1. Full System

In this section we present the full type system for the core language of §3 of the main paper.

1.1 Formal Languages

**FRSC** Figure 1 shows the full syntax for the input language. The type language is the same as described in the main paper. The operational semantics, shown in Figure 2, is borrowed from Safe TypeScript [4], with certain simplifications since the language we are dealing with is simpler than the one used there. We use evaluation contexts $E$, with a left to right evaluation order.

**IRSC** Figure 3 shows the full syntax for the SSA transformed language. The reduction rules of the operational semantics for language IRSC are shown in Figure 4. We use evaluation contexts $E$, with a left to right evaluation order.

1.2 SSA Transformation

Section 3 of the main paper describes the SSA transformation from FRSC to IRSC. This section provides more details and extends the transformation to runtime configurations, to enable the statement and proof of our consistency theorem.

1.2.1 Static Transformation

Figure 5 includes some additional transformation rules that supplement the rules of Figure 3 of the main paper. The main program transformation judgment is:

$$P \mapsto P \mapsto \Delta$$

A global SSA environment $\Delta$ is the result of the translation of the entire program $P$ to $P$. In particular, in a program translation tree:

- each expression node introduces a single binding to the relevant SSA environment

  $$\delta \vdash e \mapsto e \quad \text{produces binding} \quad e \mapsto \delta$$

- each statement introduces two bindings, one for the input environment and one for the output (we use the notation $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$, respectively):

  $$\delta_0 \vdash s \mapsto u; \delta_1 \quad \text{produces bindings} \quad \lceil s \rceil \mapsto \delta_0 \quad \lfloor s \rfloor \mapsto \delta_1$$

We assume all AST nodes are uniquely identified.

1.2.2 Runtime Configuration Transformation

Figure 6 includes the rules for translating runtime configurations. The main judgment is of the form:

$$K; w \xrightarrow{\Delta} K; e$$

This assumes that the program containing expression (or body) $w$ was SSA-translated producing a global SSA environment $\Delta$. Rule S-EXP-RtCONF translates a term $w$ under a state $K$. This process gets factored into the translation of:
Syntax

Expressions  
\[ e ::= x \mid c \mid \text{this} \mid e.f \mid e.m(v) \mid \text{new } C(v) \mid <T>e \]

Statements  
\[ s ::= \text{var } x = e \mid e.f = e \mid x = e \mid \text{if}(e)\{s\} \text{ else } \{s\} \mid s; s \mid \text{skip} \]

Field Decl.  
\[ F ::= \cdot \mid \text{of}: T \mid \square f: T \mid F_1; F_2 \]

Method Body  
\[ B ::= s; \text{return } e \]

Expr. or Body  
\[ w ::= e \mid B \]

Method Decl.  
\[ M ::= \cdot \mid m(x: T) \{p\}: T \mid M_1; M_2 \]

Field Def.  
\[ \bar{F} ::= \cdot \mid f := v \mid \bar{F}_1; \bar{F}_2 \]

Method Def.  
\[ \bar{M} ::= \cdot \mid m(x: T) \{p\}: T\{B\} \mid \bar{M}_1; \bar{M}_2 \]

Class Def.  
\[ \bar{C} ::= \text{class } C \{p\} \text{ extends } R \{F, \bar{M}\} \]

Signature  
\[ S ::= \cdot \mid \bar{C}; S_1; S_2 \]

Program  
\[ P ::= S; B \]

Runtime Configuration

Evaluation Context  
\[ E ::= [ ] \mid E.f \mid E.m(v) \mid v.m(v, E, v) \mid \text{new } C(v, E, v) \mid <T>E \mid \text{var } x = E \mid E.f = e \mid v.f = E \mid x = E \mid \text{if}(E)\{s\} \text{ else } \{s\} \mid \text{return } E \mid E; s \mid E; \text{return } e \]

Runtime Conf.  
\[ R ::= K; s \]

State  
\[ K ::= S; L; X; H \]

Store  
\[ L ::= \cdot \mid x \mapsto v \mid L_1; L_2 \]

Value  
\[ v ::= l \mid c \]

Stack  
\[ X ::= \cdot \mid X; L, E \]

Heap  
\[ H ::= \cdot \mid l \mapsto o \mid H_1; H_2 \]

Object  
\[ O ::= \{\text{proto}: l; f: \bar{F}\} \mid \{\text{name}: C; \text{proto}: l; m: \bar{M}\} \]

Figure 1: FRSC: syntax and runtime configuration

- the signatures \( K.S \), which is straight-forward (same as in static translation),
- the heap \( K.H \), which is described in Figure 7, and
- term \( w \) under a local store \( K.L \) and a stack \( K.X \).
Operational Semantics for FRSC

The last part breaks down into rules that expose the structure of the stack. Rule S-STACK-EMP translates configurations involving an empty stack, which are delegated to the judgment \( L; w \triangleleft e \), and rule S-STACK-CONS separately translates the top of the stack and the rest of the stack frames, and then composes them into a single target expression.

Finally, judgments of the forms \( L; X; w \triangleleft e \) and \( L; X; E \triangleleft W \) translate expressions and statements under a local store \( L \). The rules here are similar to their static counterparts. The key difference stems from the fact that in IRSC variable are replaced with the respective values as soon as they come into scope. On the contrary, in FRSC variables are only instantiated with the matching (in the store) value when they get into an evaluation position. To wit, rule SR-VARREF performs the necessary substitution \( \theta \) on the translated variable, which we calculate though the meta-function toSubst, defined as follows:

\[
\text{toSubst} (\delta, L, H) = \begin{cases} 
\{[v/x] | x \mapsto x \in \delta, x \mapsto v \in L, H; v \mapsto v \} & \text{if } \text{dom}(\delta) = \text{dom}(L) \\
\text{impossible} & \text{otherwise}
\end{cases}
\]

1.3 Object Constraint System

Our system leverages the idea introduced in the formal core of X10 [3] to extend a base constraint system \( C \) with a larger constraint system \( \mathcal{O}(C) \), built on top of \( C \). The original system \( C \) comprises formulas taken from a decidable SMT logic [2], including, for example, linear arithmetic constraints and uninterpreted predicates. The Object Constraint System \( \mathcal{O}(C) \) introduces the constraints:

- \text{class} \( (C) \), which it true for all classes \( C \) defined in the program;
- \( x \hspace{1pt} \text{hasMm} f \), to denote that the \textit{immutable} field \( f \) is accessible from variable \( x \);
- \( x \hspace{1pt} \text{hasMut} f \), to denote that the \textit{mutable} field \( f \) is accessible from variable \( x \); and
- fields \( \langle x \rangle = F \), to expose all fields available to \( x \).
Figure 8 shows the constraint system as ported from CFG [3]. We refer the reader to that work for details. The main differences are syntactic changes to account for our notion of strengthening. Also the SC-FIELD rule accounts now for both immutable and mutable fields. The main judgment here is of the form:

$$\Gamma \vdash_{S} P$$

where $S$ is the set of classes defined in the program. Substitutions and strengthening operations on field declarations are performed on the types of the declared fields (e.g. SC-FIELD-I, SC-FIELD-C).

1.4 Well-formedness Constraints

The well-formedness rules for predicates, terms, types and heaps can be found in Figure 9. The majority of these rules are routine.

The judgment for term well-formedness assigns a sort to each term $t$, which can be thought of as a base type. The judgment $\Gamma \vdash_{q} T$ is used as a shortcut for any further constraints that the $f$ operator might impose on its arguments $\tilde{t}$. For example if $f$ is the equality operator then the two arguments are required to have types that are related via subtyping, i.e. if $t_1 : N_1$ and $t_2 : N_2$, it needs to be the case that $N_1 \leq N_2$ or $N_2 \leq N_1$.

Type well-formedness is typical among similar refinement types [1].

1.5 Subtyping

Figure 10 presents the full set of subtyping rules, which borrows ideas from similar systems [1, 5].
Syntax

Expression  
\[ e ::= x | c | \textbf{this} | e.f | e.m(\overline{e}) | \textbf{new} C(\overline{e}) | e \textbf{ as } T | e_1.f \leftarrow e_2 | u \langle e \rangle \]

SSA context  
\[ u ::= \langle \rangle | \textbf{let} x = e \textbf{ in} \langle \rangle | \textbf{letif} \left[ \overline{\phi}(e) \right] ? u_1 : u_2 \textbf{ in} \langle \rangle \]

Term  
\[ w ::= e | u \]

\( \Phi \)-Vars  
\[ \phi ::= (x, x_1, x_2) \]

Field Decl.  
\[ F ::= \cdot | o.f:T | \Box f:T | F_1; F_2 \]

Method Decl.  
\[ M ::= \cdot | m(\overline{x}; T) \{ p \} : T \mid M_1; M_2 \]

Field Def.  
\[ \widetilde{F} ::= \cdot | f:=v \mid \widetilde{F}_1; \widetilde{F}_2 \]

Method Def.  
\[ \widetilde{M} ::= \cdot | \textbf{def} m(\overline{x}; T) \{ p \} : T = e \mid \widetilde{M}_1; \widetilde{M}_2 \]

Class Def.  
\[ \widetilde{C} ::= \textbf{class} C \{ p \} \triangleleft R \{ F; \widetilde{M} \} \]

Signature  
\[ S ::= \cdot | \widetilde{C} \mid S_1; S_2 \]

Program  
\[ P ::= S; e \]

Runtime Configuration

Evaluation Context  
\[ E ::= \langle \rangle | E.f | E.m(\overline{e}) | v.m(\overline{v}, E, \overline{v}) | \textbf{new} C(\overline{v}, E, \overline{v}) | E \textbf{ as } T | \]
\[ \textbf{let} x = E \textbf{ in} e | E.f \leftarrow e | v.f \leftarrow E | \textbf{letif} \left[ \overline{\phi}(E) \right] ? e : e \textbf{ in} e \]

SSA Eval. Context  
\[ U ::= \textbf{let} x = E \textbf{ in} \langle \rangle | \textbf{letif} \left[ \overline{\phi}(E) \right] ? u_1 : u_2 \textbf{ in} \langle \rangle \]

