Dartmouth College
Dartmouth Digital Commons

Computer Science Technical Reports | Computer Science

11-26-2002

Proofs of Soundness and Strong Normalization for Linear Memory Types

Heng Huang
Dartmouth College

Chris Hawblitzel
Dartmouth College

Follow this and additional works at: https://digitalcommons.dartmouth.edu/cs_tr

 Part of the Computer Sciences Commons

Dartmouth Digital Commons Citation

Huang, Heng and Hawblitzel, Chris, "Proofs of Soundness and Strong Normalization for Linear Memory Types" (2002). Computer Science Technical Report TR2002-437. https://digitalcommons.dartmouth.edu/cs_tr/204

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.

Proofs of Soundness and Strong Normalization for Linear Memory Types Dartmouth Technical Report TR2002-437

Heng Huang and Chris Hawblitzel

Nov 26, 2002

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.

Introduction

This report presents the complete syntax and rules for the abstract machine 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 [3] (also see [2] 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).

1 Abstract syntax

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

linearity

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

time limits

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

kinds

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

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

arithmetic

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

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

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

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

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

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

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

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

types

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

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

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

expressions

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

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

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

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

$$\mid \text{store}(e_{\text{ptr}}\ , e_{\text{Has}}, \text{e}_\text{v}) \mid \text{coerce}(e) \mid \text{distinguish}(I_1, I_2, e_1, e_2) \mid \text{make_eq}(\tau)$$

$$\mid \text{apply_eq}(\tau, e_1, e_2) \mid \text{new_fun}(K) \mid \text{discard_fun}(e)$$

$$\mid \text{define_fun}(e, \tau) \mid \text{in_domain}(I_1, I_2, e_1, e_2) \mid \text{fact}$$

values

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

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

expressions for substitution

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

environments

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

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

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

$$\Phi = \{F_1^{K_1} \overset{\phi_1}{\mapsto} \text{fun}_1, \cdots, F_n^{K_n} \overset{\phi_n}{\mapsto} \text{fun}_n\}$$

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

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

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

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

judgments

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

$$B \vdash B_1 \doteq B_2$$

$$B \vdash I_1 \doteq I_2$$

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

$$C \vdash e : \tau$$

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

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

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

The environments Ψ, Φ, δ, Δ, Γ must be well-formed functions (and the definitions in this report apply only to well-formed functions):

$(i \mapsto \tau_1 \in \Psi) \wedge (i \mapsto \tau_2 \in \Psi) \Rightarrow \tau_1 = \tau_2$

$(F^K \overset{\phi_1}{\mapsto} \delta_1 \in \Phi) \wedge (F^K \overset{\phi_2}{\mapsto} \delta_2 \in \Phi) \Rightarrow (\delta_1 = \delta_2) \wedge (\phi_1 = \phi_2)$

$(i \mapsto \tau_1 \in \delta) \wedge (i \mapsto \tau_2 \in \delta) \Rightarrow \tau_1 = \tau_2$

$(\alpha \mapsto K_1 \in \Delta) \wedge (\alpha \mapsto K_2 \in \Delta) \Rightarrow K_1 = K_2$

$(x \mapsto \tau_1 \in \Gamma) \wedge (x \mapsto \tau_2 \in \Gamma) \Rightarrow \tau_1 = \tau_2$

For Ψ, δ, Δ, Γ we use the usual function application notation: $\Gamma(x) = \tau \Leftrightarrow x \mapsto \tau \in \Gamma$.

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

Environment splitting

These definitions split environments into two parts, where linear elements must go into exactly one of the parts:

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

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

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

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

Environment extension

These add new elements to environments:

$\Psi, i \mapsto \tau \triangleq \Psi \cup \{i \mapsto \tau\}$, where $i \notin \mathit{domain}(\Psi)$

$\Phi, F^K \overset{\phi}{\mapsto} \delta \triangleq \Phi \cup \{F^K \overset{\phi}{\mapsto} \delta\}$, where $F^K \notin \mathit{domain}(\Phi)$

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

Nonlinear environments

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

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

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

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

Environment subsets

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

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

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

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

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

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

where we define subset for Γ as:

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

where $\Phi; \Delta \vdash \tau_1 :\overset{\cdot i_1}{\text{type}}, \ldots, \Phi; \Delta \vdash \tau_n :\overset{\cdot i_n}{\text{type}}$.

1.2 Arithmetic constraints

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

2 Type equivalence

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Evaluation rules

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$(E-DOMAIN)\text{in_domain}(I_1, I_2, fact, fact) \rightarrow^{\wedge} \langle \text{know}(0 \leq I_1 \wedge I_1 < I_2), fact\rangle$$

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

$$(E-MAKEEQ)\text{make_eq}(\tau) \rightarrow fact$$

$$(E-APPLYEQ)\text{apply_eq}(\tau, fact, v) \rightarrow v$$

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

$$(E-DISCARDFUN)\text{discard_fun}(fact) \rightarrow \cdot\langle\rangle$$

$$(E-DEFINEFUN)\text{define_fun}(fact, \tau) \rightarrow^{\wedge} \langle fact, fact, fact\rangle$$

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

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

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

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

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

$$\mid \mathrm{distinguish}(I_1, I_2, e, e_2) \mid \mathrm{distinguish}(I_1, I_2, \mathrm{v}_1, e) \mid \mathrm{apply_eq}(\tau, e, e_2) \mid \mathrm{apply_eq}(\tau, \mathrm{v}_1, e)$$

$$\mid \mathrm{discard_fun}(e) \mid \mathrm{define_fun}(e, \tau) \mid \mathrm{in_domain}(I_1, I_2, e, e_2) \mid \mathrm{in_domain}(I_1, I_2, \mathrm{v}_1, e)$$

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

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

4 Type well-formedness

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

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

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

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

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

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

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

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

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

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

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

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

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

$$(K-NLTUPLE)\frac{\forall j.(\Phi;\Delta\vdash \tau_j:\overset{\cdot i_j}{\text{type}})}{\Phi;\Delta\vdash \cdot\langle\vec{\tau}\rangle:\overset{\cdot\sum_j i_j}{\text{type}}}$$

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

$$(K-INT)\frac{\Phi;\Delta\vdash I:\text{int}}{\Phi;\Delta\vdash \text{Int}(I):\overset{\cdot 1}{\text{type}}}$$

$$(K-BOOL)\frac{\Phi;\Delta\vdash B:\text{bool}}{\Phi;\Delta\vdash \text{Bool}(B):\overset{\cdot 1}{\text{type}}}$$

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

$$(K-HAS)\frac{\Phi;\Delta\vdash I:\text{int}\quad \Phi;\Delta\vdash\tau:\overset{\cdot 1}{\text{type}}}{\Phi;\Delta\vdash \text{Has}(I,\tau):\overset{\wedge 0}{\text{type}}}$$

$$(K-FUNGEN)\frac{\Phi;\Delta\vdash I:\text{int}\quad \Phi;\Delta\vdash\tau:\text{int}\to J}{\Phi;\Delta\vdash \text{Gen}(\tau,I):\overset{\wedge 0}{\text{type}}}$$

$$(K-INDOMAIN)\frac{\Phi;\Delta\vdash I:\text{int}\quad \Phi;\Delta\vdash\tau:\text{int}\to J}{\Phi;\Delta\vdash \text{InDomain}(I,\tau):\overset{\cdot 0}{\text{type}}}$$

$$(K-EQ)\frac{\Phi;\Delta\vdash \tau_1:K\quad \Phi;\Delta\vdash\tau_2:K}{\Phi;\Delta\vdash \text{Eq}(\tau_1,\tau_2):\overset{\cdot 0}{\text{type}}}$$

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

5 Type checking rules

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

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

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

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

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

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

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

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

$$(T-TAPP)\frac{\begin{array}{c}C_1,C_2=\Psi;\Phi;\Delta;\Gamma;B;\mathrm{limit}_C\\ C_1\vdash e_1:\tau_a\ \overset{\phi,\mathrm{limit}_f}{\longrightarrow}\ \tau_b\quad C_2\vdash e_2:\tau_a\\ (\mathrm{limit}_C=\mathrm{limit}_f=\infty)\,\mathrm{or}\,(B\vdash\mathrm{limit}_f<\mathrm{limit}_C)\\ \mathrm{or}\,(\mathrm{limit}_C=\infty,\ \mathrm{limit}_f=I,\ \Phi;\Delta\vdash\tau_b:\overset{\phi_b 0}{\mathrm{type}})\end{array}}{C_1,C_2\vdash e_1e_2:\tau_b}$$

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

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

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

$$(T-UNPACK)\frac{\begin{array}{c}C_1\vdash e_1:\exists\alpha:K;B.\tau_1\qquad C_1,C_2\vdash\tau_2:K_2\\ C_2,\alpha:K,x:\tau_1,B\vdash e_2:\tau_2\end{array}}{C_1,C_2\vdash\mathrm{unpack}\,\alpha,x=e_1\,\mathrm{in}\,e_2:\tau_2}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 $C_1 \dot{\subseteq} C_1'$, then there is some C_2' such that $C_2 \hat{\subseteq} C_2'$ and $C_1, C_2 \dot{\subseteq} C_1', C_2'$.

