Translation of Multi-Staged Language

Makoto Tatsuta

NII-2010-003E
Mmm 2010
Translation of Multi-Staged Language

Makoto Tatsuta
National Institute of Informatics

Abstract

This paper provides a translation of multi-staged language into a record calculus. It is based on the translations given by Aktemur and Yi. This paper gives simpler and detailed proofs of the soundness type theorem.

1 Introduction

Our contribution is a simpler proof of the type soundness of the translation.

We will investigate \([R, K]\) instead of \(K\). On the other hand \([1, 3]\) investigated \(K\). The notion \([R, K]\) drastically simplifies proofs of the type soundness.

2 Type Translation

2.1 Multi-Staged Language \(\lambda\)

Our multi-staged language is the same as that in \([3]\).

- Variables \(x, y, z, \ldots\).
- Constants \(c, \ldots\).
- Expressions \(e ::= c | x | \lambda x.e | ee | fix f x.e | box e | run e | unbox e\).

2.2 Types for \(\lambda\)

Our types are the same as \([2]\).

- Base types \(\alpha, \beta, \ldots\).
- Types and type contexts are defined inductively together:
  - Types \(A, B ::= \alpha | A \rightarrow A | \Pi(\Gamma_{\alpha})\).
  - Type contexts \(\Gamma, \Pi, \ldots\). A type context is a finite function from variables to types.
  - We will write \((x_1 : A_1, \ldots, x_n : A_n)\) for the type context \(\Gamma\) such that \(\text{Dom}(\Gamma) = \{x_1, \ldots, x_n\}\) and \(\Gamma(x_i) = A_i\).

Judgments \(\Gamma_0, \Gamma_1, \ldots, \Gamma_n \vdash e : A\).

Inference rules:

- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash e : A\) (Const) (if it is assumed)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash x : A\) (Var) \((\Gamma_n(x) = A)\)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash (x : A) \vdash e : B\) (Abs)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash \lambda x.e : A \rightarrow B\)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash e_1 : A \rightarrow B\) \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash e_2 : A\) (App)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash e : A\) (Box)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash box e : \Box(\Gamma \vdash A)\)
- \(\Gamma_0, \ldots, \Gamma_n \vdash \Gamma \vdash unbox e : A\) (Unbox)

The rules for run and fix are similar.
2.3 Record Calculus $\lambda_R$

Our record calculus is the same as that in [3].

Variables $x, y, z, \ldots$

Constants $c, \ldots$

Record variables $\rho, \ldots$

Record labels $x', y', z', \ldots$ (We use $x'$ for record labels instead of $x$ for clarity.)

Renaming Records $r ::= \{\} | \rho r + \{x' : x\}$.

Expressions $e ::= c[x]_1 \mid \lambda x.e \mid \rho e | ee | let \ x = e \ in \ fix \ f.x.e | r \cdot x'$.

2.4 Types for $\lambda_R$

Our types are standard for the simply typed lambda calculus with records.

Base types $\alpha, \beta, \ldots$

Types $A, B ::= \alpha \mid A \rightarrow A \mid \{x'_1 : A_1, \ldots, x'_n : A_n\}$.

Type contexts $\Gamma, \Pi, \ldots$ A type context is a finite function from variables to types.

Judgments $\Gamma \vdash e : A$.

Inference rules:

- $\frac{}{\Gamma \vdash e : A} (\text{Const})$ (if it is assumed)
- $\frac{}{\Gamma \vdash x : A} (\text{Var})$ ($\Gamma(x) = A$)
- $\frac{\Gamma \vdash \lambda x.e : A \rightarrow B \quad \Gamma \vdash e_1 : A}{\Gamma \vdash \lambda x.e : A \rightarrow B} (\text{Abs})$
- $\frac{\Gamma \vdash \lambda_1.e_1 : A \rightarrow B \quad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1 e_2 : B} (\text{App})$
- $\frac{\Gamma \vdash \{\} \vdash \{\} : \{x'_1 : A_1, \ldots, x'_n : A_n\}}{\Gamma \vdash \{x'_1 : A_1, \ldots, x'_n : A_n\} : A} (\text{REmp})$
- $\frac{\Gamma \vdash r + \{x'_{n+1} : e\} : \{x'_1 : A_1, \ldots, x'_{n+1} : A_{n+1}\}}{\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_{n+1} : A_{n+1}\}} (\text{RExt})$
- $\frac{\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_n : A_n\}}{\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_n : A_n\}} (\text{RAcc})$

The rules for let and fix are similar.

Remark. We will not use

- $\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_n : A_n\}$
- $\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_n : A_n\}$

2.5 Term Translation

Our translation for terms is the same as [3].

$\bot$ denotes the empty stack.

Renaming Record Stacks $R ::= \bot | R, r$ where $r$ is a renaming record. (We use a comma for the separator instead of a semicolon.)

Contexts $\kappa ::= \bot | (\lambda h. \kappa) e$.

Context stacks $K ::= \bot | K, \kappa$ (We use a comma for the separator instead of a semicolon.)