Term Eval. Context  
\[ W ::= E \mid U \]

Runtime Conf.  
\[ R ::= K ; e \]

State  
\[ K ::= S ; H \]

Heap  
\[ H ::= \cdot | l \mapsto O \mid H_1; H_2 \]

Store  
\[ L ::= \cdot | x \mapsto v \mid L_1; L_2 \]

Value  
\[ v ::= l | c \]

Object  
\[ O ::= \{ \text{proto:} l; f: \widetilde{F} \} | \{ \text{name:} C; \text{proto:} l; m: \widetilde{M} \} \]

Figure 3: IRSC: syntax and runtime configuration
Operational Semantics for IRSC

**R-Field**

\[ K.H(l) = \{ \text{proto} : l'; \ f : \overline{F} \} \]

\[ f := v \in \overline{F} \]

\[ K.l.f \rightarrow K.v \]

**R-Call**

\[ \text{resolveMethod}(H, l, m) = (\text{def} \ m (\overline{x}; \overline{S}) \{ p \} : T = e) \]

\[ \text{eval}(\overline{x}/x, l/\text{this}| p) = \text{true} \]

\[ K.l.m(\overline{v}) \rightarrow K.\overline{v}/x, l/\text{this}| e \]

**R-New**

\[ H(l_0) = \{ \text{name} : C; \ \text{proto} : l_0'; \ m : \overline{M} \} \]

\[ \text{fields}(S, C) = \overline{M} \]

\[ O = \{ \text{proto} : l_0; \ f : \overline{F} \} \]

\[ H' = H[l \mapsto O] \]

\[ l \text{ fresh} \]

**R-LetIn**

\[ K.l.\text{let } x = v \text{ in } e \rightarrow K.\overline{v}/x \]

**R-DotAsgn**

\[ H' = K.H[l \mapsto K.H(l)[f \mapsto v]] \]

\[ K.l.f \leftarrow v \rightarrow K.a H'; v \]

**R-LetIf**

\[ c = \text{true} \Rightarrow i = 1 \]

\[ c = \text{false} \Rightarrow i = 2 \]

\[ K.l.\text{let } if \]
SSA Transformation for Runtime Configurations

K; w ▷ K; e  
S-EXP-RtConf  
K.S ▷ S  
K; K.H ▷ H  
K.L; K.X; w ▷ e  
K; w ▷ S; H; e  
S-STMT-RtConf  
K.S ▷ S  
K; K.H ▷ H  
K.L; K.X; s ▷ u  
K; s ▷ S; H; u  

Runtime Stack Translation

L; X; w ▷ e  
S-STACK-EMP  
L; w ▷ e  
L; : w ▷ e  
S-STACK-CONS  
L; : w ▷ e  
L; : E ▷ W  
S-EC-STACK-EMP  
L; E ▷ W  
L; : E ▷ W  
S-EC-STACK-CONS  
L; : E ▷ W  
L; : E ▷ E  
L; : E ▷ E  

Runtime Term Translation (selected rules)

SR-METH  
:: B ▷ e  
m(\pi) \{B\} ▷ def m(\pi) = e  
SR-VAL  
H; v ▷ v  
L; v ▷ v  
SR-VARREF  
\Delta (x) \models x ▷ x  
\theta = toSubst (\Delta (x), L, H)  
L; x ▷ \theta x  
SR-CALL  
L; e ▷ e  
L; \pi ▷ \pi  
toString (m) = toString (m)  
L; e, m(\pi) ▷ e.m(\pi)  
SR-VARDECL  
\pi \models x ▷ x \in \Delta  
\var x = e  
L; \var x = e ▷ e  
\pi = e \in (\pi)  
SR-ASGN  
\pi \models x \in \Delta  
\var x = e  
L; \var x = e ▷ e  
\pi = e \in (\pi)  

Evaluation Context Translation (selected rules)

\pi \models x ▷ x  
\var x = e ▷ e  
\pi = e \in (\pi)  
\pi \models x \in \Delta  
\var x = e ▷ e  
\pi = e \in (\pi)  

Figure 6: SSA Transformation Rules for Runtime Configurations
Heap Translation

\[
\begin{align*}
\text{K; } H & \mapsto H \\
\text{H; } v & \mapsto v
\end{align*}
\]

S-HEAP-EMP
\[
\text{K; } \cdot \mapsto .
\]

S-HEAP-BND
\[
\text{K; } H; \ 0 \mapsto \ O \quad \text{l fresh}
\]

S-HEAP-CONS
\[
\text{K; } H_1 \mapsto H_1 \quad \text{K; } H_2 \mapsto H_2
\]

S-LOC
\[
1 \mapsto 0 \in H \quad \text{H; } (1 \mapsto 0) \mapsto (l \mapsto O)
\]

S-CONST
\[
toValue (c) = toValue (c) \quad \text{c fresh}
\]

H; 0 \mapsto O

H; 1 \mapsto l \\
H; F \mapsto \bar{F}

H; \{\text{proto: } l; \ f; F\} \mapsto \{\text{proto: } l; \ f; \bar{F}\}

\[
\begin{align*}
\text{H; } l \mapsto l \\
\text{H; } F \mapsto \bar{F}
\end{align*}
\]

Heap Object Translation

\[
\text{H; } l \mapsto l \\
\text{H; } F \mapsto \bar{F}
\]

\[
\text{H; } \{\text{name: } C; \ \text{proto: } l; \ m: M\} \mapsto \{\text{name: } C; \ \text{proto: } l; \ m: \bar{M}\}
\]

Figure 7: SSA Transformation Rules for Heaps and Objects

Structural Constraints

\[
\begin{align*}
\text{SC-CLASS} & \\
\text{class } C \ {\{p\} \ a R \ {\{F; \bar{M}\} \in S} & \text{ } \Gamma \vdash_S \text{class} (C)
\end{align*}
\]

\[
\begin{align*}
\text{SC-INV} & \\
\Gamma \vdash_S x: C, \ \text{class} (C) & \text{ } \Gamma \vdash_S \text{inv} (C, x)
\end{align*}
\]

\[
\begin{align*}
\text{SC-FIELD} & \\
\Gamma \vdash_S \text{fields} (x) = \circ f_1, \bar{T_i}, \circ g_i, S_i & \text{ } \Gamma \vdash_S x \ \text{hasImm} f_1, T_i \\
& \text{ } \Gamma \vdash_S x \ \text{hasMut} g_i, S_i
\end{align*}
\]

SC-FIELD-I
\[
\Gamma, x: D \vdash_S \text{fields} (x) = F \\
\text{class } C \ {\{p\} \ a R \ {\{F'; \bar{M}'\} \in S} & \text{ } \Gamma, x: D \vdash_S \text{fields} (x) = F, \ [x/\text{this}] F'
\]

SC-METH-B
\[
\Gamma \vdash_S \text{class} (C) \quad \theta = [x/\text{this}] \\
\text{def } m \ (\bar{\xi}; T) \ {\{p\} : T = e \in C} & \text{ } \Gamma, x: C \vdash_S x \ \text{has} \ (\text{def } m \ (\bar{\xi}; T) \ {\{\theta p\} : \theta T = e})
\]

SC-METH-C
\[
\Gamma, x: C \vdash_S x \ \text{has} \ (\text{def } m \ (\bar{\xi}; T) \ {\{p_0\} : T = e}) & \text{ } \Gamma, x: \nu: C \ | \ p \vdash_S x \ \text{has} \ (\text{def } m \ (\bar{\xi}; T) \ {\{p_0\} : T \cap [x/\text{this}] p = e})
\]

\[
\begin{align*}
\text{SC-METH-I} & \\
\Gamma, x: D \vdash_S x \ \text{has} \ (\text{def } m \ (\bar{\xi}; T) \ {\{p\} : T = e}) \\
\text{class } C \ {\{p\} \ a D \ {\{F; \bar{M}\} \in S} & \text{ } m \notin \bar{M}
\end{align*}
\]

Figure 8: Structural Constraints (adapted from [3])
Well-Formed Predicates

\[ \Gamma \vdash p \]

Well-Formed Terms

\[ \Gamma \vdash t : N \]

WF-AND
\[ \Gamma \vdash p_1 \quad \Gamma \vdash p_2 \quad \Gamma \vdash p_1 \land p_2 \]

WF-NOT
\[ \Gamma \vdash p \quad \Gamma \vdash \neg p \]

WF-Term
\[ \Gamma \vdash t : \text{bool} \]

Well-Formed Types

\[ \Gamma \vdash T \]

WT-BASE
\[ \Gamma, \nu : N \vdash p \quad \Gamma \vdash \{ \nu : N \mid p \} \]

Well-Formed Heaps

\[ \Sigma \vdash H \]

WF-Heap-Inst
\[ O \equiv \{ \text{proto: } l' ; f : \overline{F} \} \quad \overline{F} = \circ \overline{f} := \nu_1, \circ \overline{g} = \nu_M \quad |\Sigma(l)| = C \]
\[ \Gamma, z : C \vdash \text{fields}(z) = \circ \overline{f} : \overline{R}, \circ \overline{g} : \overline{U} \quad \Sigma \vdash \nu_1 : T_1 \quad \Sigma \vdash \nu_M : T_M \]
\[ \Gamma, z : C, \nu_1 : \text{self} (\overline{T}_1, z, \overline{f}) \vdash T_1 \leq \overline{R}, T_M \leq \overline{U}, \text{inv}(C, z) \]
\[ \Sigma \vdash \lambda \rightarrow O \]