If $C = C_1, C_2$ and $C_1 \dot{\subseteq} C_1'$, then there is some C_2' such that $C_2 \subseteq C_2'$ and $C_1, C_2 \dot{\subseteq} C_1', C_2'$. Proof: if $C_1 \dot{\subseteq} C_1'$, 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 $F^K \overset{\phi}{\mapsto} \delta$, it can add new elements to δ. In this case, simply add the same elements to the same δ in C_2 (by the definition of splitting C_1, C_2, we know C_2 contains the same δ) to get C_2'.
- It can change a $F^K \hat{\mapsto} \delta$ to $F^K \dot{\mapsto} \delta$. In this case, C_2 must already contain $F^K \dot{\mapsto} \delta$, so choose $C_2' = C_2$.
- It can add a new $F^K \overset{\phi}{\mapsto} \delta$, which appears neither in C_1 nor C_2. In this case, add $F^K \dot{\mapsto} \delta$ to C_2 to get C_2'.

6.2 Preservation

6.2.1 LEMMA [INVERSION]

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

1. (VAR) If $C \vdash x : \tau$, then $x : \tau \in C$.
2. (APP) If $C \vdash e_1 e_2 : \tau$, then there is some type τ_a such that $C_1 \vdash e_1 : \tau_a \overset{\text{limit}_f}{\longrightarrow} \tau'$, $\tau \equiv \tau'$ and $C_2 \vdash e_2 : \tau_a$, (($\text{limit}_C = \text{limit}_f = \infty$) or ($\text{limit}_f < \text{limit}_C$) or ($\text{limit}_C = \infty$, $\text{limit}_f = I$, $\Phi; \Delta \vdash \tau' :\overset{\phi 0}{\text{type}}$)).
3. (ABS) If $C \vdash \lambda x : \tau_x \overset{\phi, \text{limit}_f}{\longrightarrow} e : \tau_1 \overset{\text{limit}_f}{\longrightarrow} \tau_2$, then $\tau_1 \equiv \tau_x$ and $\Psi; \Phi; \Delta; \Gamma, x : \tau_x; B; \text{limit}_f \vdash e : \tau_2$. ($C \vdash \tau_1 :: K_1$, $C \vdash \tau_x :: K_1$), ($\text{limit}_f = \infty$) 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 $C, \alpha : K_\alpha, B' \vdash v : \tau'$, $B' \equiv B_\alpha$, $\tau' \equiv \tau$.
6. (TUPLE)If $C \vdash \phi\langle \vec{e} \rangle : \tau$, then there is $\phi\langle \vec{\tau'} \rangle \equiv \tau$, and $C \vdash \phi\langle \vec{e} \rangle : \phi\langle \vec{\tau'} \rangle$, $\forall i.(C_i \vdash e_i : \tau_i')$, $C \vdash \phi\langle \vec{\tau'} \rangle : K$.
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 $C_{e_a} \vdash e_a : \phi\langle \vec{\tau'} \rangle$, $C_{e_b}, x : \vec{\tau'} \vdash e_b : \tau_b$.

8. (LOAD)If $C \vdash \text{load}(e_{\text{ptr}}, e_{\text{Has}}) : \tau$, then there is $\wedge\langle \tau', \text{Has}(I, \tau')\rangle \equiv \tau$ and $C \vdash \text{load}(e_{\text{ptr}}, e_{\text{Has}}) :\wedge \langle \tau', \text{Has}(I, \tau')\rangle$, $C_{e_{\text{ptr}}} \vdash e_{\text{ptr}} : \text{Int}(I)$, $C_{e_{\text{Has}}} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$, $(C = C_{e_{\text{ptr}}}, C_{e_{\text{Has}}})$.

9. (STORE) $C \vdash \text{store}(e_{\text{ptr}}, e_{\text{Has}}, e_{\text{v}}) : \tau$, then there is $\text{Has}(I, \tau_2) \equiv \tau$, and $C_{e_{\text{ptr}}} \vdash e_{\text{ptr}} : \text{Int}(I)$, $C_{e_{\text{has}}} \vdash e_{\text{has}} : \text{Has}(I, \tau_1)$, $C_{e_{\text{v}}} \vdash e_{\text{v}} : \tau_2$, $C \vdash \tau_2 :\overset{\cdot 1}{\text{type}}$.

10. (FIX)$C \vdash \text{fix}\, x : \tau_x.v : \tau$, then $\tau \equiv \tau_x$ and $C, x : \tau_x \vdash v : \tau$, $C \vdash \tau :\overset{\phi i}{\text{type}}$

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 $C \vdash (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n : K$, $C \vdash e_0 : (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n$

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 $C \vdash (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n : K$, $C \vdash e : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$

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 $C \vdash \tau_1 : K$, $C \vdash [\alpha \mapsto \tau_1]\tau_B$, $C \vdash e : [\alpha \mapsto \tau_1]\tau_2'$, $C \vdash \exists\alpha : K; \tau_B.\tau_2' :\overset{\phi i}{\text{type}}$

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

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

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

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

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

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

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

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

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

Proof :

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

1. (*VAR*) If $C \vdash x : \tau$, then $x : \tau \in C$.

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

If using (T-EQ), then there will be τ_n. For τ_n, we will use (T-VAR) for $C \vdash x : \tau_n$. $(\tau_1 \equiv \cdots \equiv \tau_n \equiv \tau)$

If using (T-VAR), then τ_n is τ.

By (T-VAR) rule, we obtain $\dot{C}_0, x : \tau_n \vdash x : \tau_n$, then $C = \dot{C}_0, x : \tau_n$

By (T-EQ) rule, thus $x : \tau \in C$.

2. (*APP*) If $C \vdash e_1 e_2 : \tau$, then there is some type τ_a such that $C_1 \vdash e_1 : \tau_a \overset{\text{limit}_f}{\longrightarrow} \tau'$, $\tau \equiv \tau'$ and $C_2 \vdash e_2 : \tau_a$, $((\text{limit}_C = \text{limit}_f = \infty)$ or $(\text{limit}_f < \text{limit}_C)$ or $(\text{limit}_C = \infty, \text{limit}_f = I, \Phi; \Delta \vdash \tau' :\overset{\phi 0}{\text{type}}))$.

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 τ'. For τ', we will use (T-APP) for $C \vdash e_1 e_2 : \tau'$. $(\tau \equiv \tau_1 \equiv \cdots \equiv \tau_n \equiv \tau')$

If using (T-APP), then τ' is τ.

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

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

6.2.2 LEMMA [LIMIT-CHANGE]

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

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

Proof by induction on the type derivation:

1. $s = i$

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

By (T-INT) rule,

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

2. $s = b$

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

By (T-BOOL) rule,

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

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

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

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 $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash \lambda x : \tau \overset{\phi,\text{limit}_f}{\longrightarrow} v : \tau \xrightarrow{f} \tau_v$

4. $s = \text{fact}$

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

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

For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\varnothing; \dot{\Phi}; \Delta; \dot{\Gamma}; B; \text{limit}_1 \vdash \text{fact} : \tau'$, and we will only use one rule of the following rules.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6. $s = \text{pack}[\tau_1, v] \text{ as } \exists\alpha : K; B.\tau_2$

$\Psi; \Phi; \Delta; \Gamma; B_1; \text{limit}_1 \vdash \text{pack}[\tau_1, v] \text{ as } \exists\alpha : K; B.\tau_2 : \tau$

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

For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\Psi; \Phi; \Delta; \Gamma; B_1; \text{limit}_1 \vdash \text{pack}[\tau_1, v] \text{ as } \exists\alpha : K; B.\tau_2 : \tau'$,

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

By (T-PACK) rule, we get:

$\tau' = \exists\alpha : K; B.\tau_2$, and $\Phi; \Delta \vdash \tau_1 : K$, $\Phi; \Delta \vdash \exists\alpha : K; B.\tau_2 :\overset{\phi i}{\text{type}}$

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

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. $s = \text{roll}[\tau](v) \quad \tau = (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n$

By Lemma (6.2.1.(12)), we have $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{roll}[\tau](v) : \tau$

$\Phi; \Delta \vdash (\mu\alpha : K.\tau_0)\tau_1 \cdots \tau_n : K$,

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

By induction, we obtain $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash v : ([\alpha \mapsto \mu\alpha : K.\tau_0]\tau_0)\tau_1 \cdots \tau_n$

Then by (T-ROLL) rule, we have

$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_2 \vdash \text{roll}[\tau](v) : \tau$

