ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC

ROSAEC MEMO
2010-009
September 9, 2010

Semantics Preservation Proof of an Unstaging Translation of Lisp-Like Multi-Staged Languages

Wontae Choi
Seoul National University
wtchoi@ropas.snu.ac.kr

Baris Aktemur
Ozyegin University
baris.aktemur@ozyegin.edu.tr

Kwangkeun Yi
Seoul National University
kwang@ropas.snu.ac.kr

September 9, 2010

Abstract

In this paper, we present an unstaging translation from multi-stage programs to record calculus programs and prove that the translation preserves semantics. That is, a translated program simulates every evaluation step of the original multi-staged program. Also, we prove that a translated program can be inverse translated to the original program. Thanks to those properties, the translation can serve as a basis of reasoning about multi-staged programs.

1 Notation

Notation 1. *The Kleene closure of a reduction* $\xrightarrow{a}$ *is denoted as* $\xrightarrow{a^*}$.

Notation 2. *We use* $\xrightarrow{a;b}$ *to denote sequential application of* $\xrightarrow{a}$ *and* $\xrightarrow{b}$.

Notation 3. *We use* $(a, b) = (a', b')$ *to denote both* $a = a'$ *and* $b = b'$ *holds.*

Notation 4. *(Set Restriction) We are going to use* $A|_B^-$ *to denote* $\{a | a \in A \wedge a \notin B\}$.

Notation 5. *The function update operator (+) is defined as follows.*

$$(f + \{x : a\})(y) = \begin{cases} a & \textit{if } y = x \\ f(y) & \textit{o.w.} \end{cases}$$

Notation 6. *To distinguish record calculus expressions from staged calculus expressions when necessary, we underline expression variables, as in* $\underline{e}$, *to stand for record calculus expressions.*

2 Languages

This section presents languages we are going to use.

2.1 Multi-Staged Language λ_S

The language λ_S is a typed, call-by-value λ-calculus with staging annotations and reference.

Syntax

Variable	$x, y, f \in Var_{\mathcal{S}}$
Constant	$i \in Const$
Location	$\ell \in Loc$
$Expr_{\mathcal{S}}$	$e ::= i \mid x \mid \lambda x.e \mid e\ e \mid \texttt{fix}\ fx.e$ $\mid \texttt{ref}\ e \mid !\ e \mid \ell \mid e := e$ $\mid \texttt{box}\ e \mid \texttt{unbox}\ e \mid \texttt{run}\ e$

The syntax of $\lambda_{\mathcal{S}}$ is given above. The language contains constants, variables, lambda abstraction, application, and fix-point operator `fix`. Syntactic constructs related to mutable references are the referencing and dereferencing operators, locations, and assignment. Finally, there are staging annotations: `box` is used to define code templates; a code template is said to be in the next stage. `unbox` is the escape operator that defines a "hole" inside a code template which is filled in with another code template. `box` and `unbox` operators can be arbitrarily nested. `run` executes a code template.

Definition 1. *(Depth and Stage) The depth of a $\lambda_{\mathcal{S}}$ expression e, denoted as $depth(e)$, is the maximum depth of nested* `unbox` *expressions that are not enclosed by* `box`*. An expression e is said to be at stage n if $depth(e) \leq n$.*

Operational Semantics

$\lambda_{\mathcal{S}}$ has a small-step, call-by-value, operational semantics. Evaluation rules of the language are presented in Figure 1 and substitution rules are given in Figure 2. The evaluation $M, e \xrightarrow{n} e', M'$ has the meaning that "the expression e, under store M, evaluates to e' and store M' at stage n."

Notice that operational semantics inhibit code templates with free variables from being demoted. Only closed code templates are allowed to be demoted to stage-0 expressions. The inhibition is reasonable in that every expressions pass staged type systems satisfy the condition.

2.2 The Record Calculus $\lambda_{\mathcal{R}}$

The language $\lambda_{\mathcal{R}}$ is a λ-calculus with record operations and mutable references. For simplicity, we do not allow arbitrary expressions to be used in record expressions; we allow variables and values only.

Syntax

Variable	ρ	$\in$	Var_P	record variables
	h	$\in$	Var_H	hole variables
	x, y, f	$\in$	$Var_X = Var_{\mathcal{S}}$	ordinary variables
	w	$\in$	$Var_{\mathcal{R}} = Var_X \cup Var_P \cup Var_H$	
Constant	$i \in Const$			
Location	$\ell \in Loc$			
Label	$\texttt{x}$	$\in$	$Label = \{\texttt{x} \mid x \in Var_X\}$	tt-fonted ordinary variables

Definitions

$$
\begin{array}{lll}
\textit{Value}^0 & v^0 ::= i \mid \lambda x.e \mid \texttt{fix}\ fx.e \mid \texttt{box}\ v^1 \mid \ell \\
\textit{Value}^n\ (n>0) & v^n ::= i \mid x \mid \lambda x.v^n \mid v^n v^n \mid \texttt{fix}\ fx.v^n \\
& \quad \mid \texttt{ref}\ v^n \mid !\,v^n \mid \ell \mid v^n := v^n \\
& \quad \mid \texttt{box}\ v^{n+1} \mid \texttt{unbox}\ v^{n-1}\ (n>1)
\end{array}
$$

$$
\textit{Stores}_{\mathcal{S}} \qquad M \in \textit{Label} \xrightarrow{\textit{fin}} \textit{Value}^0
$$

Operational Semantics

$$
\text{(APP)} \quad \frac{M, e_1 \xrightarrow{n} e_1', M'}{M, e_1\ e_2 \xrightarrow{n} e_1'\ e_2, M'} \qquad \frac{M, e \xrightarrow{n} e', M' \quad v \in \textit{Value}^n}{M, v\ e \xrightarrow{n} v\ e', M'}
$$

$$
M, (\lambda x.e)\ v \xrightarrow{0} [x \overset{0}{\mapsto} v]e, M
$$

$$
M, (\texttt{fix}\ fx.e)\ v \xrightarrow{0} [x \overset{0}{\mapsto} v][f \overset{0}{\mapsto} \texttt{fix}\ fx.e]e, M
$$

$$
\text{(REF)} \quad \frac{M, e \xrightarrow{n} e', M'}{M, \texttt{ref}\ e \xrightarrow{n} \texttt{ref}\ e', M'} \qquad \frac{v \in \textit{Value}^0}{M, \texttt{ref}\ v \xrightarrow{0} \ell, M + \{\ell : v\}}\ \mathsf{new}\ \ell
$$

$$
\text{(DER)} \quad \frac{M, e \xrightarrow{n} e', M'}{M, !\,e \xrightarrow{n} !\,e', M'} \qquad \frac{M(\ell) = v}{M, !\,\ell \xrightarrow{0} v, M}
$$

$$
\text{(ASG)} \quad \frac{M, e_1 \xrightarrow{n} e_1', M'}{M, e_1 := e_2 \xrightarrow{n} e_1' := e_2, M'} \qquad \frac{M, e \xrightarrow{n} e', M'}{M, v := e \xrightarrow{n} v := e', M'}
$$

$$
M, \ell := v \xrightarrow{n} v, M + \{\ell : v\}
$$

$$
\text{(BOX)} \quad \frac{M, e \xrightarrow{n+1} e', M'}{M, \texttt{box}\ e \xrightarrow{n} \texttt{box}\ e', M'}
$$

$$
\text{(RUN)} \quad \frac{M, e \xrightarrow{n} e', M'}{M, \texttt{run}\ e \xrightarrow{n} \texttt{run}\ e', M'} \qquad \frac{v \in \textit{Value}^1 \quad FV_0(v) = \varnothing}{M, \texttt{run}\ (\texttt{box}\ v) \xrightarrow{0} v, M}
$$

$$
\text{(UNB)} \quad \frac{M, e \xrightarrow{n} e', M'}{M, \texttt{unbox}\ e \xrightarrow{n+1} \texttt{unbox}\ e', M'} \qquad \frac{v \in \textit{Value}^1}{M, \texttt{unbox}\ (\texttt{box}\ v) \xrightarrow{1} v, M}
$$

$$
\text{(ABS)} \quad \frac{M, e \xrightarrow{n+1} e', M'}{M, \lambda x.e \xrightarrow{n+1} \lambda x.e', M'}
$$

$$
\text{(FIX)} \quad \frac{M, e \xrightarrow{n+1} e', M'}{M, \texttt{fix}\ fx.e \xrightarrow{n+1} \texttt{fix}\ fx.e', M'}
$$

Figure 1: Operational Semantics of $\lambda_{\mathcal{S}}$.

$$
\begin{array}{lcl}
[x \overset{n}{\mapsto} v]i & = & i \\
[x \overset{0}{\mapsto} v]y & = & \begin{cases} v & \text{if } y = x \\ y & \text{o.w.} \end{cases} \\
[x \overset{n+1}{\mapsto} v]y & = & y \\
[x \overset{0}{\mapsto} v]\lambda y.e & = & \begin{cases} \lambda y.e & \text{if } y = x \\ \lambda y.[x \overset{0}{\mapsto} v]e & \text{o.w.} \end{cases} \\
[x \overset{n+1}{\mapsto} v]\lambda y.e & = & \lambda y.[x \overset{n+1}{\mapsto} v]e \\
[x \overset{0}{\mapsto} v]\texttt{fix}\ fy.e & = & \begin{cases} \texttt{fix}\ fy.e & \text{if } y = x \vee f = x \\ \texttt{fix}\ fy.e[x \overset{0}{\mapsto} v]e & \text{o.w.} \end{cases} \\
[x \overset{n+1}{\mapsto} v]\texttt{fix}\ fy.e & = & \texttt{fix}\ fy.[x \overset{n+1}{\mapsto} v]e \\
[x \overset{n}{\mapsto} v](e_1\ e_2) & = & [x \overset{n}{\mapsto} v]e_1\ [x \overset{n}{\mapsto} v]e_2 \\
[x \overset{n}{\mapsto} v]\texttt{ref}\ e & = & \texttt{ref}\ [x \overset{n}{\mapsto} v]e \\
[x \overset{n}{\mapsto} v]!\ e & = & !\ [x \overset{n}{\mapsto} v]e \\
[x \overset{n}{\mapsto} v]\ell & = & \ell \\
[x \overset{n}{\mapsto} v](e_1 := e_2) & = & [x \overset{n}{\mapsto} v]e_1 := [x \overset{n}{\mapsto} v]e_2 \\
[x \overset{n}{\mapsto} v]\texttt{box}\ e & = & \texttt{box}\ [x \overset{n+1}{\mapsto} v]e \\
[x \overset{n}{\mapsto} v]\texttt{run}\ e & = & \texttt{run}\ [x \overset{n}{\mapsto} v]e \\
[x \overset{n+1}{\mapsto} v]\texttt{unbox}\ e & = & \texttt{unbox}\ [x \overset{n}{\mapsto} v]e
\end{array}
$$