Figure 9: Well-Formedness Rules

Subtyping

\[ \Gamma \vdash T \leq T' \]

\[ \leq \text{-RefL} \]
\[ \Gamma \vdash T \leq T \]
\[ \leq \text{-Trans} \]
\[ \Gamma \vdash T_1 \leq T_2 \quad \Gamma \vdash T_2 \leq T_3 \quad \Gamma \vdash T_1 \leq T_3 \]
\[ \leq \text{-Extends} \]
\[ \text{class } C \{ p \} \triangleleft D \{ F ; \overline{M} \} \]
\[ \Gamma \vdash C \leq D \]
\[ \leq \text{-Base} \]
\[ \Gamma \vdash N \leq N' \]
\[ \Gamma \vdash \forall \nu : N \mid p \leq \forall \nu : N' \mid p' \]

\[ \leq \text{-Witness} \]
\[ \Gamma \vdash e : S \quad \Gamma \vdash T \leq [e/x] T' \]
\[ \Gamma \vdash T \leq \exists x : S. T' \]

\[ \leq \text{-Bind} \]
\[ \Gamma, x : S \vdash T \leq T' \quad x \notin \text{FV}(T') \]
\[ \Gamma \vdash \exists x : S. T \leq T' \]

Figure 10: Subtyping Rules

Runtime Typing Rules

\[ \Sigma \vdash v : T \]
\[ \Sigma \vdash_H O : T \]

RT-T-LOC
\[ \Sigma (l) = T \quad \Sigma \vdash l : T \]

RT-T-Const
\[ \Sigma \vdash v : \text{ty}(c) \]

RT-T-Obj
\[ \Sigma (l) = C \quad \text{fieldDefs}(H, l) = \circ \overline{f} := \nu_1, \circ \overline{g} = \nu_M \quad \Sigma \vdash \nu_1 : T_1 \]
\[ \Sigma \vdash_H \{ \text{proto: } l ; f : \overline{F} \} : \exists \nu_1 : T_1, \forall \nu : C \mid \nu.f = \nu_1 \land \text{inv}(C, \nu) \}

Figure 11: Typing Runtime Configurations for IRSC
2. Proofs
The main results in this section are:

- Program Consistency Lemma (Lemma 13, page 18)
- Forward Simulation Theorem (Theorem 1, page 22)
- Subject Reduction Theorem (Theorem 2, page 24)
- Progress Theorem (Theorem 3, page 31)

2.1 SSA Translation

Definition 1 (Environment Substitution).

\[ [\delta_1/\delta_2] = [\delta_1/\delta_2] \quad \text{where} \quad (x, \delta_1, \delta_2) = \delta_1 \bowtie \delta_2 \]

Definition 2 (Valid Configuration).

\[
\text{validConf } (K; w) = \begin{cases} 
  \text{true} & \text{if } (K.X = :) \Rightarrow \exists B \ s.t. \ w \equiv B \\
  \text{false} & \text{otherwise}
\end{cases}
\]

Assumption 1 (Stack Form). Let stack \(X = X_0; L, E\). Evaluation context \(E\) is of one of the following forms:

- \(E_0; \text{return } e\)
- \(\text{return } E_0\)

Lemma 1 (Global Environment Substitution). If \(L; e \xrightarrow{H, \Delta} e\), then \(L; e \xrightarrow{H, \Delta'} [\Delta' (e) / \Delta (e)] e\)

Lemma 2 (Evaluation Context). If \(L; w \xrightarrow{H, \Delta} E[e]\) then there exist \(E\) and \(e\) s.t.:

- \(w \equiv E[e]\)
- \(L; E \xrightarrow{H, \Delta} E\)
- \(L; e \xrightarrow{H, \Delta} e\)

Proof. By induction on the derivation of the input transformation.

Lemma 3 (Translation under Store). If \(\cdot; B \xrightarrow{l.m. (\bar{v})} e\), then \(L; B \xrightarrow{H, \Delta} \theta e\), where \(\theta = \text{toSubst} (\Delta (B), L, H)\).

Proof. By induction on the structure of the input translation.

Lemma 4 (Canonical Forms).

(a) If \(L; w \xrightarrow{H, \Delta} c\), then \(w \equiv c\)

(b) If \(L; w \xrightarrow{H, \Delta} l.m (\bar{v})\), then \(w \equiv 1.m(\bar{v})\)

(c) If \(L; w \xrightarrow{H, \Delta} \text{letif } [\bar{v}] (e) ? u_1 : u_2 \text{ in } e'\), then \(w \equiv \text{if}(e)\{s_1\} \text{else}\{s_2\}; \text{return } e'\)

(d) If \(M \xrightarrow{H, \Delta} \text{def } m (\bar{v}) = e_0, \text{then } M \equiv m(\bar{v}) \{B\}\)

Lemma 5 (Translation Closed under Evaluation Context Composition). If

(a) \(L; E_0 \xrightarrow{H, \Delta} E_0\)

(b) \(L'; (L; E_1); B \xrightarrow{H, \Delta} e\)

then \(L'; (L; E_0[E_1]); B \xrightarrow{H, \Delta} E_0[e]\)

Lemma 6 (Heap and Store Weakening). If \(L; X; E \xrightarrow{H, \Delta} W\)
\[
\forall H', L' \text{ s.t. } H' \supseteq H \text{ and } L' \supseteq L, \text{ it holds that } L'; X; E \xrightarrow{H', \Delta} W
\]

**Lemma 7** (Translation Closed under Stack Extension). If

(a) \( L_0; X_0; E_0 \xrightarrow{H, \Delta} E_0 \)

(b) \( L_1; X_1; B_1 \xrightarrow{H, \Delta} e_1 \)

then \( L_1; (X_0; L_0, X_0; X_1); B_1 \xrightarrow{H, \Delta} E_0[e_1] \)

**Proof.** We proceed by induction on the structure of derivation (b):

- **[S-STACK-EMP]:** Fact (b) has the form:
  \[ L_1; ::; B_1 \xrightarrow{H, \Delta} e_1 \]  

  By applying Rule S-STACK-CONS on 2.1 and (a):
  \[ L_1; (X_0; L_0, E_0); B_1 \xrightarrow{H, \Delta} E_0[e_1] \]  

  Which proves the wanted result.

- **[S-STACK-CONS]:** Fact (b) has the form:
  \[ L_1; (X; L, E); B_1 \xrightarrow{H, \Delta} E[e_{1,1}] \]  

  By inverting Rule S-STACK-CONS on 2.3:
  \[ L_1; ::; B_1 \xrightarrow{H, \Delta} e_{1,1} \]  

  \[ L; X; E \xrightarrow{H, \Delta} E \]  

  By induction hypothesis on (a) and 2.5 (the lemma can easily be extended to evaluation contexts):
  \[ L; (X_0; L_0, E_0; X); E \xrightarrow{H, \Delta} E_0[E] \]  

  By applying Rule S-EC-STACK-CONS on 2.4 and 2.6:
  \[ L_1; (X_0; L_0, E_0; X; L, E); B_1 \xrightarrow{H, \Delta} E_0[E[e_{1,1}]] \]  

  Which proves the wanted result.

**Lemma 8** (Translation Closed under Evaluation Context Application). If

(a) \( L; X; E \xrightarrow{H, \Delta} W \)

(b) \( L; e \xrightarrow{H, \Delta} e \)

then \( L; X; E[\sigma] \xrightarrow{H, \Delta} W[e] \)

**Proof.** By induction on the derivation of (a).  

**Lemma 9** (Method Resolution). If

(a) \( K; H \rightarrow H \)

(b) \( H; l \rightarrow l \)

(c) \( \text{toString}(m) = \text{toString}(m) \)

(d) \( \text{resolveMethod}(H, l, m) = \bar{M} \)

then:
(e) \text{resolve\_method}(H, l, m) = \bar{M}

(f) M \rightarrow \bar{M}

\textbf{Lemma 10} (Value Monotonicity). If

(a) validConf \((K; w)\)
(b) \(K; w \xrightarrow{\Delta} K; v\)

then there exist \(L'\) and \(w'\) s.t.:

(c) \(K; w \rightarrow^* K'; w'\)
(d) \(K'; w' \xrightarrow{\Delta} K; v\)

(e) \(w' \equiv \begin{cases} \text{return } v & \text{if } w \equiv B \\ v & \text{otherwise} \end{cases}\)

(f) If \(K.X = \cdot\) then \(K'.L = K.L\)

where \(K' \equiv K.S; L'; \cdot; K.H\)

\textit{Proof. By induction on the structure of the derivation (b).}

\textbf{Lemma 11} (Top-Level Reduction). If

\(S; L; X; H; w \rightarrow S; L'; X'; H'; w'\)

then for a stack \(X_0\) it holds that:

\(S; L; (X_0; X); H; w \rightarrow S; L'; (X_0; X'); H'; w'\)

\textit{Proof. By induction on the structure of the input reduction.}

\textbf{Lemma 12} (Empty Stack Consistency). If

(a) \(K; w \rightarrow K; e\)
(b) \(K.X = \cdot\)
(c) \(K; e \rightarrow K'; e'\)

then there exist \(K'\) and \(w'\) s.t.:

(d) \(K; w \rightarrow^* K'; w'\)
(e) \(K'; w' \xrightarrow{\Delta} K'; e'\)

(f) \(\triangleright\) If \(w \equiv E[l.m(\emptyset)]\) then:

- \(K'.X = K.L; E\)
- \(K'.H = K.H\)
- \(\exists B' s.t. w' \equiv B'\)
- \(K' = K\)

\(\triangleright\) Otherwise:

- \(K'.X = \cdot\)
- \(K'.H \supseteq K.H\)
- \(K'.L \supseteq K.L\)
- \(\text{If } \exists e s.t. w \equiv e \text{ then } \exists e' s.t. w' \equiv e'\)
- \(\text{If } \exists B s.t. w \equiv B \text{ then } \exists B' s.t. w' \equiv B'\)

\textit{Proof. Fact (a) has the form:

\(K; w \rightarrow S; H; e\) (6.1)\n
Because of fact (b):

\(K \equiv S; L; ::; H\) (6.2)\n
12 2016/4/8
By inverting Rule S-EXP-RTCONF on 6.1:

\[
\begin{align*}
S & \triangleleft S \\
K; H & \leftrightarrow H \\
L; \cdot; \cdot & \text{s.t.} \frac{H, \Delta}{e}
\end{align*}
\] (6.3)

\[
\begin{align*}
K; H & \leftrightarrow H \\
L; \cdot; \cdot & \text{s.t.} \frac{H, \Delta}{e}
\end{align*}
\] (6.4)

\[
\begin{align*}
L; \cdot; \cdot & \text{s.t.} \frac{H, \Delta}{e}
\end{align*}
\] (6.5)