8. $s = \text{union}(b, \tau_1, \tau_2, v)$

$\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{union}(b, \tau_1, \tau_2, v) : \tau$

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

For (T-EQ) rule, then there is a $\tau' \equiv \tau$, and $\Psi; \Phi; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{union}(b, \tau_1, \tau_2, v) : \tau'$,

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

By (T-UNION) rule, we get:

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

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

By (T-UNION) rule, $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \text{union}(b,\tau_1,\tau_2,v) : \tau'$
By (T-EQ) rule, thus $\Psi;\Phi;\Delta;\Gamma;B;\text{limit}_2 \vdash \text{union}(b,\tau_1,\tau_2,v) : \tau$

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

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

If $C,\overrightarrow{\alpha : K_\alpha} \vdash \tau : K$, and $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$, then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$.
Proof by induction on the kinding derivation.
1. $\tau = \beta$
If $\beta = \alpha_i$, then $C,\overrightarrow{\alpha : K_\alpha} \vdash \tau : K$, $K = K_{\alpha_i}$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = \tau_{\alpha_i}$, then $C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K_{\alpha_i}$
assuming $C = C_1, \tau_{\alpha_i} : K_{\alpha_i}$, then
$C_1, \tau_{\alpha_i} : K_{\alpha_i} \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K_{\alpha_i}$
By (K-TVAR) rule, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C_1, \tau_{\alpha_i} : K_{\alpha_i} \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$
and $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C_1, [\overrightarrow{\alpha \mapsto \tau_\alpha}](\tau_{\alpha_i} : K_{\alpha_i}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$
Thus, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \tau_{\alpha_i} : K_{\alpha_i}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$

If $\beta \neq \alpha_i$, then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = \beta$, and because $C,\overrightarrow{\alpha \mapsto \tau_\alpha} \vdash \tau : K$
By (K-TVAR) rule, $C_1,\overrightarrow{\alpha \mapsto \tau_\alpha},\tau : K \vdash \tau : K,\ \ C = C_1,\tau : K$
$\Rightarrow C_1,\tau : K \vdash \tau : K$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]C_1,\tau : K \vdash \tau : K$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]C_1,[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K \vdash \tau : K$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1,\tau : K) \vdash \tau : K$

thus $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau : K$

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

3. $\tau = B_1 \,\text{bop}\, B_2$
then by (K-BOP) rule, $\dot{C}, \overrightarrow{\alpha \mapsto \tau_\alpha} \vdash B_1 \,\text{bop}\, B_2 : \text{bool} \Rightarrow \dot{C} \vdash B_1 \,\text{bop}\, B_2 : \text{bool}$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau = B_1 \,\text{bop}\, B_2$,
$\dot{C} \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]B_1 \,\text{bop}\, B_2 : \text{bool}$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C} \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]B_1 \,\text{bop}\, B_2 : \text{bool}$

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

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

6. $\tau = \forall\beta : K; B.\tau_0$
By (K-ALL) rule, $C, \overrightarrow{\alpha : K_\alpha} \vdash \forall\beta : K; B.\tau_0 :\overset{\phi i}{\text{type}}$, and
$C, \beta : K, \overrightarrow{\alpha : K_\alpha} \vdash B : \text{bool}$, $C, \beta : K, \overrightarrow{\alpha : K_\alpha} \vdash \tau_0 :\overset{\phi i}{\text{type}}$
By induction,

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

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

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

6.2.4 LEMMA [TERM SUBSTITUTION]

If $C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e : \tau$, $(C = \Psi; \Phi; \Delta; \Gamma; B; \text{limit}_C)$ and $z_i \notin \Gamma$, and $\dot{C} \vdash s_{x_i} : \tau_{x_i}$, $C_{y_i} \vdash s_{y_i} : \tau_{y_i}$, $C_{z_i} \vdash s_{z_i} : \tau_{z_i}$,$C \vdash \tau_{\alpha_i} : K_{\alpha_i}$, ($\tau_{y_i}$ and τ_{z_i} are linear type; τ_{x_i} are nonlinear type), then $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$, $([s] = [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x}, \overrightarrow{y \mapsto s_y}, \overrightarrow{z \mapsto s_z}])$

Proof by induction on the type derivation.

1. $e = x_0$, where $x_0 \neq y_i, z_i$
If $x_0 = x_i$, then $\tau = \tau_{x_i}$ and $\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash x_0 : \tau_{x_i}$
$[s]e = s_{x_i}$ and $\dot{C} \vdash s_{x_i} : \tau_{x_i}$,
By Lemma 6.2.1.(1), $\dot{C}_0, s_{x_i} : \tau_{x_i} \vdash s_{x_i} : \tau_{x_i}$
By Lemma 6.2.1.(1) again, $\dot{C}_0, s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i} \vdash s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$
then $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C}_0, [\overrightarrow{\alpha \mapsto \tau_\alpha}](s_{x_i} : \tau_{x_i}) \vdash s_{x_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_{x_i}$,

Thus, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C} \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

If $x_0 \neq x_i$, then $\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash x_0 : \tau$
$[s]e = x_0$
Thus, $\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash [s]e : \tau$
By Lemma 6.2.1.(1), $\dot{C}_0, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, x_0 : \tau \vdash [s]e : \tau$
Writing as: $\dot{C}_0, x_0 : \tau \vdash [s]e : \tau$,
By Lemma 6.2.1.(1) again, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C}_0, x_0 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$,
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}_0, x_0 : \tau) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$,
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C} \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

2. $e = y_0$, where $y_0 \neq x_i$

If $y_0 = y_i$, then $\dot{C}, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, y_i : \tau_{y_i} \vdash e : \tau_{y_i}$. $\quad \tau = \tau_{y_i}$
$[s]e = s_{y_i}$ and $C_{y_i} \vdash s_{y_i} : \tau_{y_i}$,
thus $C_{y_i} \vdash [s]e : \tau$
Because $\dot{C} = \dot{C}_{y_i}$, $C_{y_i} = \dot{C}, C_{y_i}$
$\dot{C}, C_{y_i} \vdash [s]e : \tau$
By Lemma 6.2.1.(1), $\dot{C}_0, s_{y_i} : \tau \vdash [s]e : \tau$
By Lemma 6.2.1.(1) again, $\dot{C}_0, s_{y_i} : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C}_0, [\overrightarrow{\alpha \mapsto \tau_\alpha}](s_{y_i} : \tau) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}_0, s_{y_i} : \tau) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$
Thus, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}, C_{y_i}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

If $y_0 \neq y_i$, then $C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash e : \tau$
Because $z_i \notin \Gamma$, then $y_0 \neq z_i$
$[s]e = y_0$,
and $C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x} \vdash [s]e : \tau$
By Lemma 6.2.1.(1), $\dot{C}_0, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, y_0 : \tau \vdash [s]e : \tau$
$\Rightarrow \dot{C}_0, y_0 : \tau \vdash [s]e : \tau$,
By Lemma 6.2.1.(1) again, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C}_0, y_0 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$,
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](\dot{C}_0, y_0 : \tau) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$,
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]\ \dot{C} \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau$

3. $e = e_1 e_2$
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e_1 e_2 : \tau_2$
By Lemma (6.2.1.(2)), we have

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

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

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

4. $e = \lambda w : \tau \xrightarrow{\phi,\text{limit}} e_0$
By Lemma (6.2.1.(3)) and (T-EQ) rule, we have
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \lambda w : \tau_w \xrightarrow{\phi,\text{limit}_2} e_0 : \tau_w \xrightarrow{2} \tau_0$
and $C', w : \tau_w, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e_0 : \tau_0$, $\Phi; \Delta \vdash \tau_w : K$, (Here, $C' = \Psi, \Phi; \Delta; \Gamma; B; \text{limit}_2$)
For $\dot{C} \vdash s_{x_i} : \tau_{x_i}$, $C_{y_i} \vdash s_{y_i} : \tau_{y_i}$, $C_{z_i} \vdash s_{z_i} : \tau_{z_i}$,$C \vdash \tau_{\alpha_i} : K_{\alpha_i}$,
we use Limit-Change Lemma, then $\dot{C'} \vdash s_{x_i} : \tau_{x_i}$, $C'_{y_i} \vdash s_{y_i} : \tau_{y_i}$, $C'_{z_i} \vdash s_{z_i} : \tau_{z_i}$
and in $C \vdash \tau_{\alpha_i} : K_{\alpha_i}$,we need $\Phi; \Delta$, then $C' \vdash \tau_{\alpha_i} : K_{\alpha_i}$. (We also use limit_2 in $\dot{C}, C', C'_{y_i}, C'_{z_i}$)