Figure 2: Substitutions for $\lambda_{\mathcal{S}}$.

$$
\begin{array}{lll}
\textit{Expr}_{\mathcal{R}} & e & ::= i \mid w \mid \lambda w.e \mid e\ e \mid \texttt{fix}\ fx.e \\
 & & \mid \texttt{ref}\ e \mid !\ e \mid \ell \mid e := e \\
 & & \mid r \mid r{\cdot}\texttt{x} \mid \texttt{let}\ w = e\ \texttt{in}\ e \\
\\
\textit{Value}_{\mathcal{R}} & v & ::= i \mid \lambda w.e \mid v_r \mid \ell \\
\textit{Record Value}_{\mathcal{R}} & v_r & ::= \{\} \mid v_r + \{\texttt{x} = v\} \\
\\
\textit{Store}_{\mathcal{R}} & M & ::= \varnothing \mid M + \{\ell : v\} \\
\\
\textit{Record}_{\mathcal{R}} & r & ::= \{\} \mid \rho \mid r + \{\texttt{x} = x\} \mid r + \{\texttt{x} = v\}
\end{array}
$$

The record language $\lambda_{\mathcal{R}}$ has constants (i), variables (x), lambda abstractions, applications, a fixpoint operator `fix`, and let-expressions. The constructs for mutable references are referencing (`ref`), dereferencing (!), locations (ℓ), and assignment. As for the record operations there is empty record ($\{\}$), record variables (ρ), and the record update operation $r + \{\texttt{x} = \_\}$. For field names (or labels) in records, we use variables written in teletype font.

We separate variables into three disjoint sets: ordinary variables Var_X (which are the same as variables of $\lambda_{\mathcal{S}}$), record variables Var_P, and hole variables Var_H. This syntactic distinction makes our presentation of the inverse translation easier. The operational semantics does not need to make a distinction; all variables are treated uniformly.

Operational Semantics

$\lambda_{\mathcal{R}}$ has a small-step, call-by-value operational semantics. The evaluation $M, e \xrightarrow{\mathcal{R}} e', M'$ means that "the expression e, under store M, evaluates to expression e' and store M'". The operational semantics of $\lambda_{\mathcal{R}}$ is mostly standard. Evaluation rules and the definition of values are given in Figure 3.

Operational Semantics

(APP)$_\mathcal{R}$

$$\frac{M, e_1 \xrightarrow{\mathcal{R}} e_1', M'}{M, e_1\ e_2 \xrightarrow{\mathcal{R}} e_1'\ e_2, M'} \qquad \frac{M, e \xrightarrow{\mathcal{R}} e', M'}{M, v\ e \xrightarrow{\mathcal{R}} v\ e', M'}$$

$$M, (\lambda w.e)\ v \xrightarrow{\mathcal{R}} [w \mapsto v]e, M$$

$$M, (\texttt{fix}\ fx.e)\ v \xrightarrow{\mathcal{R}} [x \mapsto v][f \mapsto \texttt{fix}\ fx.e]e, M$$

(LET)$_\mathcal{R}$

$$\frac{M, e_1 \xrightarrow{\mathcal{R}} e_1', M'}{M, \texttt{let}\ \ w = e_1\ \texttt{in}\ \ e_2 \xrightarrow{\mathcal{R}} \texttt{let}\ \ w = e_1'\ \texttt{in}\ \ e_2, M'}$$

$$M, \texttt{let}\ \ w = v\ \texttt{in}\ \ e \xrightarrow{\mathcal{R}} [w \mapsto v]e, M$$

(ACC)$_\mathcal{R}$

$$M, v_r \cdot \texttt{x} \xrightarrow{\mathcal{R}} v_r(\texttt{x}), M$$

(REF)$_\mathcal{R}$

$$\frac{M, e \xrightarrow{\mathcal{R}} e', M'}{M, \texttt{ref}\ e \xrightarrow{\mathcal{R}} \texttt{ref}\ e', M'} \qquad \frac{v \in \mathit{Value}_\mathcal{R}}{M, \texttt{ref}\ v \xrightarrow{\mathcal{R}} \ell, M + \{\ell : v\}}\ \textsf{new}\ \ell$$

(DER)$_\mathcal{R}$

$$\frac{M, e \xrightarrow{\mathcal{R}} e', M'}{M, !\ e \xrightarrow{\mathcal{R}} !\ e', M'} \qquad \frac{M(\ell) = v}{M, !\ \ell \xrightarrow{\mathcal{R}} v, M}$$

(ASG)$_\mathcal{R}$

$$\frac{M, e_1 \xrightarrow{\mathcal{R}} e_1', M'}{M, e_1 := e_2 \xrightarrow{\mathcal{R}} e_1' := e_2, M'} \qquad \frac{M, e \xrightarrow{\mathcal{R}} e', M'}{M, v := e \xrightarrow{\mathcal{R}} v := e', M'}$$

$$M, \ell := v \xrightarrow{\mathcal{R}} v, M + \{\ell : v\}$$

Record Lookup

$$v_r(\texttt{x}) = \begin{cases} v & \text{if } v_r = v_r' + \{\texttt{x} = v\} \\ v_r'(\texttt{x}) & \text{if } v_r = v_r' + \{\texttt{y} = \_\} \text{ and } \texttt{x} \neq \texttt{y} \end{cases}$$

Substitution

$$\begin{array}{lcl}
[w \mapsto e]i & = & i \\
[w_1 \mapsto e]w_2 & = & \begin{cases} e & \text{if } w_1 = w_2 \\ w_2 & \text{o.w.} \end{cases} \\
[w_1 \mapsto e_1]\lambda w_2.e_2 & = & \begin{cases} \lambda w_2.e_2 & \text{if } w_1 = w_2 \\ \lambda w_2.[w_1 \mapsto e_1]e_2 & \text{if } w_2 \notin FV(e_1) \end{cases} \\
[w_1 \mapsto e_1]\texttt{fix}\ fw_2.e_2 & = & \begin{cases} \texttt{fix}\ fw_2.e_2 & \text{if } w_1 = w_2 \\ \texttt{fix}\ fw_2.[w_1 \mapsto e_1]e_2 & \text{if } w_2 \notin FV(e_1) \end{cases} \\
[w \mapsto e](e_1\ e_2) & = & [w \mapsto e]e_1\ [w \mapsto e]e_2 \\
[w \mapsto e_1]\texttt{ref}\ e_2 & = & \texttt{ref}\ [w \mapsto e_1]e_2 \\
[w \mapsto e_1]!\ e_2 & = & !\ [w \mapsto e_1]e_2 \\
[w \mapsto e]\ell & = & \ell \\
[w \mapsto e](e_1 := e_2) & = & [w \mapsto e]e_1 := [w \mapsto e]e_2 \\
[w_1 \mapsto e]\texttt{let}\ \ w_2 = e_1\ \texttt{in}\ \ e_2 & = & \begin{cases} \texttt{let}\ \ w_2 = [w_1 \mapsto e]e_1\ \texttt{in}\ \ e_2 & \text{if } w_1 = w_2 \\ \texttt{let}\ \ w_2 = [w_1 \mapsto e]e_1\ \texttt{in}\ \ [w_1 \mapsto e]e_2 & \text{if } w_2 \notin FV(e_1) \end{cases} \\
[w \mapsto e]\{\} & = & \{\} \\
[w \mapsto e](r + \{\texttt{x} = x\}) & = & [w \mapsto e]r + \{\texttt{x} = [w \mapsto e]x\}
\end{array}$$

Figure 3: Operational Semantics of $\lambda_\mathcal{R}$.

Definitions

Environment	$r ::= \{\} \mid \rho \mid r{+}\{\mathtt{x}{=}x\}$
Environment Stack	$R ::= \bot \mid R, r$
Context	$\kappa ::= ((\lambda h.[\cdot])\ e) \mid ((\lambda h.\kappa)\ e)$
Context Stack	$K ::= \bot \mid K, \kappa$

Environment Lookup

$$r(\mathtt{x}) = \begin{cases} x & \text{if } r = r'{+}\{\mathtt{x}{=}x\} \\ r'(\mathtt{x}) & \text{if } r = r'{+}\{\mathtt{y}{=}\_\} \text{ and } \mathtt{x} \neq \mathtt{y} \\ \rho{\cdot}\mathtt{x} & \text{if } r = \rho \end{cases}$$

Term Translation

(TCON) $$R \vdash i \mapsto (i, \bot)$$

(TVAR) $$R, r \vdash x \mapsto (r(\mathtt{x}), \bot)$$

(TABS) $$\frac{R, r{+}\{\mathtt{x}{=}x\} \vdash e \mapsto (\underline{e}, K)}{R, r \vdash \lambda x.e \mapsto (\lambda x.\underline{e}, K)}$$

(TFIX) $$\frac{R, r{+}\{\mathtt{x}{=}x\}{+}\{\mathtt{f}{=}f\} \vdash e \mapsto (\underline{e}, K)}{R, r \vdash \mathtt{fix}\ fx.e \mapsto (\mathtt{fix}\ fx.\underline{e}, K)}$$

(TAPP) $$\frac{R \vdash e_1 \mapsto (\underline{e_1}, K_1) \quad R \vdash e_2 \mapsto (\underline{e_2}, K_2)}{R \vdash e_1\ e_2 \mapsto (\underline{e_1}\ \underline{e_2}, K_1 \bowtie K_2)}$$