By inverting S-STACK-EMP on 6.10:

\[
\begin{align*}
L; \cdot; \cdot & \text{s.t.} \frac{H, \Delta}{e}
\end{align*}
\] (6.6)

Suppose \( w \) is a value. By Rules S-CONST and S-LOC, \( e \) is also a value: a contradiction because of (c). Hence:

\[
\begin{align*}
w & \text{ not a value}
\end{align*}
\] (6.7)

We proceed by induction on the structure of reduction (c):

- [RC-ECTx]

\[
\begin{align*}
K; E_0[e_0] & \rightarrow K'; E_0[e'_0]
\end{align*}
\] (6.8)

By inverting RC-ECTx on 6.8:

\[
\begin{align*}
K; e_0 & \rightarrow K'; e'_0
\end{align*}
\] (6.9)

Fact 6.6 is of the form:

\[
\begin{align*}
L; w & \text{s.t.} \frac{H, \Delta}{e_0}
\end{align*}
\] (6.10)

By Lemma 2 on 6.10:

\[
\begin{align*}
w & \equiv E_0[e_0] \\
L; E_0 & \text{s.t.} \frac{H, \Delta}{E_0} \\
L; e_0 & \text{s.t.} \frac{H, \Delta}{e_0}
\end{align*}
\] (6.11)

(6.12)

(6.13)

By applying Rule S-STACK-EMP on 6.13:

\[
\begin{align*}
L; \cdot; e_0 & \text{s.t.} \frac{H, \Delta}{e_0}
\end{align*}
\] (6.14)

By applying Rule S-EXP-RTCONF on 6.3, 6.4 and 6.14:

\[
\begin{align*}
K; e_0 & \triangleleft K; e_0
\end{align*}
\] (6.15)

By induction hypothesis using 6.15, (b) and 6.9:

\[
\begin{align*}
S; L; \cdot; H; e_0 & \rightarrow S; L'; X'; H'; w'_0 \\
S; L'; X'; H'; w'_0 & \triangleleft K'; e'_0
\end{align*}
\] (6.16)

(6.17)

We examine cases on the form of \( e_0 \):

- Case \( e_0 \equiv E_1[1 \cdot m(\overline{v})] \):

\[
\begin{align*}
X' & = L; E_1 \\
H' & = H \\
w'_0 & = B' \\
K' & = K
\end{align*}
\] (6.18)

(6.19)

(6.20)

(6.21)
For some method body $B'$. So 6.17 becomes:

$$S; L'; (L; E_1); H; B' \xrightarrow{\Delta} K; e'_0$$

(6.22)

By inverting rule R-CALL on 6.16:

 resolve_method $(H, 1, m) = m(\forall) \{B'\}$

(6.23)

$$L' = \forall \mapsto \forall; \text{this} \mapsto 1$$

(6.24)

$$X'_0 = L, E_1$$

(6.25)

By applying rule R-CALL using 6.23, 6.24 and $X' = L, E_0[E_1]$ on $K; w = S; L; \cdot; H; (E_0[E_1])[1.m(\forall)]:$

$$S; L; \cdot; H; (E_0[E_1])[1.m(\forall)] \rightarrow S; L'; (L; E_0[E_1]); H; B'$$

(6.26)

Which proves (d). By inverting Rule S-EXP-RtCONF on 6.22:

$$K'; H \mapsto H$$

(6.27)

$$L'; (L; E_1); B' \xrightarrow{H.\Delta} e'_0$$

(6.28)

From Lemma 5 on 6.12 and 6.28:

$$L'; (L; E_0[E_1]); B' \xrightarrow{H.\Delta} E_0[e'_0]$$

(6.29)

By applying rule S-EXP-RtCONF using 6.3, 6.27 and 6.29:

$$S; L'; (L; E_0[E_1]); H; B' \xrightarrow{\Delta} K; E_0[e'_0]$$

(6.30)

Which proves (e). By 6.11 and the current case:

$$w = (E_0[E_1])[1.m(\forall)]$$

(6.31)

By 6.26 and 6.30:

$$K'.X = L; E_0[E_1]$$

(6.32)

$$w' = B'$$

(6.33)

$$K' = K$$

(6.34)

By 6.32, 6.19, 6.33 and 6.34 we prove (f).

- All remaining cases:

  $$X' \equiv .$$

  (6.35)

  $$H' \supseteq H$$

  (6.36)

  $$L' \supseteq L$$

  (6.37)

  $$w'_0 \equiv e'_0$$

  (6.38)

So 6.16 and 6.17 become:

$$S; L; \cdot; H; e_0 \rightarrow S; L'; \cdot; H'; e'_0$$

(6.39)

$$S; L'; \cdot; H'; e'_0 \xrightarrow{\Delta} K'; e'_0$$

(6.40)

By applying Rule R-EVALCTX using 6.39:

$$S; L; \cdot; H; E_0[e_0] \rightarrow S; L'; \cdot; H'; E_0[e'_0]$$

(6.41)
Which proves (d) and (f). By inverting Rules S-EXP-RtCONF and S-STACK-EMP on 6.40:

\[ L'; \emptyset \vdash \frac{\Delta}{e_0} \]

(6.42)

From Lemma 6 using 6.12, 6.36 and 6.37:

\[ L'; E_0 \vdash \frac{\Delta}{E_0} \]

(6.43)

From Lemma 8 on 6.42 and 6.43:

\[ L'; E_0[\emptyset_0] \vdash \frac{\Delta}{E_0[\emptyset_0]} \]

(6.44)

By inverting rule S-EXP-RtCONF on 6.40:

\[ K'; H' \leftrightarrow H' \]

(6.45)

By Rule S-EXP-RtCONF using 6.3, 6.44 and 6.45:

\[ S; L'; :: H'; E_0[\emptyset_0] \leftrightarrow S; H'; E_0[\emptyset_0] \]

(6.46)

Which proves (e).

• [R-CALL]:

\[ K; l.m (\overline{\text{v}}) \longrightarrow K; [\overline{\text{v}}/\overline{x}, l/\overline{\text{this}}] e_0 \]

(6.47)

Where by inverting R-CALL on 6.47:

\[ \text{resolveMethod} (H, l, m) = (\text{def} \ m (\overline{x}) = e_0) \]

(6.48)

Fact 6.5 is of the form:

\[ L; :: w \vdash l.m (\overline{x}) \]

(6.49)

By Lemma 4(b) on 6.49:

\[ w \equiv l.m(\overline{x}) \]

(6.50)

So 6.49 becomes:

\[ L; :: l.m(\overline{x}) \vdash l.m (\overline{x}) \]

(6.51)

By inverting Rule S-STACK-EMP on 6.51:

\[ L; l.m(\overline{x}) \vdash l.m (\overline{x}) \]

(6.52)

By inverting Rule SR-CALL on 6.52:

\[ L; l \vdash l \]

(6.53)

\[ L; \overline{\text{v}} \vdash \overline{\text{v}} \]

(6.54)

\[ \text{toString} (m) = \text{toString} (m) \]

(6.55)

By inverting SR-VAL on 6.53 and 6.54:

\[ H; l \leftrightarrow l \]

(6.56)

\[ H; \overline{\text{v}} \leftrightarrow \overline{\text{v}} \]

(6.57)
By Lemma 9 on 6.4, 6.56, 6.55 and 6.48:

\[
\text{resolve}\_\text{method}(H, l, m) = \tilde{M}
\]

(6.58)

\[
\tilde{M} \xrightarrow{\Delta} \text{def } m(x) = e_0
\]

(6.59)

By Lemma 4(d) on 6.59:

\[
\tilde{M} \equiv m(x) \{B\}
\]

(6.60)

By applying Rule R-CALL using 6.58, 6.63, 6.64 and \(E = [\ ]\):

\[
S; L; X; H; l . m(\tau) \rightarrow S; L'; X'; H; B
\]

(6.61)

Which proves (d). By inverting rule SR-METH on 6.59:

\[
\vdash_B \xrightarrow{\Delta} e
\]

(6.62)

Let a store \(L'\) and a stack \(X'\) s.t.:

\[
L' \equiv x \mapsto v; \text{this} \mapsto 1
\]

(6.63)

\[
X' \equiv L; [\ ]
\]

(6.64)

By applying Lemma 3 on 6.62

\[
L'; B \xrightarrow{H, \Delta} \theta e_0
\]

(6.65)

Where:

\[
\theta = \text{toSubst}\ (\Delta(B), L', H)
\]

\[
= \{ [v/x] | x \mapsto x \in \Delta(B), x \mapsto v \in L', H; v \equiv v \}
\]

\[
= [v/x, l/\text{this}]
\]

(6.66)

We pick:

\[
w' \equiv B
\]

(6.67)

By applying Rule S-STACK-EMP using 6.65:

\[
L'; : ; B \xrightarrow{H, \Delta} \theta e_0
\]

(6.68)

It holds that:

\[
L; : ; [ ] \xrightarrow{H, \Delta} [ ]
\]

(6.69)

By Rule S-STACK-CONS on 6.68 and 6.69:

\[
L'; (L, [ ]); : ; B \xrightarrow{H, \Delta} \theta e_0
\]

(6.70)

By Rule S-EXP-RtCONF using 6.3, 6.4 and 6.70:

\[
S; L'; X'; H; B \xrightarrow{\Delta} S; H; \theta e_0
\]

(6.71)

Which proves (e). From 6.64, 6.61, 6.67 and 6.56 we prove (f).
• [R-LETIF]:

\[ K; \text{letif } [x, x_1, x_2](c) ? u_1 : u_2 \text{ in } e_0 \rightarrow K; u_i \langle [x_i/x] e_0 \rangle \]

\[ c = \text{true} \Rightarrow i = 1 \]
\[ c = \text{false} \Rightarrow i = 2 \]

Let:

\[ c = \text{true} \]

The case for false is symmetrical. Facts 6.72 and 6.6 become:

\[ K; \text{letif } [x, x_1, x_2](\text{true}) ? u_1 : u_2 \text{ in } e_0 \rightarrow K; u_1 \langle [x_1/x] e_0 \rangle \]