If τ_w is nonlinear, then by the Weakening Lemma, $\dot{C'}, w : \tau_w \vdash s_{x_i} : \tau_{x_i}$, $C'_{y_i}, w : \tau_w \vdash s_{y_i} : \tau_{y_i}$,
$C'_{z_i}, w : \tau_w \vdash s_{z_i} : \tau_{z_i}$, $C', w : \tau_w \vdash \tau_{\alpha_i} : K_{\alpha_i}$. If τ_w is linear, weakening is not needed.
By induction, we have:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C', \overrightarrow{C'_{y_i}}), w : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w \vdash [s]e_0 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0$
By Type Substitution Lemma we know, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](\Phi; \Delta) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w : K$
By (T-ABS) rule, we obtain:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_w \xrightarrow{2} [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_0$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e : [\overrightarrow{\alpha \mapsto \tau_\alpha}](\tau_w \xrightarrow{2} \tau_0)$

5. $e = \text{pack}[\tau_1, e_0] \text{ as } \exists \beta : K; B'.\tau_2$
By Lemma (6.2.1.(14)) and (T-EQ) rule, we have
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \text{pack}[\tau_1, e_0] \text{ as } \exists \beta : K; B'.\tau_2 : \exists \beta : K; B'.\tau_2$, and

$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \tau_1 : K \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash \tau_1 : K,$
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash [\beta \mapsto \tau_1]B' \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash [\beta \mapsto \tau_1]B',$
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash e_0 : [\beta \mapsto \tau_1]\tau_2,$
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \exists\beta : K; B'.\tau_2 :\overset{\phi i}{\text{type}} \Rightarrow C, \overrightarrow{\alpha : K_\alpha} \vdash \exists\beta : K; B'.\tau_2 :\overset{\phi i}{\text{type}}$
By induction, we get:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e_0 : [\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]\tau_2$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [s]e_0 : [\beta \mapsto \tau_1]([\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2)$
By Type Substitution Lemma,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1 : K,$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}][\beta \mapsto \tau_1]B'$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\beta \mapsto [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1]([\overrightarrow{\alpha \mapsto \tau_\alpha}]B'),$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}](\exists\beta : K; B'.\tau_2) :\overset{\phi i}{\text{type}}$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash \exists\beta : K; ([\overrightarrow{\alpha \mapsto \tau_\alpha}]B').[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 :\overset{\phi i}{\text{type}}$
By Weakening Lemma,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1 : K,$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash [\beta \mapsto [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1]([\overrightarrow{\alpha \mapsto \tau_\alpha}]B'),$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash \exists\beta : K; ([\overrightarrow{\alpha \mapsto \tau_\alpha}]B').[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 :\overset{\phi i}{\text{type}}$
By (T-PACK) rule,
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash \text{pack}[[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1, [s]e_0] \text{ as } \exists\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B'.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 :$
$\exists\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B'.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$
Therefore, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash$
$[s](\text{pack}[\tau_1, e_0] \text{ as } \exists\beta : K; B'.\tau_2) : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\exists\beta : K; B'.\tau_2$

6. $e = \text{unpack}\,\beta, w = e_1 \,\text{in}\, e_2$
By Lemma (6.2.1.(13)) and (T-EQ) rule, we get
$C, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y : \tau_y} \vdash \text{unpack}\,\beta, w = e_1 \,\text{in}\, e_2 : \tau_2$, and
$C_1, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y' : \tau_{y'}} \vdash e_1 : \exists\beta : K; B'.\tau_1, \;\; C \vdash \tau_2 : K_2$
$C_2, \overrightarrow{\alpha : K_\alpha}, \overrightarrow{x : \tau_x}, \overrightarrow{y'' : \tau_{y''}}, \beta : K, w : \tau_1, B' \vdash e_2 : \tau_2$
By Type Substitution Lemma, $[\overrightarrow{\alpha \mapsto \tau_\alpha}]C \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 : K_2$
By Weakening Lemma, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i'}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2 : K_2$
By induction, we obtain:
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \overrightarrow{C_{y_i'}}) \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x}, \overrightarrow{y' \mapsto s_{y'}}, \overrightarrow{z \mapsto s_z}, , \overrightarrow{y'' \mapsto s_{y''}}]e_1 : [\overrightarrow{\alpha \mapsto \tau_\alpha}](\exists\beta :$
$K; B'.\tau_1)$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_1, \overrightarrow{C_{y_i'}}) \vdash [s]e_1 : \exists\beta : K; [\overrightarrow{\alpha \mapsto \tau_\alpha}]B'.[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}](C_2, \overrightarrow{C_{y_i''}}, \beta : K, w : \tau_1, B') \vdash [\overrightarrow{\alpha \mapsto \tau_\alpha}, \overrightarrow{x \mapsto s_x}, \overrightarrow{y'' \mapsto s_{y''}}, \overrightarrow{z \mapsto s_z}, , \overrightarrow{y' \mapsto s_{y'}}]e_2 :$
$[\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$
$\Rightarrow [\overrightarrow{\alpha \mapsto \tau_\alpha}](C_2, \overrightarrow{C_{y_i''}}), \beta : K, w : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_1, [\overrightarrow{\alpha \mapsto \tau_\alpha}]B' \vdash [s]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$
By (T-UNPACK) rule, $[\overrightarrow{\alpha \mapsto \tau_\alpha}](C, \overrightarrow{C_{y_i}}) \vdash$
$\text{unpack}\,\beta, w = [s]e_1 \,\text{in}\, [s]e_2 : [\overrightarrow{\alpha \mapsto \tau_\alpha}]\tau_2$

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

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

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

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

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

By induction,

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

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

By Type Substitution Lemma,

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

By Weakening Lemma,

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

By (T-FIX) rule,

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

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

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

6.2.5 THEOREM [PRESERVATION]

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

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

1. $Case\, T - VAR$: $e = x$

There are no evaluation rules for variables

2. $Case\, T - ABS$: $e = \lambda x : \tau_1 \overset{\phi,\text{limit}}{\longrightarrow} e_0$

There are no evaluation rules for e (e is value)

3. $Case\, T - TABS$: $e = \Lambda\alpha : K; B.v$

There are no evaluation rules for e (e is value)

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

($E - APPABS1$)
$e_1 = \lambda x : \tau_x \xrightarrow{\phi, I} e_{12} \qquad e_2 = v_2 \qquad e' = \mathrm{coerce}([x \mapsto v_2]e_{12})$
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, $\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \mathrm{limit}_C \vdash e_2 : \tau_x$
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 ($I < \mathrm{limit}_C$) or ($\mathrm{limit}_C = \infty$, $\Phi_e; \Delta \vdash \tau' :\overset{\phi' 0}{\mathrm{type}}$),
by (T-COERCE) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash \mathrm{coerce}([x \mapsto v_2]e_{12}) : \tau'$
By (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

($E - APPABS2$)
$e_1 = \lambda x : \tau_x \xrightarrow{\phi, \infty} e_{12} \qquad e_2 = v_2 \qquad e' = [x \mapsto v_2]e_{12}$
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, $\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \mathrm{limit}_C \vdash e_2 : \tau_x$
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 ($\mathrm{limit}_C = \mathrm{limit}_f = \infty$),
By (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \mathrm{limit}_C \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

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

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 α doesn't appear in $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}$, and by (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi_e'; \Phi_{spare}, \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi_e' = \Psi_e$; $\Phi_e' = \Phi_e$; $M' = M$

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

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

$\Psi_e'; \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash \phi\langle\overrightarrow{e'}\rangle : \phi\langle\overrightarrow{\tau'}\rangle$
By (T-EQ), we obtain $\Psi_e'; \Phi_e'; \Delta; \Gamma; B; \text{limit} \vdash \phi\langle\overrightarrow{e'}\rangle : \tau$
Using (T-MEM) rule,

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

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

$(E - APPPROJECT)$
$e_a = \phi\langle v_1, \cdots, v_n\rangle$, $\Psi_{e_a}; \Phi_{e_a}; \Delta; \Gamma_{e_a}; B; \text{limit} \vdash \phi\langle v_1, \cdots, v_n\rangle : \phi\langle\tau'_1, \cdots, \tau'_n\rangle$,
$e' = [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e_b$
By Lemma 6.2.1.(6), there is $\phi\langle\tau''_1, \cdots, \tau''_n\rangle \equiv \phi\langle\tau'_1, \cdots, \tau'_n\rangle$
and $\forall i.(\Psi_{e_{a_i}}; \Phi_{e_{a_i}}; \Delta; \Gamma_{e_{a_i}}; B; \text{limit} \vdash v_i : \tau''_i)$
By (T-EQ), $\forall i.(\Psi_{e_{a_i}}; \Phi_{e_{a_i}}; \Delta; \Gamma_{e_{a_i}}; B; \text{limit} \vdash v_i : \tau'_i)$
Because $\Psi_{e_b}; \Phi_{e_b}; \Delta; \Gamma_{e_b}, \overrightarrow{x : \tau'}; B; \text{limit} \vdash e_b : \tau'_b$
By substitution lemma, we obtain
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash [x_1 \mapsto v_1, \cdots, x_n \mapsto v_n]e_b : \tau'_b$
By (T-EQ), $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
Thus, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e$; $\Phi'_e = \Phi_e$; $M' = M$