(TBOX) $$\frac{R, \rho \vdash e \mapsto (\underline{e}, (K, \kappa))}{R \vdash \mathtt{box}\ e \mapsto (\kappa[\lambda\rho.\underline{e}], K)}\ \mathsf{new}\ \rho \qquad \frac{R, \rho \vdash e \mapsto (\underline{e}, \bot)}{R \vdash \mathtt{box}\ e \mapsto (\lambda\rho.\underline{e}, \bot)}\ \mathsf{new}\ \rho$$

(TUNB) $$\frac{R \vdash e \mapsto (\underline{e}, K)}{R, r \vdash \mathtt{unbox}\ e \mapsto (h\ r, (K, (\lambda h.[\cdot])\ \underline{e}))}\ \mathsf{new}\ h$$

(TRUN) $$\frac{R \vdash e \mapsto (\underline{e}, K)}{R \vdash \mathtt{run}\ e \mapsto (\mathtt{let}\ \ h = \underline{e}\ \mathtt{in}\ (h\{\}), K)}\ \mathsf{new}\ h$$

(TREF) $$\frac{R \vdash e \mapsto (\underline{e}, K)}{R \vdash \mathtt{ref}\ e \mapsto (\mathtt{ref}\ \underline{e}, K)}$$

(TDRE) $$\frac{R \vdash e \mapsto (\underline{e}, K)}{R \vdash !\ e \mapsto (!\ \underline{e}, K)}$$

(TLOC) $$R \vdash \ell \mapsto (\ell, \bot)$$

(TASG) $$\frac{R \vdash e_1 \mapsto (\underline{e_1}, K_1) \quad R \vdash e_2 \mapsto (\underline{e_2}, K_2)}{R \vdash e_1 := e_2 \mapsto (\underline{e_1} := \underline{e_2}, K_1 \bowtie K_2)}$$

Context Stack Merge Operator

$$\bot \bowtie K = K$$
$$K \bowtie \bot = K$$
$$(K_1, \kappa_1) \bowtie (K_2, \kappa_2) = (K_1 \bowtie K_2), (\kappa_1[\kappa_2])$$

Figure 4: Translation from $\lambda_{\mathcal{S}}$ to $\lambda_{\mathcal{R}}$.

3 Translation

The translation is presented in Figure 4. A translation judgment has the form $R \vdash e \mapsto (\underline{e}, K)$ with the meaning that "a $\lambda_{\mathcal{S}}$ expression e, under *environment stack* R, translates to the $\lambda_{\mathcal{R}}$ expression $\underline{e}$ and the *context stack* K."

Also, the translation for stores are defined as follows.

Definition 2. *(Store Translation)*

$$\varnothing \mapsto \varnothing \qquad \frac{M \mapsto \underline{M} \quad \{\} \vdash v \mapsto (\underline{v}, \bot)}{M \cup \{\ell : v\} \mapsto \underline{M} \cup \{\ell : \underline{v}\}}$$

The translation preserves semantics of the programs. That is, translated programs simulate original multi-staged programs. Specifically, one step reduction of a multi-staged program corresponds to one step reduction of the translated record calculus program followed by exhaustive administrative reductions.

Recall that a translation yields a pair of an expression and a context stack. This pair can be constructed into a single expression using a context closure operation:

Definition 3. *(Context Closure) Let e be a $\lambda_{\mathcal{R}}$ expression and K be a context stack. The context closure $K(e)$ is defined as follows.*

$$K(e) = \begin{cases} K'(\kappa[e]) & \text{if } K = (K', \kappa) \\ e & \text{if } K = \bot \end{cases}$$

Administrative reduction is defined as follows.

Definition 4. *(Admin Reduction) Administrative reduction of an expression is a congruence closure of the following two rules:*

$$\textit{(APP)} \quad (\lambda\rho.e)\ r \xrightarrow{\mathcal{A}} [\rho \mapsto r]e$$

$$\textit{(ACC)} \quad \frac{r \neq \rho}{r \cdot \mathtt{x} \xrightarrow{\mathcal{A}} r(\mathtt{x})}$$

Note that an administrative reduction may happen anywhere, even under lambdas. Also note that an admin reduction is "safe" to perform, in the sense that no side-effecting or non-terminating expression is eliminated by an admin reduction. It is also straightforward to check that admin reductions terminate.

Definition 5. *(Admin-normal form) An expression e is said to be in admin-normal form iff there does $\underline{\text{not}}$ exist an e' such that $e \xrightarrow{\mathcal{A}} e'$.*

Theorem 1. *(Simulation) Let e be a stage-n $\lambda_{\mathcal{S}}$ expression with no free variables and M be a store that contains values with no free variables such that $M, e \xrightarrow{n} e', M'$. Let $R \vdash e \mapsto (\underline{e}, K)$ and $R \vdash e' \mapsto (\underline{e'}, K')$. Also, $\underline{M}$ be the translation of M and $\underline{M'}$ be the translation of M'. Then $\underline{M}, K(\underline{e}) \xrightarrow{\mathcal{R};\mathcal{A}^*} K'(\underline{e'}), \underline{M'}$.*

Proof is given in Section 3.2.

3.1 Auxiliary Properties

Now we are going to introduce some basic properties of the translation and admin reduction, which are useful in proving the Theorem 1.

Notation 7. *(Abbreviations) We are going to omit $\bot$ when a stack is not empty. Also, we use $a_0 \ldots a_n$ to denote sequence $a_0, \ldots, a_n$. For example, a_0, a_1 denotes stack $\bot, a_0, a_1$ and $a_0 \ldots a_n$ denotes stack $\bot, a_0, \ldots, a_n$. Also $a, b_0 \ldots b_n$ denotes $\bot, a, b_0, \ldots, b_n$ and $a_0 \ldots a_n, b$ denotes $\bot, a_0, \ldots, a_n, b$.*

Notation 8. *(Length) We use $length(a)$ to denote the length of stack a. Notation $max(a)(b)$ denotes the largest number between a and b.*

During the translation, the length of a context stack cannot exceed the length of a environment stack.

Property 1. *Let e be a $\lambda_{\mathcal{S}}$ expression. If $r_0 \ldots r_n \vdash e \mapsto (\underline{e}, K)$, then $length(K) = depth(e)$.*

Proof. Proof by structural induction on e. □

3.1.1 Admin Normal Result

We first extend the definition of admin reductions to contexts.

Definition 6. *(Admin reduction of contexts) Administrative reduction of contexts is defined the same as administrative reduction of expressions.*

Definition 7. *(Admin reduction of context stacks) Exhaustive administrative reduction of context stacks is defined as follows:*

$$\frac{K \xrightarrow{\mathcal{A}^*} K' \quad \kappa \xrightarrow{\mathcal{A}^*} \kappa'}{K, \kappa \xrightarrow{\mathcal{A}^*} K', \kappa'} \qquad \bot \xrightarrow{\mathcal{A}^*} \bot$$

The following lemma states that there are no admin-reducible terms in the result of a translation.

Lemma 1. *Assume e is a stage-n $\lambda_{\mathcal{S}}$ expression. If $r_0 \ldots r_n \vdash e \mapsto (\underline{e}, K)$, then context stack K and $\lambda_{\mathcal{R}}$ expression $\underline{e}$ are admin-normal.*

Proof. Proof by structural induction on expression e. We are going to show interesting cases only. Other cases are straightforward inductions.

- Let $e = x$. By definition of the translation judgment, we have $r_0 \ldots r_n \vdash x \mapsto (r_n(\mathtt{x}), \bot)$. Depending on r_n, $r_n(\mathtt{x})$ can be either variable x or record access $\rho \cdot \mathtt{x}$ for some record variable ρ. In both cases, the expression is admin-normal. Hence we have the claim.
- Let $e = e_1\ e_2$. Assume, R is an environment stack. Then, by definition of the translation judgment, we have $R \vdash e_1\ e_2 \mapsto (\underline{e_1}\ \underline{e_2}, K_1 \bowtie K_2)$ where $R \vdash e_1 \mapsto (\underline{e_1}, K_1)$ and $R \vdash e_2 \mapsto (\underline{e_2}, K_2)$. By induction hypothesis, we have the fact that $\underline{e_1}$, $\underline{e_2}$, K_1 and K_2 are admin normal.

  Now we are going to show that both $\underline{e_1}\ \underline{e_2}$ and $K_1 \bowtie K_2$ are admin normal. Since $\bowtie$ operator does not introduce additional reducible terms, context stack $K_1 \bowtie K_2$ is admin normal by definition of administrative reduction. For expression $\underline{e_1}\ \underline{e_2}$, an applicable candidate is $(\lambda h.e)\ r \xrightarrow{\mathcal{A}} [h \mapsto r]e$. However, $\underline{e_2}$ cannot be a record by definition of the translation judgment. Therefore, by definition of admin reductions, $e_1\ e_2$ is admin-normal. Hence we have the claim.

□

3.1.2 Value Preservation

Stage-0 values of $\lambda_{\mathcal{S}}$ are translated to values of $\lambda_{\mathcal{R}}$.

Lemma 2. *Assume v is a stage-0 $\lambda_{\mathcal{S}}$ expression such that $v \in \mathit{Value}^0$. Then $r \vdash v \mapsto (\underline{v}, \bot)$ such that $\underline{v} \in \mathit{Value}_{\mathcal{R}}$ for any environment r.*

Proof. By a straightforward case analysis. □

3.1.3 Coincidence

We first define the free variables of staged expressions.

Definition 8. *Stage-0 free variables of stage-n $\lambda_{\mathcal{S}}$ expression e, $FV_n(e)$, is defined as follows.*

$$\begin{array}{ll}
FV_n(i) & = \varnothing \\
FV_n(x) & = \begin{cases} \{x\} & \text{if } n = 0 \\ \varnothing & o.w. \end{cases} \\
FV_n(\lambda x.e) & = \begin{cases} FV_n(e) \setminus \{x\} & \text{if } n = 0 \\ FV_n(e) & o.w. \end{cases} \\
FV_n(\mathtt{fix}\ fx.e) & = \begin{cases} FV_n(e) \setminus \{f, x\} & \text{if } n = 0 \\ FV_n(e) & o.w. \end{cases} \\
FV_n(e_1\ e_2) & = FV_n(e_1) \cup FV_n(e_2) \\
FV_n(\mathtt{ref}\ e) & = FV_n(e) \\
FV_n(!\,e) & = FV_n(e) \\
FV_n(\ell) & = \{\} \\
FV_n(e_1 := e_2) & = FV_n(e_1) \cup FV_n(e_2) \\
FV_n(\mathtt{box}\ e) & = FV_{n+1}(e) \\
FV_{n+1}(\mathtt{unbox}\ e) & = FV_n(e) \\
FV_n(\mathtt{run}\ e) & = FV_n(e)
\end{array}$$

Given a $\lambda_{\mathcal{S}}$ expression e, there exists a set of environment stacks which yield the same translation result.