By Lemma 4(c) on 6.77:

\[ w = \text{if}(e_c) \{s_1\} \text{ else } \{s_2\}; \text{return } e_0 \]

So 6.77 becomes:

\[ L; \text{if}(e_c) \{s_1\} \text{ else } \{s_2\}; \text{return } e_0 \rightarrow L; \text{letif } [x, x_1, x_2](\text{true}) ? u_1 : u_2 \text{ in } e_0 \]

By inverting Rule SR-BODY on 6.80:

\[ L; e_{c} \rightarrow \text{true} \]
\[ L; s_{1} \rightarrow u_{1} \]
\[ L; s_{2} \rightarrow u_{2} \]
\[ (x, x_1, x_2) = \Delta' [s_1] \sqsupset \Delta s_2 \]
\[ x = \Delta' [\text{if}(e_c) \{s_1\} \text{ else } \{s_2\}](x) \]

By Lemma 4 on 6.83 we get:

\[ e_c = \text{true} \]

By Rules R-EVALCTX and R-ITE we get:

\[ K; \text{if}(\text{true}) \{s_1\} \text{ else } \{s_2\}; \text{return } e_0 \rightarrow K; s_1; \text{return } e_0 \]

Which proves (d). Let:

\[ \Delta'' = \Delta'(e_0) \]

By Lemma 1 on 6.82 using 6.90:

\[ L; e_0 \rightarrow (\Delta''(e_0)) e_0 \]
From 6.81 and 6.90 it holds that:

\[
\Delta'(e_0) = \Delta [\text{if(true)} \{s_1\} \text{else} \{s_2\}] \\
\Delta''(e_0) = \Delta \{s_1\}
\] (6.92)

So:

\[
\Delta'(e_0) \bowtie \Delta''(e_0) = (x, x_1, x)
\] (6.93)

By Definition 1:

\[
[ \Delta''(e_0) / \Delta'(e_0) ] = [x_1/x]
\] (6.94)

So 6.91 becomes:

\[
L; e_0 \xrightarrow{\Delta''} [x_1/x] e_0
\] (6.95)

By applying Rule SR-BODY on 6.84, 6.93 and 6.96, using 6.95:

\[
L; s_1; \text{return} e_0 \xrightarrow{\Delta} u_1 ([x_1/x] e_0)
\] (6.96)

Which, using S-EXP-RtCONF and S-STACK-EMP, prove (e) and (f).

- [R-CAST], [R-NEW], [R-LETIN], [R-DOTASGN], [R-FIELD]: Cases handled in similar fashion as before.

**Corollary 1** (Empty Stack Valid Configuration). If

(a) \(K; w \xrightarrow{\Delta} K; e\)
(b) \(K; L; X = \cdot\)
(c) \(K; e \rightarrow K'; e'\)

then \(K; w \rightarrow^* K'; w'\) with validConf \((K'; w')\).

**Proof.** Examine all cases of result (f) of Lemma 12.

**Lemma 13** (Consistency). If

(a) \(K; w \xrightarrow{\Delta} K; e\)
(b) \(K; e \rightarrow K'; e'\)
(c) validConf \((K; w)\)

then there exist \(K'\) and \(w'\) s.t.: 
(d) \(K; w \rightarrow^* K'; w'\).
(e) \(K'; w' \xrightarrow{\Delta} K'; e'\)
(f) validConf \((K'; w')\)

**Proof.** Let:

\[
K \equiv S; L; X; H
\] (6.1)

By inverting Rule S-EXP-RtCONF on (a):

\[
S \xrightarrow{\Delta} S
\] (6.2)

\[
K; H \xrightarrow{\Delta} H
\] (6.3)

\[
L; X; w \xrightarrow{\Delta} e
\] (6.4)

We proceed by induction on the derivation 6.4:
• [S-Stack-Emp]:

\[
L; \cdot; w \xrightarrow{\Delta} e \quad (6.5)
\]

By Lemma 12 using (a) and (b) there exist \( w' \) and \( K' \) s.t.:

\[
K; w \rightarrow^* K'; w' \quad (6.6)
\]

\[
K'; w' \xrightarrow{\Delta} K'; e' \quad (6.7)
\]

From Corollary 1 using (a), (b) and (c) we get:

\[
\text{validConf} (K'; w') \quad (6.8)
\]

We prove (d), (e) and (f) by 6.6, 6.7 and 6.8, respectively.

• [S-Stack-Cons]:

\[
L; (X_0; L_0; E_0); w \xrightarrow{\Delta} E_0[e_0] \quad (6.9)
\]

Where:

\[
X \equiv X_0; L_0; E_0 \quad (6.10)
\]

By (c) and the definition of a valid configuration, there exists a \( B_0 \) s.t.:

\[
w \equiv B_0 \quad (6.11)
\]

By inverting Rule S-Stack-Cons on 6.9 using 6.11:

\[
L; \cdot; B_0 \xrightarrow{\Delta} e_0 \quad (6.12)
\]

\[
L_0; X_0; E_0 \xrightarrow{\Delta} E_0 \quad (6.13)
\]

By applying rule S-Exp-RtConf on 6.2, 6.3 and 6.12:

\[
S; L; \cdot; H; B_0 \xrightarrow{\Delta} S; H; e_0 \quad (6.14)
\]

We examine cases on the configuration of \( K; e_0 \):

- Case \( K; e_0 \) is a terminal configuration, so there exists \( v \) s.t.:

\[
e_0 \equiv v \quad (6.15)
\]

Fact 6.14 becomes:

\[
S; L; \cdot; H; B_0 \xrightarrow{\Delta} S; H; v \quad (6.16)
\]

By Lemma 10 on 6.16:

\[
S; L; \cdot; H; B_0 \rightarrow^* S; L; \cdot; H; \text{return} v \quad (6.17)
\]

\[
S; L; \cdot; H; \text{return} v \xrightarrow{\Delta} K; v \quad (6.18)
\]

By Lemma 11 on 6.17:

\[
S; L; X; H; B_0 \rightarrow^* S; L; X; H; \text{return} v \quad (6.19)
\]

By inverting Rule S-Exp-RtConf on 6.18:

\[
L; \cdot; \text{return} v \xrightarrow{\Delta} v \quad (6.20)
\]
By applying Rule S-STACK-CONS on 6.20 and 6.13:

\[ L; (X_0; L_0; E_0); \text{return} v \xrightarrow{\text{H, } A} E_0[v] \]  

(6.21)

By applying Rule S-EXP-RtCONF on 6.2, 6.3 and 6.21:

\[ S; L; (X_0; L_0; E_0); H; \text{return} v \xrightarrow{\Delta} S; H; E_0[v] \]  

(6.22)

By applying Rule R-RET on on THe left-hand side of 6.22:

\[ S; L; (X_0; L_0; E_0); H; \text{return} v \longrightarrow S; L_0; X_0; H; E_0[v] \]  

(6.23)

By inverting S-STACK-EMP and SR-BODY on 6.20:

\[ L; v \xrightarrow{\text{H, } A} v \]  

(6.24)

By inverting Rule SR-VAL on 6.24:

\[ H; v \xleftarrow{} v \]  

(6.25)

By applying Rule SR-VAL on 6.25 using L_0:

\[ L_0; v \xrightarrow{\text{H, } A} v \]  

(6.26)

By applying Lemma 8 on 6.13 and 6.26:

\[ L_0; X_0; E_0[v] \xleftarrow{\text{H, } A} E_0[v] \]  

(6.27)

By applying Rule S-EXP-RtCONF on 6.2, 6.3 and 6.27:

\[ S; L_0; X_0; H; E_0[v] \xrightarrow{\Delta} S; H; E_0[v] \]  

(6.28)

Because of 6.11:

\[ \text{validConf} (S; L_0; X_0; H; E_0[v]) \]  

(6.29)

By induction hypothesis using 6.28, (b) and 6.29:

\[ S; L_0; X_0; H; E_0[v] \longrightarrow^* K'; w' \]  

(6.30)

\[ K'; w' \xrightarrow{\Delta} K'; e' \]  

(6.31)

\[ \text{validConf} (K'; w') \]  

(6.32)

We prove (d) by 6.19, 6.23 and 6.33; (e) by 6.31; and (f) by 6.32.

- Case K; e_0 is a non-terminal configuration, so there exists e'_0 s.t.:

\[ K; e_0 \longrightarrow K'; e'_0 \]  

(6.33)

By Rule RC-ECtx using 6.33:

\[ K; E_0[e_0] \longrightarrow K'; E_0[e'_0] \]  

(6.34)

By Lemma 12 using 6.14 and 6.33:

\[ S; L; H; B_0 \longrightarrow^* K'; w' \]  

(6.35)

\[ K'; w' \xrightarrow{\Delta} K'; e'_0 \]  

(6.36)

And we examine cases on the form of B_0 for the last result of the above lemma:
Case 8 \(B_0 \equiv E[1, m(\forall)]\). It holds that:

\[
K'; w' \equiv S; L_1; (L, E); H; B_1
\]

(6.37)

So 6.36 becomes:

\[
S; L_1; (L, E); H; B_1 \xrightarrow{H, \Delta} K'; e_0'
\]

(6.38)

By inverting S-EXP-RtCONF on 6.38:

\[
L_1; (L, E); B_1 \xrightarrow{H, \Delta} e_0'
\]

(6.39)

By Lemma 7 using 6.13 and 6.39:

\[
L_1; (X_0; L_0, E_0; L, E); B_1 \xrightarrow{H, \Delta} E_0[e_0']
\]

(6.40)

Let:

\[
X' \equiv X_0; L_0, E_0; L, E
\]

(6.41)

By applying Rule S-EXP-RtCONF on 6.2, 6.3 and 6.40:

\[
S; L_1; X'; H; B_1 \xrightarrow{H, \Delta} K'; E_0[e_0']
\]

(6.42)

By Lemma 11 on 6.35:

\[
S; L; X; H; B_0 \rightarrow^* S; L_1; X'; H; B_1
\]

(6.43)

We prove (d), (e) and (f) by 6.48, 6.53 and 6.46, respectively.