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

$(E - LOAD)$
$e_{\text{ptr}} = i$, $e_{\text{Has}} = \text{fact}$, and $e' =^\wedge \langle M(i), \text{fact}\rangle$
$\varnothing; \dot{\Phi}_{e_{\text{ptr}}}; \Delta; \dot{\Gamma}_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$
$\{i \mapsto \tau'\}; \dot{\Phi}_{e_{\text{Has}}}; \Delta; \dot{\Gamma}_{e_{\text{Has}}}; B; \text{limit} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$
$\{i \mapsto \tau'\}; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e :^\wedge \langle\tau', \text{Has}(I, \tau')\rangle$
By (T-MEM) rule and Weakening Lemma,
$\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; \text{true}; \text{limit} \vdash M(i) : \tau'$
$\{i \mapsto \tau'\}; \dot{\Phi}_{e_{\text{Has}}}; \Delta; \dot{\Gamma}_{e_{\text{Has}}}; B; \text{limit} \vdash \text{fact} : \text{Has}(I, \tau')$
By (T-TUPLE) rule,

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

$(E - LOAD1)\quad e_{\text{ptr}} \rightarrow e'_{\text{ptr}}$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \text{load}(e_{\text{ptr}}, e_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau')\rangle)$
then $\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M(i) : \Psi(i))$
Thus, by Lemma 6.2.1.(8) and (T-EQ) rule,
$\Psi_{e_{\text{ptr}}}; \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e_{\text{ptr}} : \text{Int}(I)$
$\Rightarrow \Psi'_{spare}, \Psi_{e_{\text{ptr}}}; \Phi'_{spare}, \Phi_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M, e_{\text{ptr}} : \text{Int}(I))$
$(\Psi'_{spare} = \Psi_{spare}, \Psi_{e_{\text{Has}}};\ \Phi'_{spare} = \Phi_{spare}, \Phi_{e_{\text{Has}}})$
By induction, we obtain $\Psi'_{e_{\text{ptr}}}; \Phi'_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash e'_{\text{ptr}} : \text{Int}(I)\quad (\Phi_{e_{\text{ptr}}} \dot{\supseteq} \Phi_{e_{\text{ptr}}})$
and $\Psi'_{spare}, \Psi'_{e_{\text{ptr}}}; \Phi'_{spare}, \Phi'_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M', e'_{\text{ptr}} : \text{Int}(I))$
$\Rightarrow\ \Psi_{spare}, \Psi_{e_{\text{Has}}}, \Psi'_{e_{\text{ptr}}}; \Phi_{spare}, \Phi_{e_{\text{Has}}}, \Phi'_{e_{\text{ptr}}}; \Delta; \Gamma_{e_{\text{ptr}}}; B; \text{limit} \vdash (M', e'_{\text{ptr}} : \text{Int}(I))$
(Here, $\forall i \in dom(\Psi').(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi'(i))))$
We still have $\Psi_{e_{\text{Has}}}; \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e_{\text{Has}} : \text{Has}(I, \tau')$
Given $C = \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}$ and $C = C_{ptr}, C_{Has}$ and $C'_{ptr} \dot{\supseteq} C_{ptr}$, by the split subset lemma, we can find a $C'_{ptr}, C'_{Has} = \Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} = C' \dot{\supseteq} C$ so that $C'_{Has} \supseteq C_{Has}$. We then use weakening to show $C'_{Has} \vdash e_{Has} : \text{Has}(I, \tau')$. Then using the (T-LOAD) rule, we have:
$\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(e'_{\text{ptr}}, e_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau')\rangle$
By (T-EQ), we obtain $\Psi'_e; \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash \text{load}(e'_{\text{ptr}}, e_{\text{Has}}) : \tau$
BY (T-MEM) rule,
$\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', \text{load}(e'_{\text{ptr}}, e_{\text{Has}}) : \tau)$

$(E - LOAD2)\quad e_{\text{Has}} \rightarrow e'_{\text{Has}}$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, \text{load}(v_{\text{ptr}}, e'_{\text{Has}}) :^\wedge \langle \tau', \text{Has}(I, \tau')\rangle)$
then $\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}; \varnothing; \varnothing; true; \infty \vdash M(i) : \Psi(i))$
Thus, by Lemma 6.2.1.(8) and (T-EQ) rule,
$\Psi_{e_{\text{Has}}}; \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e_{e_{\text{Has}}} : \text{Has}(I, \tau')$
$\Rightarrow \Psi'_{spare}, \Psi_{e_{\text{Has}}}; \Phi'_{spare}, \Phi_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash (M, e_{\text{Has}} : \text{Has}(I, \tau'))$
$(\Psi'_{spare} = \Psi_{spare}, \Psi_{v_{\text{ptr}}};\ \Phi'_{spare} = \Phi_{spare}, \Phi_{v_{\text{ptr}}})$
By induction, we obtain $\Psi'_{e_{\text{Has}}}; \Phi'_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash e'_{\text{Has}} : \text{Has}(I, \tau')$ $(\Phi_{e_{\text{Has}}} \dot{\supseteq} \Phi_{e_{\text{Has}}})$
and $\Psi'_{spare}, \Psi'_{e_{\text{Has}}}; \Phi'_{spare}, \Phi'_{e_{\text{Has}}}; \Delta; \Gamma_{e_{\text{Has}}}; B; \text{limit} \vdash (M', e'_{\text{Has}} : \text{Has}(I, \tau'))$

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

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

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

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

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

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

BY (T-MEM) rule,

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

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

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

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

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

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

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

$(E - STORE)$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

By induction, we obtain $\Psi'_{e_{\rm ptr}}; \Phi'_{e_{\rm ptr}}; \Delta; \Gamma_{e_{\rm ptr}}; B; \text{limit} \vdash e'_{\rm ptr} : \text{Int}(I)$ $(\Phi_{e_{\rm ptr}} \dot{\supseteq} \Phi_{e_{\rm ptr}})$
and $\Psi'_{spare}, \Psi'_{e_{\rm ptr}}; \Phi'_{spare}, \Phi'_{e_{\rm ptr}}; \Delta; \Gamma_{e_{\rm ptr}}; B; \text{limit} \vdash (M', e'_{\rm ptr} : \text{Int}(I))$
$\Rightarrow \Psi_{spare}, \Psi_{e_{\rm Has}}, \Psi'_{e_{\rm ptr}}, \Psi_{e_{\rm v}}; \Phi_{spare}, \Phi_{e_{\rm Has}}, \Phi'_{e_{\rm ptr}}, \Phi_{e_{\rm v}}; \Delta; \Gamma_{e_{\rm ptr}}; B; \text{limit} \vdash (M', e'_{\rm ptr} : \text{Int}(I))$
(Here, $\forall i \in dom(\Psi').(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi'(i))))$
We still have $\Psi_{e_{\rm has}}; \Phi_{e_{\rm has}}; \Delta; \Gamma_{e_{\rm has}}; B; \text{limit} \vdash e_{\rm has} : \text{Has}(I, \tau_1)$
$\Psi_{e_{\rm v}}; \Phi_{e_{\rm v}}; \Delta; \Gamma_{e_{\rm v}}; B; \text{limit} \vdash e_{\rm v} : \tau_2$

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

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 $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \text{fix}\, x : \tau_x.v : \tau$
$\Psi_e; \Phi_e; \Delta; \Gamma, x : \tau_x; B; \text{limit}_C \vdash v : \tau$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_C \vdash \tau :_{\phi i}^{\rm type}$

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

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

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

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

$(E - UNPACK)$
$e_1 = \text{pack}[\tau_0, v]\,\text{as}\,\exists\beta : K'; \tau_B.\tau'_1$ $e' = [\alpha \mapsto \tau_0, x \mapsto v]e_2$
$\Psi_{e_1}; \Phi_{e_1}; \Delta; \Gamma_{e_1}; B; \text{limit} \vdash \text{pack}[\tau_0, v]\,\text{as}\,\exists\beta : K'; \tau_B.\tau'_1 : \exists\alpha : K; B'.\tau_1$
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 α,
$[\alpha \mapsto \tau_0](\Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}, x : \tau_1; B, B'; \text{limit}) \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$
$\Rightarrow \Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B, [\alpha \mapsto \tau_0]B'; \text{limit}, x : [\alpha \mapsto \tau_0]\tau_1 \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$
$\Rightarrow \Psi_{e_2}; \Phi_{e_2}; \Delta; \Gamma_{e_2}; B; \text{limit}, x : [\alpha \mapsto \tau_0]\tau_1 \vdash [\alpha \mapsto \tau_0]e_2 : [\alpha \mapsto \tau_0]\tau_2$
by Term Substitution Lemma for v,
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash [\alpha \mapsto \tau_0, x \mapsto v]e_2 : [\alpha \mapsto \tau_0]\tau_2$
$\Rightarrow \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash [\alpha \mapsto \tau_0, x \mapsto v]e_2 : \tau_2$
Thus $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau_2$
By (T-EQ) tule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$

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

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

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

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

$(E - NEWFUN)$

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

By (T-FACT2) rule, we have $\varnothing; \dot{\Phi}_e, F^J \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Gen}(F^J, 0)$

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

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

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

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

By (T-PACK) rule, we get

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

Thus, $\varnothing; \Phi_e, F^J \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \exists\alpha : \text{int} \to J.\text{Gen}(\alpha, 0)$

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

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

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

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

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

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

15. $Case\, T - DEFINEFUN$: $e = \text{define_fun}(e_0, \tau_a)$.