Lemma 3. *Assume e is a stage-n $\lambda_{\mathcal{S}}$ expression with $FV_n(e) = \{x_1, \ldots x_m\}$. Also $r, r_1 \ldots r_n \vdash e \mapsto (\underline{e}_1, K_1)$ and $r', r_1 \ldots r_n \vdash e \mapsto (\underline{e}_2, K_2)$. If $r(\mathtt{x}_i) = r'(\mathtt{x}_i)$ for all $x_i \in \{x_1 \ldots x_m\}$, then $(\underline{e}_1, K_1) = (\underline{e}_2, K_2)$.*

Proof. By structural induction on the expression e. □

Lemma 4. *Assume e is a $\lambda_{\mathcal{S}}$ expression such that $e \in \mathit{Value}^{n+1}$. For any $R, R', r_1, \ldots, r_{n+1}$:*

$$R, r_1 \ldots r_{n+1} \vdash e \mapsto (\underline{e}, K) \iff R', r_1 \ldots r_{n+1} \vdash e \mapsto (\underline{e}, K)$$

Proof. By structural induction on the expression e. □

3.1.4 Free Variable Preservation

Lemma 5. *Assume e is a stage-n $\lambda_{\mathcal{S}}$ expression. If $r_0 \ldots r_n \vdash e \mapsto (\underline{e}, \kappa_p \ldots \kappa_1)$, then $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(r_n)$ and $\forall 1 \le i \le p : FV(\kappa_i)|^{-}_{Var_H} \subseteq FV(r_{n-i})$.*

Proof. By structural induction on e. We are going to show the interesting cases only. Other cases follow from the induction hypothesis.

- Let $e = x$. Assume $r_0 \ldots r_n \vdash x \mapsto (r_n(\mathtt{x}), \bot)$. We have two cases, depending on whether $r_n(\mathtt{x})$ is equal to x or $\rho{\cdot}\mathtt{x}$. In both cases, the claim holds by definition of *bound*.

- Let $e = \lambda x.e$. Assume $r_0 \dots r_n \vdash \lambda x.e \mapsto (\lambda x.\underline{e}, \kappa_p \dots \kappa_1)$ where $r_0 \dots r_n + \{\mathtt{x} = x\} \vdash e \mapsto (\underline{e}, \kappa_p \dots \kappa_1)$. By induction hypothesis, we have $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(r_n + \{\mathtt{x} = x\})$. Then, we have $FV(\underline{e})|^{-}_{Var_H} \setminus \{x\} \subseteq FV(r_n)$. Also, we have $FV(\lambda x.\underline{e}) = FV(\underline{e}) \setminus \{x\}$ and $FV(\underline{e}) \setminus \{x\}|^{-}_{Var_H} = FV(\underline{e})|^{-}_{Var_H} \setminus \{x\}$. Therefore, $FV(\lambda x.\underline{e})|^{-}_{Var_H} \subseteq FV(r_n)$. The other condition follows from I.H. Hence we have the claim.

- Let $e = \mathtt{box}\ e$ with $depth(e) = 0$. Assume $r_0 \dots r_n \vdash \mathtt{box}\ e \mapsto (\lambda\rho.\underline{e}, \bot)$ where $r_0 \dots r_n, \rho \vdash e \mapsto (\underline{e}, \bot)$ and ρ is a fresh variable. Since $\mathtt{box}\ e$ is a stage-n expression, e is a stage-$(n+1)$ expression. We have $FV(\lambda\rho.\underline{e})|^{-}_{Var_H} = FV(\underline{e}) \setminus \{\rho\}|^{-}_{Var_H}$. By induction hypothesis, $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(\rho) = \{\rho\}$. Hence, $FV(\lambda\rho.\underline{e})|^{-}_{Var_H} = \varnothing \subseteq FV(r_n)$. The other condition is trivial. Hence we have the claim.

- Let $e = \mathtt{box}\ e$ with $depth(e) > 0$. Assume $r_0 \dots r_n \vdash \mathtt{box}\ e \mapsto (\kappa_1[\lambda\rho.\underline{e}], \kappa_p \dots \kappa_2)$ where $r_0 \dots r_n, \rho \vdash e \mapsto (\underline{e}, \kappa_p \dots \kappa_1)$ and ρ is a fresh variable. Since $\mathtt{box}\ e$ is a stage-n expression, e is a stage-$(n+1)$ expression. We have, by definition of free variables, $FV(\kappa_1[\lambda\rho.\underline{e}])|^{-}_{Var_H} \subseteq (FV(\kappa_1) \cup FV(\lambda\rho.\underline{e}))|^{-}_{Var_H}$, which is $FV(\kappa_1)|^{-}_{Var_H} \cup (FV(\underline{e}) \setminus \{\rho\})|^{-}_{Var_H}$. By induction hypothesis, we have $FV(\kappa_1)|^{-}_{Var_H} \subseteq FV(r_n)$ and $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(\rho) = \{\rho\}$. Therefore, $FV(\kappa_1[\lambda\rho.\underline{e}])|^{-}_{Var_H} \subseteq FV(r_n)$. The other condition follows from I.H. Hence we have the claim.

- Let $e = \mathtt{unbox}\ e$. Assume $r_0 \dots r_{n+1} \vdash \mathtt{unbox}\ e \mapsto (h\ r_{n+1}, (\kappa_p \dots \kappa_1, (\lambda h.[\cdot])\ \underline{e}))$ where $r_0 \dots r_n \vdash e \mapsto (\underline{e}, \kappa_p \dots \kappa_1)$ and h is a fresh variable.

  We have to show (i)$FV(h\ r_{n+1})|^{-}_{Var_H} \subseteq FV(r_{n+1})$ (ii) $FV((\lambda h.[\cdot])\ \underline{e})|^{-}_{Var_H} \subseteq FV(r_n)$ (iii)$\forall 1 \leq i \leq p : FV(\kappa_i)|^{-}_{Var_H} \subseteq FV(r_{n-i})$. Item (i) trivially holds. By induction hypothesis, we have $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(r_n)$. By definition of free variables, $FV((\lambda h.[\cdot])\ \underline{e}) = FV(\underline{e})$. Thus, we have $FV((\lambda h.[\cdot])\ \underline{e})|^{-}_{Var_H} \subseteq FV(r_n)$, giving us item (ii). Finally, item (iii) is straightforward from the induction hypothesis. Hence we have the claim. □

3.2 Proof of the Theorem 1

We first prove several lemmas regarding substitution and variable capturing preservation. Then, we show a lemma corresponding to the simulation property, which leads to the proof of Theorem 1.

Lemma 6. *(Substitution Preservation) Assume e_1 is a stage-n $\lambda_{\mathcal{S}}$ expression, e_2 is a stage-0 $\lambda_{\mathcal{S}}$ expression with $FV_0(e_2) = \varnothing$. Let $r_0 \dots r_n \vdash e_1 \mapsto (\underline{e_1}, \kappa_p \dots \kappa_1)$ for $p \leq n$ and $\{\} \vdash e_2 \mapsto (\underline{e_2}, \bot)$ and x be a variables such that $x \in FV(r_0)$. Then*

- *If $n = 0$, then $r_0 \vdash [x \overset{0}{\mapsto} e_2]e_1 \mapsto ([x \mapsto \underline{e_2}]\underline{e_1}, \bot)$.*
- *If $n > 0$ and $n > p$, then $r_0 \dots r_n \vdash [x \overset{n}{\mapsto} e_2]e_1 \mapsto (\underline{e_1}, \kappa_p \dots \kappa_1)$.*
- *If $n > 0$ and $n = p$, then $r_0 \dots r_n \vdash [x \overset{n}{\mapsto} e_2]e_1 \mapsto (\underline{e_1}, (\kappa'_p, \kappa_{p-1} \dots \kappa_1))$ where $\kappa'_p = [x \mapsto \underline{e_2}]\kappa_p$.*

Proof. By structural induction on expression e_1. We are going to show interesting cases only. Other cases are straightforward inductions.

- Let $e_1 = y$ with $n > 0$. Assume $r_0 \dots r_n \vdash y \mapsto (r_n(\mathtt{y}), \bot)$. Since resulting context stack is empty, we have to show that the second claim holds. We have $[x \overset{n}{\mapsto} e_2]y = y$. Thus, $r_0 \dots r_n \vdash [x \overset{n}{\mapsto} e_2]y \mapsto (r_n(\mathtt{y}), \bot)$. Hence we have the claim.

- Let $e_1 = \texttt{box}\ e$ with $n = 0$ with $depth(e) = 0$. Assume $r \vdash \texttt{box}\ e \mapsto (\lambda\rho.\underline{e}, \bot)$ where $r, \rho \vdash e \mapsto (\underline{e}, \bot)$ and ρ is a fresh variable. Since $\texttt{box}\ e$ is a stage-0 expression, e is a stage-1 expression. Then, by induction hypothesis, we have $r, \rho \vdash [x \overset{1}{\mapsto} e_2]e \mapsto (\underline{e}, \bot)$. By definition of translation judgment, we have $r \vdash \texttt{box}\ ([x \overset{1}{\mapsto} e_2]e) \mapsto (\lambda\rho.\underline{e}, \bot)$. Also, by definition of substitution, we have $\texttt{box}\ ([x \overset{1}{\mapsto} e_2]e) = [x \overset{0}{\mapsto} e_2](\texttt{box}\ e)$. Therefore, $r \vdash [x \overset{0}{\mapsto} e_2](\texttt{box}\ e) \mapsto (\lambda\rho.\underline{e}, \bot)$. . . . (1)

  Recall that $r, \rho \vdash e \mapsto (\underline{e}, \bot)$. By Lemma 5, $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(\rho)$. Because $x \notin FV(\rho)$, we have $[x \mapsto \underline{e_2}]\underline{e} = \underline{e}$. Therefore, $[x \mapsto \underline{e_2}]\lambda\rho.\underline{e} = \lambda\rho.\underline{e}$. . . . (2)

  From (1) and (2), we have $r \vdash [x \overset{0}{\mapsto} e_2]\texttt{box}\ e \mapsto ([x \mapsto \underline{e_2}]\lambda\rho.\underline{e_2}, \bot)$. Hence, we have the claim.