Translation judgment $\Gamma \vdash e \mapsto (e, K)$ where $e$ is an expression in $\lambda_S$ and $e$ is an expression in $\lambda_R$.

$R(x')$ is defined by $\rho(x') = \rho \cdot x'$, $(r + \{x' : x\})(x') = x$, and $(r + \{y' : y\})(x') = r(x)$ for $x' \neq y'$.

The context stack merge operator $K_1 \Join K_2$ is defined by $\bot \Join K_2 = K_2$, $K_1 \Join \bot = K_1$, and $(K_1, \kappa_1) \Join (K_2, \kappa_2) = ((K_1 \Join K_2), \kappa_1[\kappa_2])$.

We define $\Gamma \vdash e \mapsto (e, K)$ by the following inference rules.
Inference rules:

\[
\begin{align*}
R \vdash c \mapsto (c, \perp) & & R, r \vdash x \mapsto (r\langle x' \rangle), \perp \\
R, r \vdash \{x' : x\} \vdash e \mapsto (e, K) & & R \vdash e_1 \mapsto (e_1, K_1) & & R \vdash e_2 \mapsto (e_2, K_2) \\
R, r \vdash \lambda x.e \mapsto (\lambda x.e, K) & & R \vdash e_1 e_2 \mapsto (e_1, e_2, K_1 \bowtie K_2) \\
R, r \vdash e \mapsto (e, K) & & R, r \vdash \text{box } e \mapsto (\lambda \rho e, \perp) & & (\rho \text{ is fresh}) \\
R \vdash \text{unbox } e \mapsto (\text{hr}, (K, (\lambda h.[\cdot ]):2)) & & (h \text{ is fresh})
\end{align*}
\]

The rule for \(r\) is defined similarly to unbox.

2.6 Type Translation

We define the record type \((\Gamma)^r = \{x'_1 : A_1, \ldots, x'_n : A_n\}\) if \(\Gamma = (x_1 : A_1, \ldots, x_n : A_n)\).

Our translation maps the \(\lambda_s\)-type \(A\) to the \(\lambda_R\)-type \(\tilde{A}\). \(\tilde{A}\) is defined by

\[
\tilde{\alpha} = \alpha, \\
\tilde{A} \rightarrow B = \tilde{A} \rightarrow B, \\
\square(\Gamma \vdash A) = (\tilde{\Gamma})^r \rightarrow \tilde{A}
\]

where we define \(\tilde{\Gamma} = (x_1 : \tilde{A}_1, \ldots, x_n : \tilde{A}_n)\) if \(\Gamma = (x_1 : A_1, \ldots, x_n : A_n)\).

By our translation, a modal type is mapped to a functional type with a record input. For example,

\[
\text{Inference rules:}
\]

\[
K \rho \rightarrow \lambda \rho e \mapsto (\lambda \rho e, K) \quad (\rho \text{ is fresh})
\]

\[
\text{Proof.}
\]

\[
\text{Let } r \vdash e \mapsto (e, K) \quad (\rho \text{ is fresh})
\]

\[
\text{The rule for } r \text{ is defined similarly to unbox.}
\]

\subsection*{Lemma 2.1}

If \(\tilde{\Gamma}(\langle r \rangle) = (e_1, K_1) : A \rightarrow B\) and \(\tilde{\Gamma}(\langle r \rangle) = (e_2, K_2) : A\), then \(\tilde{\Gamma}(\langle r \rangle) = (e_1 e_2, K_1 \bowtie K_2) : B\).

\[
\text{Proof.}
\]

\[
\text{Let } \tilde{\Gamma}(\langle r \rangle) = ((\rho_0 : B_0, \Pi_0), \ldots, (\rho_n : B_n, \Pi_n))\text{. Let } K_j = (k'_0, \ldots, k'_n) \text{ and } k'_i = (\lambda h.[-])(e_i) \text{ for } j = 1, 2 \text{ and } 0 \leq i \leq n\text{. By the first assumption, we have } k'_0(\lambda \rho_0 \Pi_0, \ldots, k'_n(\lambda \rho_n \Pi_n, e)) \text{.}
\]

\[
\text{By the generation lemma, we have } h_0 : C'_0, \rho_0 : B_0, \Pi_0, \ldots, h'_0 : C'_n, \rho_0 : B_n, \Pi_n \vdash e_1 : A \rightarrow B \text{ for some } C'_0, \ldots, C'_n.
\]
Corollary 2.3 If \((x_1 : A_1, \ldots, x_n : A_n) \vdash e : A\) in \(\lambda_S\) and \(\rho + \{x'_1 : x_1, \ldots, x'_n : x_n\} : e \mapsto (e, K)\), then \(x_1 : A_1, \ldots, x_n : A_n \vdash K(e) : A\) in \(\lambda_R\).