For all remaining cases on \(B_0\):

\[
H' \supseteq H
\]

(6.44)

\[
L' \supseteq L
\]

(6.45)

Because of 6.11, it holds that:

\[
K'; w' \equiv S; L'; \cdot; H'; B'
\]

(6.46)

By inverting Rule S-EXP-RtCONF on 6.36:

\[
K'; H' \hookrightarrow H'
\]

(6.47)

By Lemma 11 on 6.35:

\[
S; L; X; H; B_0 \rightarrow^* S; L'; X; H'; B'
\]

(6.48)

Fact 6.36 becomes:

\[
S; L'; \cdot; H'; B' \xrightarrow{H', \Delta} K'; e_0'
\]

(6.49)

By inverting S-EXP-RtCONF on 6.49:

\[
L'; \cdot; B' \xrightarrow{H', \Delta} e_0'
\]

(6.50)

By applying Lemma 6 on 6.13 using 6.44:

\[
L_0; X_0; E_0 \xrightarrow{H', \Delta} E_0
\]

(6.51)

By applying rule S-STACK-CONS on 6.13 and 6.50:

\[
L'; (X_0; L_0, E_0); B' \xrightarrow{H', \Delta} E_0[e_0']
\]

(6.52)

By applying rule S-EXP-RtCONF on 6.2, 6.47 and 6.52:

\[
S; L'; X; H'; B' \xleftarrow{H, \Delta} K'; E_0[e_0']
\]

(6.53)

We prove (d), (e) and (f) by 6.48, 6.53 and 6.46, respectively.
Theorem 1 (Forward Simulation). If $R \xrightarrow{\Delta} R$, then:

(a) if $R$ is terminal, then there exists $R'$ s.t. $R \xrightarrow{} R'$ and $R' \xrightarrow{\Delta} R$

(b) if $R \xrightarrow{} R'$, then there exists $R'$ s.t. $R \xrightarrow{} R'$ and $R' \xrightarrow{\Delta} R'$

Proof. Part (a) is proven by use of by Lemma 10, and part (b) by Lemma 13.
2.2 Type Safety

Lemma 14 (Substitution Lemma). If

(a) \( \Gamma \vdash w : S \)
(b) \( \Gamma, x : S \vdash S \leq S' \)
(c) \( \Gamma, x : S' \vdash e : T \)

then

\( \Gamma \vdash [w/x] e : R, R \leq T \)

Proof. By induction on the derivation of the statement \( \Gamma, x : S \vdash e : T \).

Lemma 15 (Environment Substitution). If \( \Gamma_1, x : T, \Gamma_2 \vdash w : S \), then \( \Gamma_1, x : T, \Gamma_2 \vdash [z/x] w : [z/x] S \).

Proof. Straightforward.

Lemma 16 (Weakening Subtyping). If \( \Gamma \vdash S \leq T \), then \( \Gamma, x : R \vdash S \leq T \).

Proof. Straightforward.

Lemma 17 (Weakening Typing). If \( \Gamma \vdash e : T \), then for \( \Gamma' \supseteq \Gamma \), it holds that \( \Gamma' \vdash e : T \).

Proof. Straightforward.

Lemma 18 (Store Type). If \( \Sigma \vdash H, H (l) = O \) and \( \Sigma (l) = T \), then \( \Sigma \vdash H, O : S, T \leq S \).

Proof. Straightforward.

Lemma 19 (Method Body Type – Lemma A.3 from [3]). If

(a) \( \Gamma, z : T \vdash z \) has \( \text{def} \ m (\ z : T \} \{p \} : S = e \)
(b) \( \Gamma, z : T, z : T \vdash T \leq T \)

Then for some type \( S' \) it is the case that:

\( \Gamma, z : T, z : T \vdash e : S', \ S' \leq S \)

Proof. Straightforward.

Lemma 20 (Cast). If \( \Sigma \vdash H \) and \( \Gamma ; \Sigma \vdash l : S, S \leq T \), then \( \Gamma ; \Sigma \vdash H (l) : R, R \leq T \).

Proof. Straightforward.

Lemma 21 (Evaluation Context Typing). If \( \Gamma \vdash E[e] : T \), then for some type \( S \) it holds that \( \Gamma \vdash e : S \).

Proof. By induction on the structure of the evaluation context \( E \).

Lemma 22 (Evaluation Context Step Typing). If

\( \Gamma ; \Sigma \vdash E[e] : T, e : S \)

and for some expression \( e' \) and heap typing \( \Sigma' \supseteq \Sigma \) it holds that

\( \Gamma ; \Sigma' \vdash e' : S', \ S' \leq S \)

then

\( \Gamma ; \Sigma' \vdash E[e'] : T, \ S' \leq T \)

Proof. By induction on the structure of the evaluation context \( E \).

Lemma 23 (Selfification). If \( \Gamma, x : S \vdash S \leq T \) then \( \Gamma, x : S \vdash S \leq \text{self} (T, x) \).

Proof. Straightforward.

Lemma 24 (Existential Weakening). If \( \Gamma \vdash R \leq R' \) then \( \Gamma \vdash \exists x : R, T \leq \exists x : R', T \).
Proof. Straightforward.

**Lemma 25** (Boolean Facts). If

(a) $\Gamma \vdash x : T, T \leq \{ \nu : \text{bool} \mid \nu = \text{true} \}$
(b) $\Gamma, x \vdash e : S, S \leq T$

then

$\Gamma \vdash e : S, S \leq T$

Proof. Straightforward.

**Theorem 2** (Subject Reduction). If

(a) $\Gamma; \Sigma \vdash e : T$
(b) $K; e \rightarrow K'; e'$
(c) $\Sigma \vdash K.H$

then for some $T'$ and $\Sigma' \supseteq \Sigma$:

(d) $\Gamma; \Sigma' \vdash e' : T'$
(e) $\Gamma \vdash T' \preceq T$
(f) $\Sigma' \vdash H'$.

Proof. We proceed by induction on the structure of fact (b):

$K; e \rightarrow K'; e'$

We have the following cases:

- [RC-ECTX]: Fact (b) has the form:

  $K; E[e_0] \rightarrow K'; E[e'_0]$ (6.1)

  From (a):

  $\Gamma; \Sigma \vdash E[e_0] : T$ (6.2)

  By Lemma 21 on 6.2:

  $\Gamma; \Sigma \vdash e_0 : T_0$ (6.3)

  By inverting Rule RC-ECTX on 6.1:

  $K; e_0 \rightarrow K'; e'_0$ (6.4)

  By induction hypothesis, using 6.3, 6.4 and (c) we get:

  $\Gamma; \Sigma' \vdash e'_0 : T'_0$ (6.5)

  $\Gamma; \Sigma' \vdash T'_0 \preceq T_0$ (6.6)

  $\Sigma' \vdash K'.H$ (6.7)

  $\Sigma' \supseteq \Sigma$ (6.8)

  For some type $T'_0$ and heap $K'.H$.

  From 6.7 we prove (f).

  By Lemma 22 using 6.2, 6.3, 6.5, 6.6 and 6.8:

  $\Gamma; \Sigma' \vdash E[e'_0]: T', T' \preceq T$ (6.9)

  From 6.9 we prove (d) and (e).
• [R-FIELD]: Fact (b) has the form:

\[ K; l.h \rightarrow K; v \]  
(6.10)

By Fact (a) for \( e \equiv l.h \) we have:

\[ \Gamma; \Sigma \vdash l.h : T \]  
(6.11)

By inverting R-FIELD on 6.10:

\[ K.H (l) \equiv O = \{\text{proto: } l'; f : \bar{F}\} \]  
(6.12)

\[ f := v \in \bar{F} \]  
(6.13)

By inverting WF-HEAP-INST on (c) for location \( l \):

\[ \bar{F} \doteq \circ \bar{J} := \bar{\tau}_1, \bigcirc \bar{g} := \bar{\tau}_M \]  
(6.14)

\[ |\Sigma (l)| = C \]  
(6.15)

\[ \Gamma, z : C \vdash \text{fields} (z) = \circ \bar{T}; \bar{R}, \bigcirc \bar{g}; \bar{U} \]  
(6.16)

\[ \Sigma \vdash \tau_1 : T_1 \]  
(6.17)

\[ \Sigma \vdash \tau_M : T_M \]  
(6.18)

\[ \Gamma, z : C, \bar{z}_1 : \text{self} (T_1, z.\bar{J}) \vdash T_1 \leq \bar{R}, T_M \leq \bar{U}, \text{inv} (C, z) \]  
(6.19)

By applying RT-T-OBJ on 6.15, 6.14 and 6.17:

\[ \Gamma; \Sigma \vdash O : S' \]  
(6.20)

Where:

\[ S' \equiv \exists \bar{z}_1 ; T_1 , \{\nu : C \mid \nu.\bar{J} = \bar{z}_1 \land \text{inv} (C, \nu)\} \]  
(6.21)

By Lemma 18 using (c), 6.12 and 6.15:

\[ \Gamma \vdash S \leq S' \]  
(6.22)

Where:

\[ \Sigma (l) = S \]  
(6.23)

We examine cases on the typing statement 6.11:

• [T-FIELD-I]: Field \( h \) is an immutable field \( f_i \), so fact 6.11 becomes:

\[ \Gamma; \Sigma \vdash l.f_i : \exists z ; S . \text{self} (R_i, z.f_i) \]  
(6.24)

By inverting T-FIELD-I on 6.24:

\[ \Sigma \vdash l : S \]  
(6.25)

\[ \Gamma, z : S ; \Sigma \vdash z \text{hasImm} f_i ; R_i \]  
(6.26)

For a fresh \( z \).

Keeping only the relevant part of 6.17 and 6.19:

\[ \Gamma; \Sigma \vdash v_i : T_i \]  
(6.27)

\[ \Gamma, z : C, \bar{z}_1 : \text{self} (T_1, z.\bar{J}) ; \Sigma \vdash T_i \leq R_i \]  
(6.28)
By 6.27 we prove (d).