- Let $e = \texttt{box}\ e$ with $n = 0$ and $depth(e) = 1$. Assume $r \vdash \texttt{box}\ e \mapsto (\kappa[\lambda\rho.\underline{e}], \bot)$ where $r, \rho \vdash e \mapsto (\underline{e}, \kappa)$ and ρ is a fresh variable. Since $\texttt{box}\ e$ is a stage-0 expression, e is a stage-1 expression. Then, by induction hypothesis, we have $r, \rho \vdash [x \overset{1}{\mapsto} e_2]e \mapsto (\underline{e}, \kappa')$ where $\kappa' = [x \mapsto \underline{e_2}]\kappa$. By definition of translation judgment, we have $r \vdash \texttt{box}\ ([x \overset{1}{\mapsto} e_2]e) \mapsto (\kappa'[\lambda\rho.\underline{e}], \bot)$. Also, we have $[x \overset{0}{\mapsto} e_2](\texttt{box}\ e) = \texttt{box}\ ([x \overset{1}{\mapsto} e_2]e)$. Thus, we have $r \vdash [x \overset{0}{\mapsto} e_2](\texttt{box}\ e) \mapsto (\kappa'[\lambda\rho.\underline{e}], \bot)$. . . . (1)

  Recall that $r, \rho \vdash e \mapsto (\underline{e}, \kappa)$. By Lemma 5, $FV(\underline{e})|^{-}_{Var_H} \subseteq FV(\rho)$. Because $x \notin FV(\rho)$, we have $[x \mapsto \underline{e_2}]\underline{e} = \underline{e}$. Therefore, $[x \mapsto \underline{e_2}](\kappa[\lambda\rho.\underline{e}]) = \kappa'[\lambda\rho.\underline{e}]$. . . . (2)

  From (1) and (2), we have $r \vdash [x \overset{0}{\mapsto} e_2]\texttt{box}\ e \mapsto ([x \mapsto \underline{e_2}](\kappa[\lambda\rho.\underline{e}]), \bot)$. Hence we have the claim. □

Lemma 7. *Assume r and r' are environments. Then,* $[\rho \mapsto r'](r(\texttt{x})) \xrightarrow{\mathcal{A}^*} ([\rho \mapsto r']r)(\texttt{x})$.

Proof. By structural induction on r.

- Let $r = \{\}$. We have $[\rho \mapsto r'](\{\}(\texttt{x})) = \texttt{error}$ and $([\rho \mapsto r']\{\})(\texttt{x}) = \texttt{error}$ by definition.
- Let $r = \rho'$. If $\rho = \rho'$, then $[\rho \mapsto r'](\rho(\texttt{x})) = r' \cdot \texttt{x}$ and $(([\rho \mapsto r']\rho))(\texttt{x}) = r'(\texttt{x})$. By the definition of admin reductions, $r' \cdot \texttt{x} \xrightarrow{\mathcal{A}} r'(\texttt{x})$. If $\rho \neq \rho'$, then $[\rho \mapsto r']\rho'(\texttt{x}) = \rho' \cdot \texttt{x}$ and $([\rho \mapsto r']\rho')(\texttt{x}) = \rho' \cdot \texttt{x}$. Hence we have the claim.
- Let $r = r'' + \{\texttt{y} = y\}$. If $x = y$, then $[\rho \mapsto r']((r'' + \{\texttt{x} = x\})(\texttt{x})) = x$ and $([\rho \mapsto r'](r'' + \{\texttt{x} = x\}))(\texttt{x}) = x$. If $x \neq y$, then $[\rho \mapsto r']((r'' + \{\texttt{x} = x\})(\texttt{x})) = [\rho \mapsto r'](r''(\texttt{x}))$ and $([\rho \mapsto r'](r'' + \{\texttt{x} = x\}))(\texttt{x}) = ([\rho \mapsto r']r'')(\texttt{x})$. By induction hypothesis, we have $[\rho \mapsto r'](r''(\texttt{x})) \xrightarrow{\mathcal{A}^*} ([\rho \mapsto r']r'')(\texttt{x})$. Hence we have the claim. □

Definition 9. *(Substitution in Environment Stacks)*

$$[x \mapsto e]R = \begin{cases} ([x \mapsto e]R'), ([x \mapsto e]r) & \textit{if } R = R', r \\ \bot & \textit{if } R = \bot \end{cases}$$

Definition 10. *(Substitution in Contexts and Context Stacks)*

$$[x \mapsto e]K = \begin{cases} ([x \mapsto e]K'), ([x \mapsto e]\kappa) & \textit{if } K = K', \kappa \\ \bot & \textit{if } K = \bot \end{cases}$$

$$[x \mapsto e]\kappa = \begin{cases} (\lambda h.([x \mapsto e]\kappa'))\ ([x \mapsto e]e') & \textit{if } \kappa = (\lambda h.\kappa')\ e' \\ (\lambda h.[\cdot])\ ([x \mapsto e]e') & \textit{if } \kappa = (\lambda h.[\cdot])\ e' \end{cases}$$

Lemma 8. *(Variable Capturing Preservation) Assume e is a stage-n λ_S expression and S is a substitution such that $S = [\rho \mapsto r]$. Let $r_0 \ldots r_n \vdash e \mapsto (\underline{e}, K)$. Then, $S(r_0 \ldots r_n) \vdash e \mapsto (\underline{e}', K')$ such that $S\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ and $SK \xrightarrow{\mathcal{A}^*} K'$.*

Proof. By structural induction on e. We are going to show interesting cases only. Other cases are straightforward inductions.

- Let $e = x$. We have $r_0 \dots r_n \vdash x \mapsto (r_n(\mathtt{x}), \bot)$. Also, we have $S(r_0 \dots r_n) \vdash x \mapsto ((Sr_n)(\mathtt{x}), \bot)$ since $S(r_0 \dots r_n) = Sr_0 \dots Sr_n$. Then, by Lemma 7, we have $S(r_n(\mathtt{x})) \xrightarrow{\mathcal{A}^*} (Sr_n)(\mathtt{x})$. Hence we have the claim.
- Let $e = \mathtt{box}\ e$ with $depth(e) = 0$. Assume $r_0 \dots r_n \vdash \mathtt{box}\ e \mapsto (\lambda\rho'.\underline{e}, \bot)$ where $r_0 \dots r_n, \rho' \vdash e \mapsto (\underline{e}, \bot)$ and ρ' is a fresh variable. By induction hypothesis, we have $S(r_0 \dots r_n, \rho') \vdash e \mapsto (\underline{e}', \bot)$ such that $S\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$. Since ρ' is a fresh variable, we have $S(\lambda\rho'.\underline{e}) = \lambda\rho'.S\underline{e}$ and $S(r_0 \dots r_n) \vdash \mathtt{box}\ e \mapsto (\lambda\rho'.\underline{e}', \bot)$. Because admin reduction is a congruence closure, we have $S(\lambda\rho'.\underline{e}) \xrightarrow{\mathcal{A}^*} \lambda\rho'.\underline{e}'$. Hence we have the claim.
- Let $e = \mathtt{box}\ e$ with $depth(e) > 0$. Assume $r_0 \dots r_n \vdash \mathtt{box}\ e \mapsto (\kappa_1[\lambda\rho'.\underline{e}], \kappa_p \dots \kappa_2)$ where $r_0 \dots r_n, \rho' \vdash e \mapsto (\underline{e}, \kappa_p \dots \kappa_1)$, $p \leq n$ and ρ' is a fresh variable. By induction hypothesis, we have $S(r_0 \dots r_n, \rho') \vdash e \mapsto (\underline{e}', \kappa'_p \dots \kappa'_1)$ such that $S\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ and $S(\kappa_p \dots \kappa_1) \xrightarrow{\mathcal{A}^*} \kappa'_p \dots \kappa'_1$. Since ρ' is a fresh variable, we have $S(\lambda\rho'.\underline{e}) = \lambda\rho'.S\underline{e}$ and $S(\kappa_1[\lambda\rho'.\underline{e}]) = (S\kappa_1)[\lambda\rho'.S\underline{e}]$ and $S(r_0 \dots r_n) \vdash \mathtt{box}\ e \mapsto (\kappa'_1[\lambda\rho'.\underline{e}'], \kappa'_p \dots \kappa'_2)$. Because admin reduction is a congruence closure, we have $S(\kappa_1[\lambda\rho'.\underline{e}]) \xrightarrow{\mathcal{A}^*} \kappa'_1[\lambda\rho'.\underline{e}']$. Hence we have the claim. □

Lemma 9. *(Simulation) Assume e is a stage-n $\lambda_{\mathcal{S}}$ expression with $FV_n(e) = \varnothing$ and M is a store that contains values with no free variables such that $M, e \xrightarrow{n} e', M'$. Also $r_0 \dots r_n \vdash e \mapsto (\underline{e}, K)$ and $r_0 \dots r_n \vdash e' \mapsto (\underline{e}', K')$. Let $\underline{M}$ be the translation of M and $\underline{M}'$ be the translation of M'.*

