_{\overrightarrow{K}, n} : \forall R : K_R. \forall E : \text{int}. \forall \alpha : \mathcal{A}_{\text{tyarg}}(K). \forall \beta_1 : \mathcal{A}_{\text{tyarg}}(\text{ptr}). \dots$

$\forall \beta_n : \mathcal{A}_{\text{tyarg}}(\text{ptr}). \wedge \langle \tau_g(R, E), \overrightarrow{\mathcal{A}(\alpha : K) \text{ RE}}, \overrightarrow{\mathcal{A}(\beta : \text{ptr}) \text{ RE}} \rangle \rightarrow \exists E'. \wedge \langle \tau_g(R, E'), \overrightarrow{\mathcal{A}(\langle \alpha : K \rangle) \text{ RE}'}, \overrightarrow{\mathcal{A}(\beta : \text{ptr}) \text{ RE}'} \rangle$

Using lemma2,  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma) \text{ RE}, \{g : \tau_g(R, E)\} \vdash_L \text{alloc}_{\overrightarrow{K}, n} \text{ RE } \overrightarrow{\mathcal{A}_{\text{tyarg}}(\tau_x)} \overrightarrow{\mathcal{A}_{\text{tyarg}}(\tau_y)} \wedge \langle g, \overrightarrow{x}, \overrightarrow{y} \rangle :$

$\exists E'. \wedge \langle \tau_g(R, E'), \overrightarrow{\mathcal{A}(\langle \tau_x \rangle) \text{ RE}'}, \overrightarrow{\mathcal{A}(\tau_y) \text{ RE}'} \rangle$

We know  $\mathcal{A}_{\text{tyarg}}(\Delta), R : K_R, E' : \text{int}; \mathcal{A}(\Gamma) \text{ RE}', \{g' : \tau_g(R, E')\}, x_0 : \mathcal{A}(\overrightarrow{\langle \tau_x \rangle}) \text{ RE}' \vdash_L \mathcal{A}(e) \text{ RE } g' : \tau_{\text{halt}}.$

Say that  $\Gamma = \Gamma_{\text{ptr}} \cup \Gamma_{\text{ptretra}} \cup \Gamma_{\text{prim}}$  where  $\Gamma_{\text{ptr}}, \Gamma_{\text{ptretra}}, \Gamma_{\text{prim}}$  have disjoint domains and:

$\Gamma_{\text{ptr}} = \{y_1 \mapsto \tau_{y_1} : \text{ptr}, \dots, y_n \mapsto \tau_{y_n} : \text{ptr}\}$

$\Gamma_{\text{ptretra}} = \{y'_1 \mapsto \tau_{y'_1} : \text{ptr}, \dots, y'_{n'} \mapsto \tau_{y'_{n'}} : \text{ptr}\}$

$\Gamma_{\text{prim}} = \{z \mapsto \tau_{z_1} : \text{prim}, \dots, z_m \mapsto \tau_{z_m} : \text{prim}\}$

All free pointer variables in  $e$  are in  $\Gamma_{ptr}$ , so no free variables of  $e$  appear in  $\Gamma_{ptr\text{extra}}$ . It's easy to show that  $e$  typechecks without  $\Gamma_{ptr\text{extra}}$  (by induction on the typing derivation), and that each free pointer variable  $y_i$  in  $e$  must appear in  $\Gamma$  (also by induction on the typing derivation):

$$\mathcal{A}_{tyarg}(\Delta), R : K_R, E' : \text{int}; \mathcal{A}(\Gamma_{ptr} \cup \Gamma_{prim})RE', \{g' : \tau_g(R, E')\}, x_0 : \mathcal{A}(\langle \overline{\tau_x} \rangle) RE' \vdash_L \mathcal{A}(e) RE' g' : \tau_{halt}.$$

Then by (T-LET) and (T-VAR):

$$\mathcal{A}_{tyarg}(\Delta), R : K_R, E' : \text{int}; \mathcal{A}(\Gamma_{prim})RE', z : \wedge \langle \tau_g(R, E'), \mathcal{A}(\langle \overline{\tau_x} \rangle) RE', \overline{\mathcal{A}(\tau_y) RE'} \rangle \vdash_L$$

$$\text{let } \langle g', x_0, \overline{y} \rangle = z \text{ in } \mathcal{A}(e) RE' g' : \tau_{halt}$$

For primitive types,  $\mathcal{A}(\tau : \text{primitive}) = \lambda R : K_R. \lambda E : \text{int}. \mathcal{A}_{prim}(\tau)$ , so  $\mathcal{A}(\tau : \text{primitive}) RE' = \mathcal{A}(\tau : \text{primitive}) RE$ . This means that  $\mathcal{A}(\Gamma_{prim}) RE' = \mathcal{A}(\Gamma_{prim}) RE$ , which we can use after weakening the environment with  $E$ :

$$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}, E' : \text{int}; \mathcal{A}(\Gamma_{prim})RE, z : \wedge \langle \tau_g(R, E'), \mathcal{A}(\langle \overline{\tau_x} \rangle) RE', \overline{\mathcal{A}(\tau_y) RE'} \rangle \vdash_L$$

$$\text{let } \langle g', x_0, \overline{y} \rangle = z \text{ in } \mathcal{A}(e) RE' g' : \tau_{halt}$$

We can weaken  $\Gamma_{prim}$  to the superset  $\Gamma$ :

$$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}, E' : \text{int}; \mathcal{A}(\Gamma)RE, z : \wedge \langle \tau_g(R, E'), \mathcal{A}(\langle \overline{\tau_x} \rangle) RE', \overline{\mathcal{A}(\tau_y) RE'} \rangle \vdash_L$$

$$\text{let } \langle g', x_0, \overline{y} \rangle = z \text{ in } \mathcal{A}(e) RE' g' : \tau_{halt}$$

By (T-UNPACK), we get

$$\mathcal{A}_{tyarg}(\Delta), R : K_R, E : \text{int}; \mathcal{A}(\Gamma)RE, \{g : \tau_g(R, E)\} \vdash_L$$

$$\text{unpack } E', z = \text{alloc}_{\overline{R}, n} RE \overline{\mathcal{A}_{tyarg}(\tau_x)} \overline{\mathcal{A}_{tyarg}(\tau_y)} \wedge \langle g, \overline{x}, \overline{y} \rangle \text{ in } \text{let } \langle g', x_0, \overline{y} \rangle = z \text{ in } \mathcal{A}(e) RE' g' : \tau_{halt}$$

## 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 Cray, 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.