By Lemma 23 using 6.28 and picking \( z_i \) as the selfification variable:

\[
\Gamma, z : C, z_1 : \text{self } (T_1, z.f_i) ; \Sigma \vdash T_i \leq \text{self } (R_i, z_i)
\]  
(6.29)

For the above environment it holds that:

\[
[[\Gamma, z : C, z_1 : \text{self } (T_1, z.f_i) ; \Sigma]] \Rightarrow z_i = z.f_i
\]  
(6.30)

By \( \leq \text{-REFL} \) and By Lemma 23 using 6.30:

\[
\Gamma, z : C, z_1 : \text{self } (T_1, z.f_i) ; \Sigma \vdash \text{self } (R_i, z_i) \leq \text{self } (R_i, z_i), z.f_i)
\]  
(6.31)

By simplifying 6.31 using \( \leq \text{-TRANS} \) on 6.29 and 6.31 we get:

\[
\Gamma, z : C, z_1 : \text{self } (T_1, z.f_i) ; \Sigma \vdash T_i \leq \text{self } (R_i, z.f_i)
\]  
(6.32)

By 6.32 it also holds that:

\[
\Gamma, z : \exists z_1 : \text{self } (T_1, z.f_i) ; \Sigma \vdash T_i \leq \text{self } (R_i, z.f_i)
\]  
(6.33)

By 6.33 it also holds that:

\[
\Gamma, z : \exists z_1 : T_1, \text{self } (C, z_1) \vdash T_i \leq \text{self } (R_i, z.f_i)
\]  
(6.34)

By expanding 6.34 and 6.19:

\[
\Gamma, z : \exists z_1 : T_1, \{ \nu : C \mid \nu.f_i = z_1 \wedge \text{inv } (C, \nu) \} \vdash T_i \leq \text{self } (R_i, z.f_i)
\]  
(6.35)

By using 6.21 on 6.35:

\[
\Gamma, z : S' \vdash T_i \leq \text{self } (R_i, z.f_i)
\]  
(6.36)

By Lemma 16 using 6.36 and 6.22:

\[
\Gamma, z : S \vdash T_i \leq \text{self } (R_i, z.f_i)
\]  
(6.37)

From Rule \( \leq \text{-WITNESS} \) using 6.45:

\[
\Gamma \vdash T_i \leq \exists z : S. \text{self } (R_i, z.f_i)
\]  
(6.38)

Using 6.24, 6.17 and 6.38 we prove (e).

Heap \( K.H \) does not evolve so (f) holds trivially.

[\text{T-FIELD-M}]: Field \( h \) is a mutable field \( g_i \), so fact (a) becomes:

\[
\Gamma; \Sigma \vdash l.g_i : \exists z : S.V_i
\]  
(6.39)

By inverting \text{T-FIELD-M} on 6.39:

\[
\Gamma \vdash l : S
\]  
(6.40)

\[
\Gamma, l : S \vdash z \text{ hasMut } g_i : U_i
\]  
(6.41)

For a fresh \( z \).

Keeping only the relevant parts of 6.17 and 6.19:

\[
\Gamma \vdash v_i : T_i
\]  
(6.42)

\[
\Gamma, z : C, z_1 : \text{self } (T_1, z.f_i) \vdash T_i \leq U_i
\]  
(6.43)
By 6.42 we prove (d).

By similar reasoning as before and using 6.43 we get:

$$\Gamma, z : S' \vdash T_i \leq U_i$$  \hfill (6.44)

By Lemma 16 using 6.44 and 6.22:

$$\Gamma, z : S \vdash T_i \leq U_i$$  \hfill (6.45)

By Rule $\leq$-WITNESS using 6.45:

$$\Gamma \vdash T_i \leq \exists z : S. U_i$$  \hfill (6.46)

Using 6.39, 6.17 and 6.46 we prove (e).

Heap $K.H$ does not evolve so (f) holds trivially.

• [R-CALL]: Fact (b) has the form:

$$K; l.m (\tau) \rightarrow K; [\tau/\pi, l/\textbf{this}] e'$$  \hfill (6.47)

By (a) for $e \equiv l.m (\tau)$ we have:

$$\Gamma; \Sigma \vdash l.m (\tau) : \exists z : T. \exists \pi : \overline{T}. S$$  \hfill (6.48)

By inverting T-INV on 6.48:

$$\Gamma; \Sigma \vdash l : T_i, \overline{T}$$  \hfill (6.49)

$$\Gamma, z : T_i, \overline{T} \vdash z \text{ has } (\textbf{def } m (\pi : \overline{R}) \{p\} : S = e')$$  \hfill (6.50)

$$\Gamma, z : T, \overline{T} \vdash \overline{T} \leq \overline{R}$$  \hfill (6.51)

$$\Gamma, z : T, \overline{T} \vdash p$$  \hfill (6.52)

With fresh $z$ and $\overline{z}$.

By inverting R-CALL on 6.47:

$$\text{resolveMethod} (H, l, m) = (\textbf{def } m (\pi : \overline{R}) \{p\} : S = e)$$

$$\text{eval} (p) = \text{true}$$  \hfill (6.53)

Note that this has already been substituted by $l$ in $S$ and $p$.

By Lemma 19 using 6.50 and 6.51:

$$\Gamma, z : T, \overline{T} \vdash e' : S', S' \leq S$$  \hfill (6.55)

By 6.55 we prove (d).

By Rule $\leq$-WITNESS using 6.55:

$$\Gamma \vdash S' \leq \exists z : T. \exists \pi : \overline{T}. S$$  \hfill (6.56)

By Lemma 14 using 6.49, 6.51 and 6.55:

$$\Gamma \vdash [\pi/\tau, l/\textbf{this}] e' : U, U \leq S'$$  \hfill (6.57)

By Rule $\leq$-TRANS on 6.55 and 6.57:

$$\Gamma \vdash U \leq \exists z : T. \exists \pi : \overline{T}. S$$  \hfill (6.58)

By 6.58 we prove (e).

Heap $K.H$ does not evolve so (f) holds trivially.
• [R-CAST]: Fact (b) has the form: 

\[ K; l \text{ as } T \rightarrow K; l \]

By (a) for \( e \equiv l \text{ as } T \) we have:

\[ \Gamma; \Sigma \vdash l \text{ as } T : T \quad (6.59) \]

By inverting T-CAST on 6.59:

\[ \Gamma; \Sigma \vdash l : S \]
\[ \Gamma \vdash T \]
\[ \Gamma \vdash S \preceq T \quad (6.62) \]

By 6.60 and 6.62 we get (d) and (e), respectively.

\( K.H \) does not evolve, which proves (f), given (b).

• [R-NEW]: Fact (c) has the form:

\[ K; \text{new } C (\nu) \rightarrow K'; l \]

By inverting R-NEW on 6.63:

\[ H (l_0) = \{ \text{name: } C ; \text{ proto: } l'_0 ; m: \bar{M} \} \]
\[ \text{fields}(S, C) = \bar{f}: T \]
\[ O = \{ \text{proto: } l_0 ; f: \bar{f} := \nu \} \]
\[ H' = H[l \mapsto O] \quad (6.67) \]

By (a) for \( e \equiv \text{new } C (\nu) \) we have:

\[ \Gamma; \Sigma \vdash \text{new } C (\nu) : R_0 \quad (6.68) \]

Where:

\[ R_0 \equiv \exists \bar{z}_1 : T_1 . \{ \nu : C | \nu.\bar{f} = \bar{z}_1 \land \text{inv}(C, \nu) \} \quad (6.69) \]

By inverting T-NEW on 6.68:

\[ \Gamma \vdash \nu : (T_1, T_R) \]
\[ \vdash \text{class}(C) \quad (6.71) \]
\[ \Gamma, z : C \vdash \text{fields}(z) = \circ \bar{f} : \bar{R}, \circ \bar{g} : \bar{U} \]
\[ \Gamma, z : C, \bar{z} : T, z.\bar{f} = \bar{z}_1 \vdash T_1 \leq \bar{R}, T_M \leq \bar{U}, \text{inv}(C, z) \quad (6.73) \]

For fresh \( z \) and \( \bar{z} \).

We choose a heap typing \( \Sigma' \), such that:

\[ \Sigma' = \Sigma[l \mapsto R_0] \]

Hence:

\[ \Sigma' (l) = R_0 \quad (6.74) \]

By applying Rule RT-T-LOC using 6.74:

\[ \Gamma; \Sigma' \vdash l : R_0 \]

Which proves (d).
By applying Rule RT-T-OBJ using 6.74, 6.66 and 6.70:
\[ K \vdash_{\Sigma} O : R_0 \] (6.75)

By \( \leq \)-ID we trivially get:
\[ \Gamma \vdash R_0 \leq R_0 \] (6.76)

Which proves (e).

By applying Rule WF-HEAP-INST on 6.66, 6.64, 6.74, 6.72, 6.70 and 6.73:
\[ \Sigma' \vdash K'.H \]

Which proves (f).

- [R-LETIN] Similar approach to case R-CALL.

- [R-DOTASGN]: Fact (b) has the form:
\[ K; l.g_i \leftarrow v' \longrightarrow K'; v' \] (6.77)

By inverting Rule R-DOTASGN on 6.77:
\[ H' = K.H[l \mapsto K.H(l)[g_i \mapsto v']] \] (6.78)

From (a) for \( e \equiv l.g_i \leftarrow v' \):
\[ \Gamma; \Sigma \vdash l.g_i \leftarrow v' : T' \] (6.79)

By inverting Rule T-ASGN on 6.79:
\[ \Gamma; \Sigma \vdash l : T_l, v' : T' \] (6.80)
\[ \Gamma, z : [T_l]; \Sigma \vdash z \text{ hasMut } g_i ; U_i, T' \leq U_i \] (6.81)

For a fresh \( z \).

By 6.80 and \( \leq \)-REFL we prove (d) and (e).