- *If $K = \bot$, then $\underline{M}, \underline{e} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}', \underline{M}'$ and $K' = \bot$.*
- *If $K = \kappa_n \dots \kappa_1$, where $n > 0$, we have four cases depending on κ_n.*
  - *If $\kappa_n = (\lambda h.\kappa)\ \underline{e}_h$ for some h, κ and $\underline{e}_h$ where $\underline{e}_h \in Value_{\mathcal{R}}$, then $[h \mapsto \underline{e}_h]\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ and $M = M'$, and $[h \mapsto \underline{e}_h](\kappa\kappa_{n-1} \dots \kappa_1) \xrightarrow{\mathcal{A}^*} K'$.*
  - *If $\kappa_n = (\lambda h.[\cdot])\ \underline{e}_h$ for some h and $\underline{e}_h$ where $\underline{e}_h \in Value_{\mathcal{R}}$, then $[h \mapsto \underline{e}_h]\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ and $M = M'$, and $[h \mapsto \underline{e}_h](\kappa_{n-1} \dots \kappa_1) \xrightarrow{\mathcal{A}^*} K'$.*
  - *If $\kappa_n = (\lambda h.\kappa)\ \underline{e}_h$ for some h, κ and $\underline{e}_h$ where $\underline{e}_h \notin Value_{\mathcal{R}}$, then $\underline{e} = \underline{e}'$ and $\exists \underline{e}'_h$ such that $\underline{M}, \underline{e}_h \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}'_h, \underline{M}'$ and $K' = (\lambda h.\kappa)\ \underline{e}'_h, \kappa_{n-1} \dots \kappa_1$.*
  - *If $\kappa_n = (\lambda h.[\cdot])\ \underline{e}_h$ for some h and $\underline{e}_h$ where $\underline{e}_h \notin Value_{\mathcal{R}}$, then $\underline{e} = \underline{e}'$ and $\exists \underline{e}'_h$ such that $\underline{M}, \underline{e}_h \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}'_h, \underline{M}'$ and $K' = (\lambda h.[\cdot])\ \underline{e}'_h, \kappa_{n-1} \dots \kappa_1$.*

Proof. By induction on evaluation of $e \xrightarrow{n} e'$. For given evaluation, we proceed by cases on the finally used evaluation. We are going to show interesting cases; other cases are either straightforward inductions or very similar to given cases. We write "assump." for assumption, "adm." for administrative reduction, "ctx." for context closure.

- Case (APP) (1) : Let $e = e_1\ e_2$.

  We have $\dfrac{M, e_1 \xrightarrow{n} e'_1, M'}{M, e_1\ e_2 \xrightarrow{n} e'_1\ e_2, M'}$ and $\dfrac{r_0 \dots r_n \vdash e_1 \mapsto (\underline{e_1}, K_1) \quad r_0 \dots r_n \vdash e_2 \mapsto (\underline{e_2}, K_2)}{r_0 \dots r_n \vdash e_1\ e_2 \mapsto (\underline{e_1}\ \underline{e_2}, K_1 \bowtie K_2)}$

  This case follows from the I.H. The following facts are used:

– Note that $length(\kappa_1) = depth(e_1)$ and $length(\kappa_2) = depth(e_2)$. Also, $depth(e_1\ e_2) = max(depth(e_1))(depth(e_2))$ and $length(K_1 \ \bowtie K_2) = max(length(e_1))(length(e_2))$ by definition. Hence $length(e_1\ e_2) = depth(K_1 \bowtie K_2)$.

– Assume K_1 and K_2 are context stacks. If $K_1 \xrightarrow{\mathcal{A}^*} K_1'$ and $K_2 \xrightarrow{\mathcal{A}^*} K_2'$, then $K_1 \bowtie K_2 \xrightarrow{\mathcal{A}^*} K1' \bowtie K_2'$.

– The outermost context in K_1 is also the outermost context in $K_1 \bowtie K_2$.

- Case (APP) (3) :

We have $\dfrac{v \in \mathit{Value}^0}{M, (\lambda x.e)\ v \xrightarrow{0} [x \overset{0}{\mapsto} v]e, M}$ and $\dfrac{\dfrac{r{+}\{\mathtt{x}{=}x\} \vdash e \mapsto (\underline{e}, \bot)}{r \vdash \lambda x.e \mapsto (\lambda x.\underline{v}, \bot)} \quad r \vdash v \mapsto (\underline{v}, \bot)}{r \vdash (\lambda x.e)\ v \mapsto ((\lambda x.\underline{e})\ \underline{v}, \bot)}$

Let $r \vdash [x \overset{0}{\mapsto} v]e \mapsto (\underline{e'}, \bot)$. (Since $n = 0$, the context stack must be $\bot$.) We want to show that $\underline{M}, (\lambda x.\underline{e})\ \underline{v} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e'}, \underline{M}$.

Also, $FV_0((\lambda x.e)v) = \varnothing$ implies $FV_0(e) \subseteq \{x\}$ because $FV_0(\lambda x.e) = FV_0(e) \setminus \{x\}$.

1. $\underline{v} \in \mathit{Value}_\mathcal{R}$ — by $v \in \mathit{Value}^0$ and Lemma 2.
2. $\underline{M}, (\lambda x.\underline{e})\ \underline{v} \xrightarrow{\mathcal{R}} [x \mapsto \underline{v}]\underline{e}, \underline{M}$ — by (1) and $(\mathrm{APP})_\mathcal{R}$
3. $r \vdash v \mapsto (\underline{v}, \bot)$ — by assump.
4. $FV_0(v) = \varnothing$, — by assump.
5. $\{\} \vdash v \mapsto (\underline{v}, \bot)$ — by (3),(4) and Lemma 3
6. $r{+}\{\mathtt{x}{=}x\} \vdash e \mapsto (\underline{e}, \bot)$ — by assump.
7. $r{+}\{\mathtt{x}{=}x\} \vdash [x \overset{0}{\mapsto} v]e \mapsto ([x \mapsto \underline{v}]\underline{e}, \bot)$. — by (5), (6) and Lemma 6
8. $FV_0(e) \subseteq \{x\}$ — by assump.
9. $FV_0([x \mapsto v]e) = \varnothing$ — by (8)
10. $r \vdash [x \overset{0}{\mapsto} v]e \mapsto ([x \mapsto \underline{v}]\underline{e}, \bot)$ — by (7), (9) and Lemma 3

So, we want to show that $\underline{M}, (\lambda x.\underline{e})\ \underline{v} \xrightarrow{\mathcal{R};\mathcal{A}^*} [x \mapsto \underline{v}]\underline{e}, \underline{M}$. This is immediately obtained from (2) and the fact that $[x \mapsto \underline{v}]\underline{e} \xrightarrow{\mathcal{A}^*} [x \mapsto \underline{v}]\underline{e}$.

- Case (BOX), $n = 0$.

We have

$$\frac{M, e \xrightarrow{1} e', M'}{M, \mathtt{box}\ e \xrightarrow{0} \mathtt{box}\ e', M'}$$

Expression e is at stage 1, since $\mathtt{box}\ e$ is a stage 0 expression. Because e reduces, its depth is 1; otherwise it would be a value and no reduction could be taken. So, for some κ_1, the translation is

$$\frac{r, \rho \vdash e \mapsto (\underline{e}, \kappa_1)}{r \vdash \mathtt{box}\ e \mapsto (\kappa_1[\lambda \rho.\underline{e}], \bot)}\ \textsf{fresh}\ \rho$$

Let $r, \rho \vdash e' \mapsto (\underline{e'}, K)$ and $r \vdash \mathtt{box}\ e' \mapsto (\underline{e}_b, K_b)$. (Subscript b refers to $\mathtt{box}$.) So, we want to show that $K_b = \bot$ and $\underline{M}, \kappa_1[\lambda \rho.\underline{e}] \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}_b, \underline{M'}$.

We now consider sub-cases depending on the outermost context κ_1.

– Let $\kappa_1 = (\lambda h.\kappa)\ \underline{e}_h$ where $\underline{e}_h \in \mathit{Value}_\mathcal{R}$. Also, $S = [h \mapsto \underline{e}_h]$. Then,

1. $M = M'$ by I.H.
2. $S\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e'}$ by I.H.
3. $S\kappa \xrightarrow{\mathcal{A}^*} K$ by I.H.
4. Length of K is 1. Let $\kappa = \kappa'$. by (3)
5. $\underline{e_b} = \kappa'[\lambda\rho.\underline{e'}]$ by (4) and assump.
6. $K_b = \bot$ by (4) and assump.

So, we want to show $\underline{M}, (\lambda h.\kappa[\lambda\rho.\underline{e}])\ \underline{e_h} \xrightarrow{\mathcal{R};\mathcal{A}^*} \kappa'[\lambda\rho.\underline{e'}], \underline{M}$.

7. $\underline{M}, (\lambda h.\kappa[\lambda\rho.\underline{e}])\ \underline{e_h} \xrightarrow{\mathcal{R}} S(\kappa[\lambda\rho.\underline{e}]), \underline{M}$ because $\underline{e_h} \in \mathit{Value}_{\mathcal{R}}$
8. $S(\kappa[\lambda\rho.\underline{e}]) = (S\kappa)[\lambda\rho.S\underline{e}]$ because h is unique
9. $(S\kappa)[\lambda\rho.S\underline{e}] \xrightarrow{\mathcal{A}^*} \kappa'[\lambda\rho.\underline{e'}]$ by (2), (3), (4), and adm.
10. $S(\kappa[\lambda\rho.\underline{e}]) \xrightarrow{\mathcal{A}^*} \kappa'[\lambda\rho.\underline{e'}]$ by (8) and (9)
11. $\underline{M}, (\lambda h.\kappa[\lambda\rho.\underline{e}])\ \underline{e_h} \xrightarrow{\mathcal{R};\mathcal{A}^*} \kappa'[\lambda\rho.\underline{e'}], \underline{M}$ by (7) and (10).

Thus, by (1), (5), and (11), we have $\underline{M}, \kappa_1[\lambda\rho.\underline{e}] \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e_b}, \underline{M'}$.

– Let $\kappa_1 = (\lambda h.\kappa)\ \underline{e_h}$ where $\underline{e_h} \notin \mathit{Value}_{\mathcal{R}}$. Then,

1. $\underline{e} = \underline{e'}$ by I.H.

$\exists \underline{e'_h}$ such that

2. $\underline{M}, \underline{e_h} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e'_h}, \underline{M'}$ by I.H.
3. $K = (\lambda h.\kappa)\ \underline{e'_h}$ by I.H.
4. $\underline{e_b} = (\lambda h.\kappa[\lambda\rho.\underline{e'}])\ \underline{e'_h}$ by (3) and assump.
5. $K_b = \bot$ by (3) and assump.