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

By Lemma 6.2.1.(17), then $\wedge\langle \text{Gen}(\tau_f, I+1), \text{Eq}(\tau_f I, \tau_a), \text{InDomain}(I, \tau_f)\rangle \equiv \tau$

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

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

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

$(E - DEFINEFUN)$

$e_0 = \text{fact}$, then $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e_0 : \text{Gen}(\tau_f, I)$

and $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{define_fun}(e_0, \tau_a) : \tau$

$\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \tau_f : \text{int} \to J$

$\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \tau_a : J$

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

By (T-FACT2) rule, $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Gen}(\tau_f, I+1)$

By (T-FACT3) rule, $\varnothing; \dot{\Phi}_e, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_f I, \tau_a)$

By weakening lemma, $\varnothing; \dot{\Phi}_e, \tau_f \dot{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_f I, \tau_a)$

By (T-FACT4) rule, $\varnothing; \dot{\Phi}_e, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{InDomain}(I, \tau_f)$

By weakening lemma, $\varnothing; \dot{\Phi}_e, \tau_f \dot{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{InDomain}(I, \tau_f)$

By (T-TUPLE) rule, we obtain $\varnothing; \dot{\Phi}_e, \tau_f \overset{\wedge}{\mapsto} \text{fun}'; \Delta; \dot{\Gamma}; B; \text{limit} \vdash$

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

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

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

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

$(E - INDOMAIN)$
$e_1 = \text{fact}_1$, $e_2 = \text{fact}_2$ then
$\varnothing; \dot{\Phi}_{e_1}, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_{e_1}; B; \text{limit} \vdash e_1 : \text{InDomain}(I_1, \tau_f)$
$\varnothing; \dot{\Phi}_{e_2}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_{e_2}; B; \text{limit} \vdash e_2 : \text{Gen}(\tau_f, I_2)$
$\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e : \tau$
$e' =^\wedge \langle \text{know}(0 \le I_1 \wedge I_1 < I_2), \text{fact}\rangle$
By (T-FACT2) rule, we get $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{fact} : \text{Gen}(\tau_f, I_2)$

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

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

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

$\varnothing; \dot{\Phi}_{e_0}, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \cdot\langle\rangle : [\alpha \mapsto \text{true}] \cdot \langle\rangle$,

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

By (T-PACK) rule, we obtain:

$\varnothing; \dot{\Phi}_{e_0}, \tau_f \dot{\mapsto} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{pack}[\text{true}, \cdot\langle\rangle]\,\text{as}\,\exists\alpha : \text{bool}; 0 \leq I_1 \wedge I_1 < I_2 . \cdot \langle\rangle : \exists\alpha : \text{bool}; 0 \leq I_1 \wedge I_1 < I_2 . \cdot \langle\rangle$

By using abbreviations, $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\dot{\mapsto}} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash \text{know}(0 \leq I_1 \wedge I_1 < I_2) : \text{Know}(0 \leq I_1 \wedge I_1 < I_2)$

By (T-TUPLE) rule, we obtain:

$\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\dot{\mapsto}} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash^{\wedge} \langle \text{know}(0 \leq I_1 \wedge I_1 < I_2), \text{fact}\rangle :^{\wedge} \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle$

Then, $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\dot{\mapsto}} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e' :^{\wedge} \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f, I_2)\rangle$

By (T-EQ) rule, $\varnothing; \dot{\Phi}_{e_0}, \tau_f \overset{\wedge}{\dot{\mapsto}} \text{fun}; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e' : \tau$

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

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

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

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

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

18. $Case\, T - MAKEEQ$: $e = \text{make_eq}(\tau_0)$

$\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{make_eq}(\tau_0) : \tau$

By Lemma 6.2.1.(20), then $\text{Eq}(\tau_0, \tau_0) \equiv \tau$ and $\Phi_e; \Delta \vdash \tau_0 : K$

$(E - MAKEEQ)$

$e' = \text{fact}$

By (T-FACT3) rule, $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash \text{fact} : \text{Eq}(\tau_0, \tau_0)$

Then $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}_e; B; \text{limit} \vdash e' : \text{Eq}(\tau_0, \tau_0)$

Thus, $\varnothing; \dot{\Phi}_e; \Delta; \dot{\Gamma}; B; \text{limit} \vdash e' : \tau$

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

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

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

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

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

19. $Case\, T - APPLYEQ$: $e = \text{apply_eq}(\tau_f, e_1, e_2)$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{apply_eq}(\tau_f, e_1, e_2) : \tau$
By Lemma 6.2.1.(21), then $\tau_f \tau_b \equiv \tau$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{apply_eq}(\tau_f, e_1, e_2) : \tau_f \tau_b$
$\Phi_e; \Delta \vdash \tau_f : K \to J \quad \Phi_e; \Delta \vdash \tau_a : K \quad \Phi_e; \Delta \vdash \tau_b : K$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_1 : \text{Eq}(\tau_a, \tau_b) \quad \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e_2 : \tau_f \tau_a$

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

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

$(E - CASE)$
$e_0 = \text{union}(\tau_b, \tau'_1, \tau'_2, v), \quad e' = v$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \text{union}(\tau_b, \tau'_1, \tau'_2, v) : \text{Union}(b, \tau_1, \tau_2)$
By Lemma 6.2.1.(23), then $\tau_1 \equiv \tau'_1,\ \tau_2 \equiv \tau'_2,\ \tau_b \equiv b$ and
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau'_1 : K,\ \Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash \tau'_2 : K$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash v : \tau'_i \ \ (i = \begin{matrix} 1\,if\,\tau_b \\ 2\,if\,\neg\tau_b \end{matrix})$
Thus, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau'_i \ \ (i = \begin{matrix} 1\,if\,b \\ 2\,if\,\neg b \end{matrix})$
By (T-EQ) rule, $\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash e' : \tau$
Because $\Psi_{spare}, \Psi_e; \Phi_{spare}, \Phi_e; \Delta; \Gamma; B; \text{limit} \vdash (M, e : \tau)$
And we don't change M, then $M' = M$
$\forall i \in dom(\Psi).(\varnothing; \dot{\Phi}'; \varnothing; \varnothing; true; \infty \vdash M'(i) : \Psi(i))$
By (T-MEM) rule, $\Psi_{spare}, \Psi'_e; \Phi_{spare}, \Phi'_e; \Delta; \Gamma; B; \text{limit} \vdash (M', e' : \tau)$
$\Psi'_e = \Psi_e;\ \Phi'_e = \Phi_e;\ M' = M$

21. $Case\,T - COERCE$: $\quad e = \text{coerce}(e_0)$

$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e_0) : \tau$
By Lemma 6.2.1.(24), then there is $\tau' \equiv \tau$
$\Psi_e; \Phi_e; \Delta; \Gamma; B; \text{limit}_1 \vdash \text{coerce}(e_0) : \tau'$
and $\Psi_e; \Phi_e; \Delta; \Gamma; B; I_2 \vdash e_0 : \tau'$

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

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

6.3 Progress

Rephrase the type equivalence rules as parallel reduction rules:

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

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

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

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

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

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

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

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

If $C \vdash \tau_a \Rightarrow \tau_b$, and $C \vdash \tau_a \Rightarrow \tau_c$, then there is some τ_d so that $C \vdash \tau_b \Rightarrow \tau_d$ and $C \vdash \tau_c \Rightarrow \tau_d$.

Proof by induction on the sum of the sizes of the derivations of $\tau_a \Rightarrow \tau_b$ and $\tau_a \Rightarrow \tau_c$.