By inverting RT-T-LOC on 6.80:
\[ \Sigma (l) = T_l \] (6.82)

By inverting WF-HEAP-INST on (c) for location \( l \) and using 6.82:
\[ O \doteq \{ \text{proto: } l'; f: \bar{F} \} \] (6.83)
\[ \bar{F} \doteq \circ f := \tau_1, \color{red}{\color{black} \Box \bar{g}} := \tau_M \] (6.84)
\[ |\Sigma (l)| = C \] (6.85)
\[ \Gamma, z : C \vdash \text{fields} (z) = \circ \bar{F}; \bar{R}, \color{red}{\color{black} \Box \bar{g}}; \bar{U} \] (6.86)
\[ \Sigma \vdash \tau_1 : T_1 \] (6.87)
\[ \Sigma \vdash \tau_M : T_M \] (6.88)
\[ \Gamma, z : C, \tau_1 : \text{self} (T_l, z, \bar{f}) \vdash T_1 \leq \bar{R}, T_M \leq \bar{U}, \text{ inv} (C, z) \] (6.89)

Fact 6.78 becomes:
\[ H' = K.H[l \mapsto O'] \] (6.90)
\[ O' = \{ \text{proto: } l'; f: \bar{F}' \} \] (6.91)
\[ \bar{F}' = \circ f := \tau_1', \color{red}{\color{black} \Box \bar{g}} := \tau_M' \] (6.92)
\[ \tau_M' = \tau_{M\ldots i-1}, \tau_{M,i}, \tau_{M,i+1} \ldots \] (6.93)
Also by \(6.80\) and \(6.88\) it holds that:

\[
\Sigma \vdash \pi_M' : (T_{M_{i-1}}, T', T_{M_{i+1}}) \tag{6.94}
\]

By Lemma 16 on \(6.81\):

\[
\Gamma, z : C, \pi_1 : \text{self } (T_{I_1}, z, \bar{f}) ; \Sigma \vdash T' \leq U_i \tag{6.95}
\]

By applying Rule WF-HEAP-INST on \(6.91, 6.92, 6.85, 6.86, 6.87, 6.94, 6.89\) and \(6.95\):

\[
\Sigma \vdash H'
\]

Which proves (f).

• [R-LETIF]: Assume \(c \equiv \text{true} \) (the case for false is symmetric).

Fact (b) has the form:

\[
K ; \text{letif} [\pi, \pi_1, \pi_2] \text{ (true)} \ ? u_1 : u_2 \text{ in } e \rightarrow K ; u_1 \langle [\pi_1/\pi] e \rangle \tag{6.96}
\]

By Rule T-CTX fact (a) has the form:

\[
\Gamma \vdash \text{letif} [\pi, \pi_1, \pi_2] \text{ (true)} \ ? u_1 : u_2 \text{ in } e : \exists \pi : S. R \tag{6.97}
\]

So type \(T\) has the form:

\[
T \equiv \exists \pi : S. R \tag{6.98}
\]

By inverting Rule T-CTX on (a):

\[
\Gamma \vdash \text{letif} [\pi, \pi_1, \pi_2] \text{ (true)} \ ? u_1 : u_2 \text{ in } \langle \rangle \triangleright S \tag{6.99}
\]

\[
\Gamma, \pi : S \vdash e : R \tag{6.100}
\]

By inverting Ryle T-LETIF on \(6.99\):

\[
\Gamma \vdash \text{true} : S, S \leq \text{bool} \tag{6.101}
\]

\[
\Gamma, z : S, z \vdash u_1 \triangleright \Gamma_1 \tag{6.102}
\]

\[
\Gamma, z : S, \neg z \vdash u_2 \triangleright \Gamma_2 \tag{6.103}
\]

\[
\Gamma, \Gamma_1 \vdash \Gamma_1 (\pi_1) \leq S \tag{6.104}
\]

\[
\Gamma, \Gamma_2 \vdash \Gamma_2 (\pi_2) \leq S \tag{6.105}
\]

\[
\Gamma \vdash S \tag{6.106}
\]

By Rule T-CST on true:

\[
\Gamma \vdash \text{true} : \{\nu : \text{bool} \mid \nu = \text{true}\} \tag{6.107}
\]

By Lemma 25 on \(6.101\) and \(6.102\):

\[
\Gamma \vdash u_1 \triangleright \Gamma_1 \tag{6.108}
\]

Environment \(\Gamma_1\) has the form:

\[
\Gamma_1 \equiv \pi_1 : \Gamma_1 (\pi_1), \pi'_1 : \Gamma_1 (\pi'_1) \tag{6.109}
\]

For some \(\pi'_1\).

By Lemma 15 using \(6.100\):

\[
\Gamma, \pi_1 : S \vdash [\pi_1/\pi] e : [\pi_1/\pi] R \tag{6.110}
\]
By Lemma 17 using 6.110:
\[ \Gamma, \bar{x}_1 : \bar{\Sigma}, \bar{x}_1' : \Gamma_1 (\bar{x}_1') \vdash [\bar{x}_1/\bar{x}] e : [\bar{x}_1/\bar{x}] R \quad (6.111) \]

By applying rule T-CTX on 6.108 and 6.111:
\[ \Gamma \vdash u \langle [\bar{x}_1/\bar{x}] e \rangle : \exists \bar{x}_1 : \Gamma_1 (\bar{x}_1). \exists \bar{x}_1' : \Gamma_1 (\bar{x}_1'). [\bar{x}_1/\bar{x}] R \quad (6.112) \]

Which proves (d).

Fact 6.112 can be rewritten as:
\[ \Gamma \vdash u \langle [\bar{x}_1/\bar{x}] e \rangle : \exists \bar{x}_1 : \Gamma_1 (\bar{x}_1). \exists \bar{x}_1' : \Gamma_1 (\bar{x}_1'). R \quad (6.113) \]

Applying Rule $\leq$-BIND using 6.113:
\[ \Gamma \vdash \exists \bar{x}_1 : \Gamma (\bar{x}_1). \exists \bar{x}_1' : \Gamma_1 (\bar{x}_1'). R \leq \exists \bar{x}_1 : \Gamma_1 (\bar{x}_1). R \quad (6.114) \]

By Lemma 24 on the right-hand side of 6.114:
\[ \Gamma \vdash \exists \bar{x}_1 : \Gamma_1 (\bar{x}_1). R \leq \exists \bar{x}_1 : S. R \quad (6.115) \]

By 6.113, 6.114 and 6.115, and using Rule $\leq$-TRANS we prove (e).

Heap $K.H$ does not evolve so (f) holds trivially.

\[ \Box \]

**Theorem 3** (Progress). If
(a) $\Gamma ; \Sigma \vdash e : T$,
(b) $\Sigma \vdash H$

then one of the following holds:

- $e$ is a value,
- there exist $e'$, $H'$ and $\Sigma' \supseteq \Sigma$ s.t. $\Sigma' \vdash H'$ and $H ; e \rightarrow H'; e'$.

**Proof.** We proceed by induction on the structure of derivation (a):

- [T-FIELD-I]
\[ \Gamma ; \Sigma \vdash e_0.f_i : \exists z : T_0. \text{self} (T, z, f_i) \quad (2.1) \]

By inverting T-FIELD-I on 2.1:
\[ \Gamma ; \Sigma \vdash e_0 : T_0 \quad (2.2) \]
\[ \Gamma, z : T_0 ; \Sigma \vdash z \text{hasLmm} f_i : T \quad (2.3) \]

By i.h. using 2.2 and (b) there are two possible cases on $e_0$:

- $[e_0 \equiv l_0]$ Statement 2.2 becomes:
\[ \Gamma ; \Sigma \vdash l_0 : T_0 \quad (2.4) \]

By (b) for location $l_0$:
\[ \Sigma \vdash H[l_0 \mapsto O] \quad (2.5) \]

Where:
\[ O \equiv \{\text{proto}: l_0'; f : F\} \quad (2.6) \]
By Lemma 18 using (b) and 2.5:

\[ \Sigma (l_0) = T_0 \]  
(2.7)

\[ \Gamma; \Sigma \vdash O: S_0, S_0 \leq T_0 \]  
(2.8)

By Lemma A.6 in [3] using 2.3 and 2.8:

\[ \Gamma; z : S_0; \Sigma \vdash z \text{ hasImm } f_i: T \]  
(2.9)

By applying Rule R-\textsc{Field} using 2.5, 2.6 and 2.9:

\[ H; l_0.f_i \rightarrow H; v_i \]

- \([\exists e_0' \text{ s.t. } H; e_0 \rightarrow H'; e_0']\] By applying Rule RC-\textsc{Ectx}:

\[ H; e_0.f_i \rightarrow H'; e_0'.f_i \]

- \([\text{T-\textsc{Field-M}}] \text{ Similar to previous case.}\]

- \([\text{T-\textsc{Inv}}, \text{T-\textsc{New}}] \text{ Similar to the respective case of CFJ [3].}\]

- \([\text{T-\textsc{Cast}}]:\]

\[ \Gamma; \Sigma \vdash e_0 \text{ as } T : T \]  
(2.10)

By inverting T-Cast on 2.10:

\[ \Gamma \vdash e_0 : S_0 \]  
(2.11)

\[ \Gamma; \Sigma \vdash T \]  
(2.12)

\[ \Gamma; \Sigma \vdash S_0 \not\leq T \]  
(2.13)

By i.h. using 2.11 and (b) there are two possible cases on \(e_0\):

- \([e_0 \equiv l_0]\) Statement 2.11 becomes:

\[ \Gamma; \Sigma \vdash l_0 : S_0 \]  
(2.14)

By Lemma 20 using (b) and 2.13:

\[ \Gamma; \Sigma \vdash H(l_0): R_0, R_0 \leq T \]  
(2.15)

From R-Cast using 2.15:

\[ H; l_0 \text{ as } T \rightarrow H; l_0 \]

- \([\exists e_0' \text{ s.t. } H; e_0 \rightarrow H'; e_0']\] By rule RC-\textsc{Ectx}:

\[ H; e_0 \text{ as } T \rightarrow H'; e_0' \text{ as } T \]

- \([\text{T-\textsc{Let}}], [\text{T-\textsc{Asgn}}], [\text{T-\textsc{If}}] \text{ These cases are handled in a similar manner.} \]

References