So, we want to show

$$\underline{M}, (\lambda h.\kappa[\lambda\rho.\underline{e}])\ \underline{e_h} \xrightarrow{\mathcal{R};\mathcal{A}^*} (\lambda h.\kappa[\lambda\rho.\underline{e'}])\ \underline{e'_h}, \underline{M'}$$

which is straightforward from (1) and (2).

– Proof for sub-cases with $\kappa_1 = (\lambda h.[\cdot])\ \underline{e_h}$ are similar to previous cases.

- Case (UNB) (1) : Let current stage be $n+1$. We have
$$\frac{M, e \xrightarrow{n} e', M'}{M, \texttt{unbox}\ e \xrightarrow{n+1} \texttt{unbox}\ e', M'}\ ,\quad \frac{R \vdash e \mapsto (\underline{e}, K)}{R, r_{n+1} \vdash \texttt{unbox}\ e \mapsto (h\ r_{n+1}, (K, (\lambda h.[\cdot])\ \underline{e}))}$$
and $\dfrac{R \vdash e' \mapsto (\underline{e'}, K')}{R, r_{n+1} \vdash \texttt{unbox}\ e' \mapsto (h\ r_{n+1}, (K', (\lambda h.[\cdot])\ \underline{e'}))}$.
where h is a fresh variable and $R = r_0 \ldots r_n$.

Since e reduces, its depth is n (otherwise it would be a value and no reduction could take place). Thus, let $K = \kappa_n \ldots \kappa_1$. Also, because $e \notin \mathit{Value}^n$, we have $\underline{e} \notin \mathit{Value}_{\mathcal{R}}$ by Lemma 2.

We have two cases depending on the stage number n.

– Let $n = 0$.

By I.H., we have $\underline{M}, \underline{e} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}', \underline{M}'$. We also have $K = \bot$ and $K' = \bot$. Therefore, $(\lambda h.[\cdot])\ \underline{e}$ becomes the outermost context in the translation of $\texttt{unbox}\ e$. Since $\underline{e} \notin \mathit{Value}_{\mathcal{R}}$ and $n + 1 > 0$, we have to show $\exists \underline{e}'_h$ such that the following conditions hold:

(i) $h\ r_1 = h\ r_1$

(ii) $\underline{M}, \underline{e} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}'_h, \underline{M}'$

(iii) $(\lambda h.[\cdot])\ \underline{e}' = (\lambda h.[\cdot])\ \underline{e}'_h$

Taking $\underline{e}'$ as the witness, i.e. taking $\underline{e}'_h = \underline{e}'$, satisfies the conditions trivially.

– For the case $n > 0$, the proof follows from the induction hypothesis.

- Case (UNB) (2) : We have

$$\frac{e \in \mathit{Value}^1}{M, \texttt{unbox box}\ e \xrightarrow{1} e, M} \quad \text{and} \quad \frac{\dfrac{r_0, \rho \vdash e \mapsto (\underline{e}, \bot)}{r_0 \vdash \texttt{box}\ e \mapsto (\lambda\rho.\underline{e}, \bot)}}{r_0, r_1 \vdash \texttt{unbox box}\ e \mapsto (h\ r_1, (\lambda h.[\cdot])\ \lambda\rho.\underline{e})}$$

where ρ and h are fresh variables. Also let $r_0, r_1 \vdash e \mapsto (\underline{e}', \bot)$.

Let $S = [h \mapsto \lambda\rho.\underline{e}]$. Since $\lambda\rho.\underline{e} \in \mathit{Value}_{\mathcal{R}}$ and $n = 1$, we have to show

(i) $S(h\, r_1) \xrightarrow{\mathcal{A}^*} \underline{e}'$

(ii) $S(\bot) = \bot$

(iii) $M = M$

Items (ii) and (iii) are trivial. We now show that item (i) holds.

1. $S(h\, r_1) = (\lambda\rho.\underline{e})\, r_1$
2. $(\lambda\rho.\underline{e})\ r_1 \xrightarrow{\mathcal{A}^*} [\rho \mapsto r_1]\underline{e}$ — by adm.
3. $r_0, \rho \vdash e \mapsto (\underline{e}, \bot)$ — by assump.
4. $r_0, r_1 \vdash e \mapsto (\underline{e}', \bot)$ — by assump.
5. $[\rho \mapsto r_1](r_0, \rho) = r_0, r_1$ — by def. of substitution and uniqueness of ρ
6. $[\rho \mapsto r_1]\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ — by (3),(4),(5) and Lemma 8
7. $S(h\, r_1) \xrightarrow{\mathcal{A}^*} \underline{e}'$ — by (1) and (6)

- Case (RUN) (2) : We have

$$\frac{e \in \mathit{Value}^1 \quad FV_0(e) = \varnothing}{M, \texttt{run box}\ e \xrightarrow{0} e, M} \quad \text{and} \quad \frac{\dfrac{r, \rho \vdash e \mapsto (\underline{e}, \bot)}{r \vdash \texttt{box}\ e \mapsto (\lambda\rho.\underline{e}, \bot)}}{r \vdash \texttt{run box}\ e \mapsto (\texttt{let}\ \ h = \lambda\rho.\underline{e}\ \texttt{in}\ \ h\ r, \bot)}$$

where ρ and h are fresh variables.

1. $\underline{M}, \texttt{let}\ \ h = \lambda\rho.\underline{e}\ \texttt{in}\ \ h\ \{\} \xrightarrow{\mathcal{R}} (\lambda\rho.\underline{e})\ \{\}, \underline{M}$ — by $(\text{LET})_{\mathcal{R}}$
2. $(\lambda\rho.\underline{e})\ \{\} \xrightarrow{\mathcal{A}} [\rho \mapsto \{\}]\underline{e}$ — by adm.
3. $r, \rho \vdash e \mapsto (\underline{e}, \bot)$ — by assump.

4. $r, \{\} \vdash e \mapsto (\underline{e}', \bot)$ such that $[\rho \mapsto \{\}]\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ by (3) and Lemma 8
5. $\{\} \vdash e \mapsto (\underline{e}', \bot)$ by (4) and Lemma 4, because $e \in \mathit{Value}$[1]
6. $(\lambda\rho.\underline{e})\ \{\} \xrightarrow{\mathcal{A}^*} \underline{e}'$ by (2) and (4)
7. $FV_0(e) = \varnothing$ by assump.
8. $r \vdash e \mapsto (\underline{e}', \bot)$ by (5), (7) and Lemma 3

Since $n = 0$, what we have to show is $\underline{M}, \texttt{let}\ h = \lambda\rho.\underline{e}\ \texttt{in}\ h\ \{\} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}', \underline{M}$. This is obtained from (1) and (6). Hence we have the claim. □

Proof of Theorem 1. By a case analysis on K. Recall that $M, e \xrightarrow{n} e', M'$, and we have $R \vdash e \mapsto (\underline{e}, K)$ and $R \vdash e' \mapsto (\underline{e}', K')$. What we want to show is $\underline{M}, K(\underline{e}) \xrightarrow{\mathcal{R};\mathcal{A}^*} K'(\underline{e}'), \underline{M}'$.

- Case $K = \bot$:

  By Lemma 9, we have $K = K' = \bot$. Therefore, $K(\underline{e}) = \underline{e}$ and $K'(\underline{e}') = \underline{e}'$. By Lemma 9, we also have $\underline{M}, \underline{e} \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}', \underline{M}'$. Hence we have the claim.

- Case $K = \kappa_n \dots \kappa_1$ where $n > 0$. We have four subcases based on κ_n. Then

  – Let $\kappa_n = (\lambda h.\kappa)\ \underline{e}_h$ for some h, κ and $\underline{e}_h$ where $\underline{e}_h \in \mathit{Value}_{\mathcal{R}}$.

  1. $[h \mapsto \underline{e}_h]\underline{e} \xrightarrow{\mathcal{A}^*} \underline{e}'$ by Lemma 9
  2. $M = M'$ by Lemma 9
  3. $[h \mapsto \underline{e}_h](\kappa\kappa_{n-1} \dots \kappa_1) \xrightarrow{\mathcal{A}^*} K'$ by Lemma 9

  Let $K' = \kappa'_n \dots \kappa'_1$. Then $K'(\underline{e}') = \kappa'_n[\kappa'_{n-1}[\dots \kappa'_1[\underline{e}']]]$ and $K(\underline{e}) = (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}_h$.

  4. $\underline{M}, (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}_h \xrightarrow{\mathcal{R}} [h \mapsto \underline{e}_h](\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]]), \underline{M}'$ by (APP)
  5. $[h \mapsto \underline{e}_h](\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]]) \xrightarrow{\mathcal{A}^*} \kappa'_n[\kappa'_{n-1}[\dots \kappa'_1[\underline{e}']]]$ by (1), (3) and uniqueness of h
  6. $\underline{M}, (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}_h \xrightarrow{\mathcal{R};\mathcal{A}^*} \kappa'_n[\kappa'_{n-1}[\dots \kappa'_1[\underline{e}']]], \underline{M}'$ by (4) and (5)

  Hence we have $\underline{M}, K(\underline{e}) \xrightarrow{\mathcal{R};\mathcal{A}^*} K'(\underline{e}'), \underline{M}'$.

  – Let $\kappa_n = (\lambda h.[\cdot])\ \underline{e}_h$ for some h and $\underline{e}_h$ where $\underline{e}_h \in \mathit{Value}_{\mathcal{R}}$. This case is similar to the previous case.

  – Let $\kappa_n = (\lambda h.\kappa)\ \underline{e}_h$ for some h, κ and $\underline{e}_h$ where $\underline{e}_h \notin \mathit{Value}_{\mathcal{R}}$. Then, $\exists \underline{e}'_h$ such that

  1. $\underline{M}, \underline{e}_h \xrightarrow{\mathcal{R};\mathcal{A}^*} \underline{e}'_h, \underline{M}'$ by Lemma 9
  2. $K' = (\lambda h.\kappa)\ \underline{e}'_h, \kappa_{n-1} \dots \kappa_1$ by Lemma 9
  3. $\underline{e} = \underline{e}'$ by Lemma 9
  4. $K(\underline{e}) = (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}_h$ by assump.
  5. $K(\underline{e}') = (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}'_h$ by (2) and (3)
  6. $\underline{M}, (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}_h \xrightarrow{\mathcal{R};\mathcal{A}^*} (\lambda h.\kappa[\kappa_{n-1}[\dots \kappa_1[\underline{e}]]])\underline{e}'_h, \underline{M}'$ by (1) and (6)

  Hence we have $\underline{M}, K(\underline{e}) \xrightarrow{\mathcal{R};\mathcal{A}^*} K'(\underline{e}'), \underline{M}'$ by (4), (5) and (6).