Proof. Let \(\Gamma\) be the type context given by \((x_1 : A_1, \ldots, x_n : A_n)\). By Theorem 2.2, we have \(\vdash K(\lambda h e) : \{\}\) \(\mapsto \tilde{\lambda} \rightarrow\). Hence \(\rho = \{\}\) \(\tilde{\lambda} \vdash K(e) : A\). Since \(\rho\) does not appear in \(K(e)\), we have the claim. \(\square\)

Corollary 2.4 If \((x_1 : A_1, \ldots, x_n : A_n), \Gamma_1, \ldots, \Gamma_n \vdash e : A\) in \(\lambda_S\) and \(\rho + \{x'_1 : x_1, \ldots, x'_n : x_n\}, \rho_1, \ldots, \rho_n : e \mapsto (e, K)\), then \(x_1 : A_1, \ldots, x_n : A_n, \rho_1 : (\Gamma_1)' \mapsto (e, K) : A\) in \(\lambda_R\).
Proof. Let $\Pi = (x_1 : A_1, \ldots, x_n : A_n)$. By Theorem 2.2, we have $(\rho : \{\}, \tilde{\Pi}, \tilde{\Gamma}(\rho) \vdash (\epsilon, K) : \tilde{A})$. We have $\Gamma_i(\rho_i) = (\rho_i : (\tilde{\Gamma}_i)', \phi)$. Hence $\vdash \kappa_0[\rho\tilde{\Pi}\kappa_1[\rho_1, \ldots, \kappa_n[\rho_n, \epsilon]] : \{\} \rightarrow \tilde{\Pi} \rightarrow (\tilde{\Gamma}_1)', \ldots \rightarrow (\tilde{\Gamma}_n)' \rightarrow \tilde{A}$. Let $\kappa_i = (\lambda h_i : [\epsilon])e_i$. By the generation lemma, $h_0 : C_0, \rho : \{\}, \tilde{\Pi}, h_1 : C_1, \rho_1 : (\tilde{\Gamma}_1)', \ldots, h_{i-1} : C_{i-1}, \rho_{i-1} : (\tilde{\Gamma}_{i-1})' \vdash e_i : C_i$ for $0 \leq i \leq n$ and $h_0 : C_0, \rho : \{\}, \tilde{\Pi}, h_1 : C_1, \rho_1 : (\tilde{\Gamma}_1)', \ldots, h_n : C_n, \rho_n : (\tilde{\Gamma}_n)' \vdash e : \tilde{A}$ for some $C_0, \ldots, C_n$. Hence $\rho : \{\}, \tilde{\Pi}, \rho : (\tilde{\Gamma})' \vdash K(\epsilon) : \tilde{A}$. Since $\rho$ does not appear in $K(\epsilon)$, we have the claim. □

Remark. From the logical point of view, our type translation is just erasing the modality. Our type translation roughly corresponds to the translation from modal logic to usual logic by mapping $\Box A$ to $A$.

2.7 Examples

The examples are taken from [3].

Example 1.
\[
\begin{align*}
\bot & \vdash \text{box } ((\lambda x.x)y) \Rightarrow (\lambda \rho.(\lambda x.x)(\rho \cdot y'), \bot), \\
\vdash & \text{box } ((\lambda x.x)y) : \Box((y : A) \searrow A), \\
\vdash & \lambda \rho.(\lambda x.x)(\rho \cdot y') : \{y' : \tilde{A}\} \rightarrow \tilde{A}.
\end{align*}
\]

Example 2.
\[
\begin{align*}
\vdash & 1 : N, \\
\bot & \vdash \text{unbox } ((\lambda x.x)(\text{box } 1)) \Rightarrow ((\lambda h_1, h_1)((\lambda x.x)(\lambda \rho_2.1)), \bot), \\
\vdash & \text{unbox } ((\lambda x.x)(\text{box } 1)) : \Box N, \\
\vdash & (\lambda h_1, h_1)((\lambda x.x)(\lambda \rho_2.1)) : \{\} \rightarrow N.
\end{align*}
\]

Example 3.
\[
\begin{align*}
\bot & \vdash \text{unbox } ((\lambda x.x)(\text{box } x)) \Rightarrow ((\lambda h_1.x.h(\rho_1 + \{x' : x\}))(\lambda \rho_2.x', \bot), \\
\vdash & \text{unbox } ((\lambda x.x)(\text{box } x)) : \Box(A \rightarrow \tilde{A}), \\
\vdash & (\lambda h_1.x.h(\rho_1 + \{x' : x\}))(\lambda \rho_2.x') : \{\} \rightarrow \tilde{A} \rightarrow \tilde{A}.
\end{align*}
\]

Example 4. Our type translation works well with variable capturing.
\[
\begin{align*}
\bot & \vdash \text{unbox } ((\lambda x.x)(\text{box } y)) \Rightarrow ((\lambda x.(\lambda h_1.y.h(\rho_1 + \{y' : y\}) + \{y' : y\}))x)(\lambda \rho_2.x', \bot), \\
\vdash & \text{unbox } ((\lambda x.x)(\text{box } y)) : \Box(A \rightarrow \tilde{A}), \\
\vdash & (\lambda x.(\lambda h_1.y.h(\rho_1 + \{y' : y\} + \{y' : y\}))x)(\lambda \rho_2.x') : \{\} \rightarrow \tilde{A} \rightarrow \tilde{A}.
\end{align*}
\]

References