1. case $\tau_a \overset{A}{\Rightarrow} \tau_b$. Then $\tau_b = \tau_a$, so choose $\tau_d = \tau_c$.
2. case $\tau_a \overset{B}{\Rightarrow} \tau_b$ and $\tau_a \overset{B}{\Rightarrow} \tau_c$. The proof for this is standard.
3. case $\tau_a \overset{G}{\Rightarrow} \tau_b$ and $\tau_a \overset{B}{\Rightarrow} \tau_c$. The proof for this is standard.
4. case $\tau_a \overset{G}{\Rightarrow} \tau_b$ and $\tau_a \overset{G}{\Rightarrow} \tau_c$. The proof for this is standard.
5. case $\tau_a = \mathrm{Eq}(\tau_{a1}, \tau_{a2}) \overset{F}{\Rightarrow} \tau_b = \mathrm{Eq}(\tau_{a2}, \tau_{a1})$. If $\tau_a \overset{F}{\Rightarrow} \tau_c$, then choose $\tau_d = \tau_b = \tau_c$. If $\tau_a = \mathrm{Eq}(\tau_{a1}, \tau_{a2}) \overset{G}{\Rightarrow} \tau_c = \mathrm{Eq}(\tau_{c1}, \tau_{c2})$, then choose $\tau_d = \mathrm{Eq}(\tau_{c2}, \tau_{c1})$.
6. case $\tau_a = I_a \overset{D}{\Rightarrow} \tau_b = I_b$ and $\tau_a = I_a \overset{D}{\Rightarrow} \tau_c = I_c$. Then $I_a \doteq I_b \doteq I_c$, so pick $\tau_d = I_a$.
7. case $\tau_a = B_a \overset{E}{\Rightarrow} \tau_b = B_b$ and $\tau_a = B_a \overset{E}{\Rightarrow} \tau_c = B_c$. Then $B_a \doteq B_b \doteq B_c$, so pick $\tau_d = B_a$.
8. case $\tau_a = \mathrm{F}^K I_a \overset{C}{\Rightarrow} \tau_b = \delta(i_b)$. If $\tau_a \overset{C}{\Rightarrow} \tau_c = \delta(i_c)$, then $i_b = i_c$, so choose $\tau_d = \tau_b = \tau_c$. If $\tau_a = \mathrm{F}^K I_a \overset{G}{\Rightarrow} \tau_c = \mathrm{F}^K I_c$, then $i_b \doteq I_a \doteq I_c$, so choose $\tau_d = \tau_b$.

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

6.3.2 LEMMA [CONFLUENCE FOR TYPES]

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

6.3.3 CORROLARY [CONFLUENCE FOR EQUIVALENT TYPES]

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

6.3.4 LEMMA [SHAPE PRESERVATION]

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

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

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

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

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

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

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

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 τ has a value shape, then τ' has the same value shape (proof by case analysis on $\tau \Rightarrow \tau'$). Combining this with confluence, we conclude that if $C \vdash \tau_b \equiv \tau_c$, and τ_b and τ_c have value shapes, there is some τ_d so that $C \vdash \tau_b \stackrel{*}{\Rightarrow} \tau_d$ and $C \vdash \tau_c \stackrel{*}{\Rightarrow} \tau_d$, and τ_d has the same shape as τ_b and τ_c, which implies that τ_b and τ_c have the same shape.

6.3.5 LEMMA [CANONICAL FORMS]

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

1. If $C \vdash v : \tau_1 \stackrel{\phi,limit}{\longrightarrow} \tau_2$, then $v = \lambda x : \tau_1 \stackrel{\phi,limit}{\longrightarrow} e$.
2. If $C \vdash v : \langle\tau_1, \ldots \tau_n\rangle$, then $v = \langle v_1, \ldots, v_n\rangle$.
3. If $C \vdash v : \forall\alpha : K; B.\tau$, then $v = \Lambda\alpha : K; B.v_0$.
4. If $C \vdash v : \exists\alpha : K; B.\tau_2$, then $v = \text{pack}[\tau_1, v_0] \,\text{as}\, \exists\alpha : K; B.\tau_2$.
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 $C \vdash v : \mathrm{Has}(I, \tau)$ or $C \vdash v : \mathrm{Gen}(\tau, I)$ or $C \vdash v : \mathrm{Eq}(\tau_1, \tau_2)$ or $C \vdash v : \mathrm{InDomain}(I, \tau)$, then $v = \mathrm{fact}$.

Proof:

1. Only two type checking rules can prove $C \vdash v : \tau_1 \overset{\phi,limit}{\longrightarrow} \tau_2$: (T-EQ) and (T-ABS).

For (T-EQ) rule, there is some $\tau' \equiv \tau_1 \overset{\phi,limit}{\longrightarrow} \tau_2$. By shape preservation, $\tau' = \tau_1' \overset{\phi,limit}{\longrightarrow} \tau_2'$. Use induction.

By (T-ABS) rule, we know $v = \lambda x : \tau_1 \overset{\phi,limit}{\longrightarrow} e$ is an abstraction.

2. The similar proof for the other cases.

6.3.6 THEOREM[PROGRESS]

If e is a closed, well-typed expression, ($C \vdash e : \tau$ for some τ and $C = \Psi; \Phi; \varnothing; \varnothing; B; \mathrm{limit}$), then either e is a value or else there is some e' for $e \to e'$. Prove by induction on the derivation of $C \vdash e : \tau$.

Proof :

First, if $e = E[e_0]$ where e_0 is not a value, then by inspection of the type checking rules, $\Psi_0; \Phi_0; \varnothing; \varnothing; B_0; \mathrm{limit}_0 \vdash e_0 : \tau_0$, so by induction, $e_0 \to e_0'$, so $e = E[e_0] \to E[e_0']$. Second, the (T-EQ) case is an easy induction. For all other cases where e is not a value:

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

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

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

4. $e = \mathrm{unpack}\, \alpha, x = v_1 \,\mathrm{in}\, e_2$, where $(T{-}UNPACK)\dfrac{\begin{array}{c} C_1 \vdash v_1 : \exists \alpha : K; B.\tau_1 \qquad C_1, C_2 \vdash \tau_2 : K_2 \\ C_2, \alpha : K, x : \tau_1, B \vdash e_2 : \tau_2 \end{array}}{C_1, C_2 \vdash \mathrm{unpack}\, \alpha, x = v_1 \,\mathrm{in}\, e_2 : \tau_2}$.
   By canonical forms, $v_1 = \mathrm{pack}[\tau_1, v] \,\mathrm{as}\, \exists \alpha : K; B.\tau_2$, so e steps by (E-UNPACK).

5. $e = \text{in_domain}(I_1, I_2, v_1, v_2)$, where $(T{-}INDOMAIN)\frac{C_1, C_2 \vdash I_1 : \text{int} \quad C_1, C_2 \vdash I_2 : \text{int} \quad C_1 \vdash v_1 : \text{InDomain}(I_1, \tau_f) \quad C_2 \vdash v_2 : \text{Gen}(\tau_f, I}{C_1, C_2 \vdash \text{in_domain}(I_1, I_2, v_1, v_2) : \wedge \langle \text{Know}(0 \leq I_1 \wedge I_1 < I_2), \text{Gen}(\tau_f}$.
   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. $e = \text{new_fun}(K)$, $e = \text{make_eq}(\tau)$, $e = \text{coerce}(v)$, $e = \text{fix}\, x : \tau.\text{v}$: these always step, so we don't even need to look at the type checking rules.

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

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 $(T{-}IFB)\frac{C = \Psi; \Phi; \Delta; \Gamma; B_C; I \quad C \vdash B : \text{bool} \quad C, B \vdash e_1 : \tau \quad C, \neg B \vdash e_2 : \tau}{C \vdash \text{if}\, B \,\text{then}\, e_1 \,\text{else}\, e_2 : \tau}$.
    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 $(T - LOAD)\frac{C_1 \vdash v_{\text{ptr}} : \text{Int}(I) \quad C_2 \vdash v_{\text{Has}} : \text{Has}(I, \tau)}{C_1, C_2 \vdash \text{load}(v_{\text{ptr}}, v_{\text{Has}}) : \wedge \langle \tau, \text{Has}(I, \tau) \rangle}$.
    By canonical forms, $v_{ptr} = i_{ptr}$, and $v_{Has} = \text{fact}$. By inversion, $C_1 \vdash i_{ptr} : Int(i_{ptr})$ and $C_2 = C_2', i_{has} \mapsto \tau \vdash \text{fact} : \text{Has}(i_{has}, \tau)$. Together, these imply $i_{ptr} = i_{has}$. Since $i_{has} \mapsto \tau \in \Psi$ and memory M is well-typed, $M(i)$ exists, so e steps by (E-LOAD).

12. $e = \text{store}(v_{\text{ptr}}, v_{\text{Has}}, v_{\text{v}})$, where $(T{-}STORE)\frac{C = C_1, C_2, C_3 = \Psi; \Phi; \Delta; \Gamma; B; \infty \quad C_2 \vdash v_{\text{Has}} : \text{Has}(I, \tau_1) \quad C_3 \vdash v_{\text{v}} : \tau_2 \quad C_1 \vdash v_{\text{ptr}} : \text{Int}(I) \quad C \vdash \tau_2 :^{\cdot 1} \text{type}}{C \vdash \text{store}(v_{\text{ptr}}, v_{\text{Has}}, v_{\text{v}}) : \text{Has}(I, \tau_2)}$.
    By canonical forms, $v_{ptr} = i_{ptr}$, and $v_{Has} = \text{fact}$, 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 $\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash e : \tau$, then there is some v so that $e \xrightarrow{*} v$.