  – Let $\kappa_n = (\lambda h.[\cdot])\ \underline{e}_h$ for some h and $\underline{e}_h$ where $\underline{e}_h \notin \mathit{Value}_{\mathcal{R}}$. This case is similar to the previous case. □

Definitions

Hole Environment $H : Var_H \rightarrow Expr_{\mathcal{R}}$

Term Translation

(IVAR) $$H \vdash x \rightarrowtail x$$

(IACC) $$H \vdash \underline{e}\cdot\mathtt{x} \rightarrowtail x$$

(IABS) $$\frac{H \vdash \underline{e} \rightarrowtail e}{H \vdash \lambda x.\underline{e} \rightarrowtail \lambda x.e}$$

(IABS) $$\frac{H \vdash \underline{e} \rightarrowtail e}{H \vdash \mathtt{fix}\ fx.\underline{e} \rightarrowtail \mathtt{fix}\ fx.e}$$

(IAPP) $$\frac{H \vdash \underline{e_1} \rightarrowtail e_1 \quad H \vdash \underline{e_2} \rightarrowtail e_2 \quad \underline{e_1} \neq \lambda h.\underline{e} \quad \underline{e_2} \notin \mathit{Record}}{H \vdash \underline{e_1}\ \underline{e_2} \rightarrowtail e_1\ e_2}$$

(IREF) $$\frac{H \vdash \underline{e} \rightarrowtail e}{H \vdash \mathtt{ref}\ \underline{e} \rightarrowtail \mathtt{ref}\ e}$$

(IDER) $$\frac{H \vdash \underline{e} \rightarrowtail e}{H \vdash !\ \underline{e} \rightarrowtail !\ e}$$

(ILOC) $$H \vdash \ell \rightarrowtail \ell$$

(IASG) $$\frac{H \vdash \underline{e_1} \rightarrowtail e_1 \quad H \vdash \underline{e_2} \rightarrowtail e_2}{H \vdash \underline{e_1} := \underline{e_2} \rightarrowtail e_1 := e_2}$$

(ICTX) $$\frac{H \cup \{h : \underline{e}'\} \vdash \underline{e} \rightarrowtail e}{H \vdash ((\lambda h.\underline{e})\ \underline{e}') \rightarrowtail e}$$

(IBOX) $$\frac{H \vdash \underline{e} \rightarrowtail e}{H \vdash \lambda \rho.\underline{e} \rightarrowtail \mathtt{box}\ e}$$

(IUNB) $$\frac{H \vdash H(h) \rightarrowtail e}{H \vdash h\ r \rightarrowtail \mathtt{unbox}\ e}$$

(IRUN) $$\frac{H \vdash \underline{e} \rightarrowtail e}{H \vdash \mathtt{let}\ \ h = \underline{e}\ \mathtt{in}\ \ (h\ \{\}) \rightarrowtail \mathtt{run}\ e}$$

Figure 5: Inverse Translation from $\lambda_{\mathcal{R}}$ to $\lambda_{\mathcal{S}}$.

4 Inverse Translation

The definition of the inverse translation is in Figure 5. An inverse translation judgment is in the form $H \vdash \underline{e} \rightarrowtail e$ with the meaning that "under the *hole environment* H, the $\lambda_{\mathcal{R}}$ expression $\underline{e}$ translates to the $\lambda_{\mathcal{S}}$ expression e." A hole environment is a function from hole variables to expressions.

To make the connection between forward translation and inverse translation, we first define how to interpret context stacks as hole environments.

Definition 11. *(From Contexts to Hole Environments) Let K be a context stack. The operation $\overline{K}$ defines a hole environment in the following way:*

$$\overline{K} = \begin{cases} \varnothing & \text{if } K = \bot \\ \overline{K'} \cup \overline{\kappa} & \text{if } K = K', \kappa \end{cases}$$
$$\overline{\kappa} = \begin{cases} \{h : e\} & \text{if } \kappa = (\lambda h.[\cdot])\ e \\ \overline{\kappa'} \cup \{h : e\} & \text{if } \kappa = (\lambda h.\kappa')\ e \end{cases}$$

Lemma 10. *Assume $\underline{e}$ is a $\lambda_{\mathcal{R}}$ expression, κ is a context and H is a hole environment. If $H \cup \overline{\kappa} \vdash \underline{e} \rightarrowtail e$, then $H \vdash \kappa[\underline{e}] \rightarrowtail e$.*

Proof. Proof by induction on κ.

- Let $\kappa = (\lambda h.[\cdot])\ \underline{e}_h$.

  Assume, $H \cup \overline{(\lambda h.[\cdot])\ \underline{e}_h} \vdash \underline{e} \rightarrowtail e$ and $H \cup \{h : \underline{e}_h\} \vdash \underline{e} \rightarrowtail e'$. We have $\overline{(\lambda h.[\cdot])\ \underline{e}_h} = \{h : \underline{e}_h\}$. Therefore, we have $e' = e$. Also, by definition of the inverse translation, we have $H \vdash (\lambda h.\underline{e})\ \underline{e}_h \rightarrowtail e'$. Hence, we have the claim.

- Let $\kappa = (\lambda h.\kappa)\ \underline{e}_h$.

  Let $H \cup \overline{(\lambda h.\kappa)\ \underline{e}_h} \vdash \underline{e} \rightarrowtail e$. We have $\overline{(\lambda h.\kappa)\ \underline{e}_h} = \{h : \underline{e}_h\} \cup \overline{\kappa}$. Therefore, we have $H \cup \{h : \underline{e}_h\} \cup \overline{\kappa} \vdash \underline{e} \rightarrowtail e$. Let $H \cup \{h : \underline{e}_h\} \vdash \kappa[\underline{e}] \rightarrowtail e'$. Then, by induction hypothesis, we have $e' = e$. Also, by definition of the inverse translation, we have $H \vdash (\lambda h.\kappa[\underline{e}])\ \underline{e}_h \rightarrowtail e'$. Therefore, $H \vdash (\lambda h.\kappa[\underline{e}])\ \underline{e}_h \rightarrowtail e$. Hence we have the claim.

□

Theorem 2. *(Inversion) Let e be a $\lambda_{\mathcal{S}}$ expression and R be an environment stack. If $R \vdash e \mapsto (\underline{e}, K)$, then $H \vdash \underline{e} \rightarrowtail e$ for any H such that $\overline{K} \subseteq H$.*

Proof. Proof by structural induction of e. We are going to show only the interesting cases; other cases follow straightforward from the induction hypothesis.

- Let $e = x$.

  Assume $R, r \vdash x \mapsto (r(\mathtt{x}), \bot)$. We have two cases on $r(\mathtt{x})$, whether it is equal to x or $\rho{\cdot}\mathtt{x}$ for some ρ. For both cases, we have $H \vdash r(\mathtt{x}) \rightarrowtail x$ by definition of the inverse translation judgment. Hence we have the claim.

- Let $e = \mathtt{box}\ e$.

  Assume $R, \rho \vdash e \mapsto (\underline{e}, K)$, where ρ is fresh in e and K.

  We have two cases on translation of $\mathtt{box}\ e$, depending on whether K is equal to $\bot$ or not.

  – Assume $K = \bot$.

  We have $\dfrac{R, \rho \vdash e \mapsto (\underline{e}, \bot)}{R \vdash \mathtt{box}\ e \mapsto (\lambda\rho.\underline{e}, \bot)}$.

Note that resulting context stack is empty. Since $\overline{\bot} = \varnothing$, we have $H \vdash e \rightarrowtail \underline{e}$ for any H by induction hypothesis. Then, by definition of the translation judgment, we have $H \vdash \lambda\rho.\underline{e} \rightarrowtail \texttt{box}\ e$. Hence we have the claim.

– Assume $K = K', \kappa$.

We have $\dfrac{R, \rho \vdash e \mapsto (\underline{e}, (K', \kappa))}{R \vdash \texttt{box}\ e \mapsto (\kappa[\lambda\rho.\underline{e}], K')}$.

Assume $\overline{K'} \subseteq H$. We have $\overline{K', \kappa} = \overline{K'} \cup \overline{\kappa} \subseteq H \cup \overline{\kappa}$. Thus, we have $H \cup \overline{\kappa} \vdash \underline{e} \rightarrowtail e$ by induction hypothesis. Then, by definition of the inverse translation, we have $H \cup \overline{\kappa} \vdash \lambda\rho.\underline{e} \rightarrowtail \texttt{box}\ e$. By Lemma 10, $H \vdash \kappa[\lambda\rho.\underline{e}] \rightarrowtail \texttt{box}\ e$. Hence we have the claim.

- Let $e = \texttt{unbox}\ e$.

  Assume $\dfrac{R \vdash e \mapsto (\underline{e}, K)}{R, r \vdash \texttt{unbox}\ e \mapsto (h\ r, (K, (\lambda h.[\cdot])\ \underline{e}))}$

  where h is fresh in $\underline{e}$ and K. Also, $\overline{(K, (\lambda h.[\cdot])\ \underline{e})} \subseteq H$.

  We have $\overline{K, (\lambda h.[\cdot])\ \underline{e}_h} = \overline{K} \cup \{h : \underline{e}_h\}$. Then, we have $\overline{K} \subseteq \overline{(K, (\lambda h.[\cdot])\ \underline{e})} \subseteq H$. Together with the assumption, we have $\overline{K} \subseteq H$. Then, $H \vdash \underline{e} \rightarrowtail e$ by induction hypothesis. Also, we have $H(h) = \underline{e}$. Therefore, $H \vdash h\ r \rightarrowtail \texttt{unbox}\ e$ by definition of the inverse translation. Hence we have the claim.

□

5 Conclusion

We have proved that a multi-staged program can be translated to a record calculus program which simulates evaluation of the original program. We have also showed that a translated program can be inverse translated to the original multi-staged program.

The proof dependency graph is as follows :