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:

$\text{depth}(x) = 0$

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

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

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 Main Lemma: single-step size reduction

If $\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash e : \tau$, and $e \rightarrow e'$, (suppose $e \rightarrow e'$ using the main evaluation rules)

then one of these three must be true:

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

Proof :

1. $e = (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2$
and $e' = \text{coerce}([x \mapsto v_2]e_1)$, and $e'' = [x \mapsto v_2]e_1$
By Lemma 6.2.1.(2) and (T-EQ) rule, we know
$\Psi; \Phi; \varnothing; \varnothing; \text{true}; \text{limit}_C \vdash (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2 : \tau_b$, $(I = \text{limit}_f)$,
$\Psi_1; \Phi_1; \varnothing; \varnothing; \text{true}; \text{limit}_f \vdash e_1 : \tau_a \xrightarrow{f} \tau_b$, and $(\text{limit}_C > \text{limit}_f)$
$\Psi_2; \Phi_2; \varnothing; \varnothing; \text{true}; \text{limit}_C \vdash v_2 : \tau_a$
By Limit-Change Lemma, $\Psi_2; \Phi_2; \varnothing; \varnothing; \text{true}; \text{limit}_f \vdash v_2 : \tau_a$
By Term Substitution Lemma, we obtain:
$\Psi; \Phi; \varnothing; \varnothing; \text{true}; \text{limit}_f \vdash [x \mapsto v_2]e_1 : \tau_b$
$\Psi; \Phi; \varnothing; \varnothing; \text{true}; \text{limit}_f \vdash e'' : \tau_b$
Now MainLemma (1) 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. $e = \text{coerce}(v)$
$e' = v$
By Lemma 6.2.1.(24) and (T-EQ) rule, we know

$\Psi; \Phi; \varnothing; \varnothing; \text{true}; \text{limit}_1 \vdash \text{coerce}(v) : \tau$
and $\Psi; \Phi; \varnothing; \varnothing; \text{true}; I_2 \vdash v : \tau$
Because $\text{limit}_1 = i$, then $i > I_2$
Now MainLemma (1) will be true.

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

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

6.
$\text{size}(\text{load}(i, \text{fact})) = 1 + \text{size}(i) + \text{size}(\text{fact}) > \text{size}(\hat{}\,\langle M(i), \text{fact}\rangle) = 1$
$\text{size}(\text{store}(i, \text{fact}, v)) = 1 + \text{size}(i) + \text{size}(\text{fact}) + \text{size}(v) > \text{size}(\text{fact}) = 1$
$\text{size}((\Lambda\alpha : K; B.v)\tau) = 1 + \text{size}(v) > \text{size}(v) = 1$
$\text{size}(\text{if}\, e_1 \,\text{then}\, e_2 \,\text{else}\, e_3) = 1{+}\text{size}(e_1){+}\text{size}(e_2){+}\text{size}(e_3) > \text{size}(e_2) \,\text{or}\, \text{size}(e_2)$
$\text{size}(\text{unroll}(\text{roll}[\tau](v))) = 1 + 1 + \text{size}(v) > \text{size}(v)$
$\text{size}(\text{fix}\, x : \tau.v) = 1 + \text{size}(v) > \text{size}([x \mapsto \text{fix}\, x : \tau.v]v) = 1 \quad$ (By Lemma 2)
$\text{size}(\text{case}(b, \text{union}(b, \tau_1, \tau_2, v))) = 1{+}1{+}\text{size}(b){+}\text{size}(v){+}\text{size}(b) > \text{size}(v) = 1$
$\text{size}(\text{in_domain}(I_1, I_2, \text{fact}, \text{fact})) = 1{+}\text{size}(I_1){+}\text{size}(I_2){+}\text{size}(\text{fact}){+}\text{size}(\text{fact}) > \text{size}(e') = 1$
$\text{size}(\text{distinguish}(I_1, I_2, \text{fact}, \text{fact})) = \text{size}(I_1){+}\text{size}(I_2){+}\text{size}(\text{fact}){+}\text{size}(\text{fact}) > \text{size}(e') = 1$
$\text{size}(\text{make_eq}(\tau)) > \text{size}(\text{fact}) = 1$
$\text{size}(\text{apply_eq}(\tau, \text{fact}, v)) > \text{size}(v) = 1$
$\text{size}(\text{new_fun}(K)) > \text{size}(e') = 1$
$\text{size}(\text{discard_fun}(\text{fact})) > \text{size}(\cdot\langle\rangle) = 1$
$\text{size}(\text{define_fun}(\text{fact}, \tau)) > \text{size}(\hat{}\,\langle \text{fact}, \text{fact}, \text{fact}\rangle) = 1$

7.2 Theorem: strong normalization

If $\Psi; \Phi; \varnothing; \varnothing; \text{true}; i \vdash e : \tau$, then e must step to a value in a finite sequence of zero or more steps $e \to e_1 \to \cdots \to e_n \to v$. (Notation: $e \xrightarrow{*} v$).

Proof :

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

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

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

1. $e = (\lambda x : \tau \xrightarrow{\phi, I} e_1)v_2$

In Main Lemma case 1, we prove $\Psi; \Phi; \varnothing; \varnothing; \text{true}; I_2 \vdash e'' : \tau$, and $I_2 < i$

By induction, we know e'' must step to a value in a finite sequence of zero or more steps.

If e'' is not a value, then we apply (E-COERCE1) to $e' = \text{coerce}(e'')$;

If e'' is a value, then we apply (E-COERCE) to $e' = \text{coerce}(e'')$, we will get a value.

So e' must step to a value in a finite sequence of one or more steps.

And e must step to a value in a finite sequence of two or more steps.

2. $e = \text{coerce}(v)$

$\Psi; \Phi; \varnothing; \varnothing; \text{true}; I_2 \vdash e' : \tau$, and $I_2 < i$

By induction, we know e' must step to a value in a finite sequence of zero or more steps.

Thus, e must step to a value in a finite sequence of one or more steps.

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

4. $e = \text{unpack}\, \alpha, x = e_1 \,\text{in}\, e_2$

In these two cases, depth(e') < depth(e), then by induction, e' must step to a value in a finite sequence of zero or more steps.

Thus, e must step to a value in a finite sequence of one or more steps.

5. $e = i \mid b \mid \lambda x : \tau \xrightarrow{\phi, \text{limit}} e_0 \mid \Lambda\alpha : K; B.v \mid \text{fact}$

They are values. Thus e must step to a value in a finite sequence of zero.

6. If e is any expression in Main Lemma case 6, then depth(e') $\leq$ depth(e), and size(e') < size(e).

By induction, we know e' must step to a value in a finite sequence of zero or more steps.

Thus, e must step to a value in a finite sequence of one or more steps.

7. In the above cases, we only use the main evaluation rules, now we will discuss about the congruence rules.

For example:

$e = \text{store}(e_1, e_2, e_3)$

If e_1 is not value, by (E-STORE1) rule $\frac{e_1 \to e_1'}{\text{store}(e_1, e_2, e_3) \to \text{store}(e_1', e_2, e_3)}$,

and size(e_1) < size(e), by induction, e_1 must step to a value in a finite sequence of zero or more steps.

Then $e = \text{store}(e_1, e_2, e_3)$ must step to $\text{store}(v_1, e_2, e_3)$ in a finite steps.

If e_2 is not value, by (E-STORE2) rule $\frac{e_2 \to e_2'}{\text{store}(v_1,e_2,e_3) \to \text{store}(v_1,e_2',e_3)}$,

and size(e_2) < size(e), by induction, e_2 must step to a value in a finite sequence of zero or more steps.

Then $e = \text{store}(v_1, e_2, e_3)$ must step to $\text{store}(v_1, v_2, e_3)$ in a finite steps.

If e_3 is not value, by (E-STORE3) rule $\frac{e_3 \to e_3'}{\text{store}(v_1,v_2,e_3) \to \text{store}(v_1,v_2,e_3')}$,

and size(e_3) < size(e), by induction, e_3 must step to a value in a finite sequence of zero or more steps.

Then $e = \text{store}(v_1, v_2, e_3)$ must step to $\text{store}(v_1, v_2, v_3)$ in a finite steps.

For $e = \text{store}(v_1, v_2, v_3)$, in case 6 we have prove it must step to a value in a finite sequence of one or more steps.

Thus, we know $e = \text{store}(e_1, e_2, e_3)$ must step to a value in a finite sequence of one or more steps.

For the other cases, the proof is similar.

References

[1] Chris Hawblitzel, Heng Huang, Eric Krupski, and Edward Wei. Low-level linear memory management. In *submitted for publication*, 2002.

[2] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[3] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.