ISSN 1346-5597

National Institute of Informatics

NII Technical Report

Translation of Multi-Staged Language

Makoto Tatsuta

NII-2010-003E
Mmm 2010

URL: http://www.nii.ac.jp/ Tokyo 101-8430, JAPAN +81-3-4212-2000 FAX: +81-3-3556-1916

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 $[R, K](e) = \kappa_0[\lambda\rho_0\Gamma_0.\kappa_1[\lambda\rho_1\Gamma_1.\ldots\kappa_n[\lambda\rho_n\Gamma_n.e]\ldots]]$ instead of $K(e) = \kappa_0[\kappa_1[\ldots\kappa_n[e]\ldots]]$. On the other hand [1, 3] investigated $K(e)$. The notion $[R, K](e)$ drastically simplifies proofs of the type soundness.

2 Type Translation

2.1 Multi-Staged Language λ_S

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

Variables $x, y, z, \ldots$.

Constants $c, \ldots$.

Expressions $e ::= c|x|\lambda x.e|ee|\text{fix } fx.e|\text{box } e|\text{run } e|\text{unbox } e$.

2.2 Types for λ_S

Our types are the same as [2].

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

Types and type contexts are defined inductively together:

Types $A, B ::= \alpha|A \to A|\Box(\Gamma \rhd A)$.

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

The type context $\Gamma + (x : A)$ is defined by $(\Gamma + (x : A))(x) = A$ and $(\Gamma + (x : A))(y) = \Gamma(y)$ for $x \neq y$.

We will write $(x_1 : A_1, \ldots, x_n : A_n)$ for the type context Γ 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:

$$\frac{}{\Gamma_0, \ldots, \Gamma_n \vdash c : A}\ (Const) \quad \text{(if it is assumed)} \qquad \frac{}{\Gamma_0, \ldots, \Gamma_n \vdash x : A}\ (Var) \quad (\Gamma_n(x) = A)$$

$$\frac{\Gamma_0, \ldots, \Gamma_n + (x : A) \vdash e : B}{\Gamma_0, \ldots, \Gamma_n \vdash \lambda x.e : A \to B}\ (Abs) \qquad \frac{\Gamma_0, \ldots, \Gamma_n \vdash e_1 : A \to B \quad \Gamma_0, \ldots, \Gamma_n \vdash e_2 : A}{\Gamma_0, \ldots, \Gamma_n \vdash e_1 e_2 : B}\ (App)$$

$$\frac{\Gamma_0, \ldots, \Gamma_n, \Gamma \vdash e : A}{\Gamma_0, \ldots, \Gamma_n \vdash \text{box } e : \Box(\Gamma \rhd A)}\ (Box) \qquad \frac{\Gamma_0, \ldots, \Gamma_n \vdash e : \Box(\Gamma_{n+1} \rhd A)}{\Gamma_0, \ldots, \Gamma_{n+1} \vdash \text{unbox } e : A}\ (Unbox)$$

The rules for run and fix are similar.

2.3 Record Calculus λ_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 $\mathtt{x}$ for clarity.)

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

Expressions $e ::= c|x|\lambda x.e|\lambda\rho.e|ee|\text{let } x = e \text{ in } e|\text{fix } fx.e|r|r \cdot x'$.

2.4 Types for λ_R

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

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

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

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

Judgments $\Gamma \vdash e : A$.

Inference rules:

$$\frac{}{\Gamma \vdash c : A}\ (Const) \quad \text{(if it is assumed)} \qquad \frac{}{\Gamma \vdash x : A}\ (Var) \quad (\Gamma(x) = A)$$

$$\frac{\Gamma + (x : A) \vdash e : B}{\Gamma \vdash \lambda x.e : A \to B}\ (Abs) \qquad \frac{\Gamma \vdash e_1 : A \to B \quad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1 e_2 : B}\ (App)$$

$$\frac{}{\{\} : \{\}}\ (REmp) \qquad \frac{\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_n : A_n\} \quad \Gamma \vdash e : A_{n+1}}{\Gamma \vdash r + \{x'_{n+1} : e\} : \{x'_1 : A_1, \ldots, x'_{n+1} : A_{n+1}\}}\ (RExt)$$

$$\frac{\Gamma \vdash r : \{x'_1 : A_1, \ldots, x'_n : A_n\}}{\Gamma \vdash r \cdot x'_i : A_i}\ (RAcc)$$

The rules for let and fix are similar.

Remark. We will not use

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

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 $\kappa ::= [\cdot]|(\lambda h.\kappa)e$.

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

Translation judgment $R \vdash e \mapsto (\underline{e}, K)$ where e is an expression in λ_S and $\underline{e}$ is an expression in λ_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 $K_1 \bowtie K_2$ is defined by $\bot \bowtie K_2 = K_2$, $K_1 \bowtie \bot = K_1$, and $(K_1, \kappa_1) \bowtie (K_2, \kappa_2) = ((K_1 \bowtie K_2), \kappa_1[\kappa_2])$.

We define $R \vdash e \mapsto (\underline{e}, K)$ by the following inference rules.

Inference rules:

$$\frac{}{R \vdash c \mapsto (c, \bot)} \qquad \frac{}{R, r \vdash x \mapsto (r(x'), \bot)}$$

$$\frac{R, r + \{x' : x\} \vdash e \mapsto (\underline{e}, K)}{R, r \vdash \lambda x.e \mapsto (\lambda x.\underline{e}, K)} \qquad \frac{R \vdash e_1 \mapsto (\underline{e_1}, K_1) \quad R \vdash e_2 \mapsto (\underline{e_2}, K_2)}{R \vdash e_1 e_2 \mapsto (\underline{e_1}\ \underline{e_2}, K_1 \bowtie K_2)}$$

$$\frac{R, \rho \vdash e \mapsto (\underline{e}, (K, \kappa))}{R \vdash \text{box } e \mapsto (\kappa[\lambda\rho.\underline{e}], K)} \ (\rho \text{ is fresh}) \qquad \frac{R, \rho \vdash e \mapsto (\underline{e}, \bot)}{R \vdash \text{box } e \mapsto (\lambda\rho.\underline{e}, \bot)} \ (\rho \text{ is fresh})$$

$$\frac{R \vdash e \mapsto (\underline{e}, K)}{R, r \vdash \text{unbox } e \mapsto (hr, (K, (\lambda h.[\cdot])\underline{e}))} \ (h \text{ is fresh})$$

The rule for run is defined similarly to unbox.

2.6 Type Translation

We define the record type $(\Gamma)' = \{x_1' : A_1, \ldots, x_n' : A_n\}$ if $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$.

Our translation maps the λ_S-type A to the λ_R-type $\widetilde{A}$. $\widetilde{A}$ is defined by

$$\widetilde{\alpha} = \alpha,$$
$$\widetilde{A \to B} = \widetilde{A} \to \widetilde{B},$$
$$\widetilde{\Box(\Gamma \rhd A)} = (\widetilde{\Gamma})' \to \widetilde{A}$$

where we define $\widetilde{\Gamma} = (x_1 : \widetilde{A}_1, \ldots, x_n : \widetilde{A}_n)$ 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, $\Box\alpha$ is mapped to $\{\} \to \alpha$. The type $\Box((x : \beta) \to \alpha)$ is mapped to $\{x' : \beta\} \to \alpha$.

The renaming record $(\rho + \{x_1' : x_1, \ldots, x_{n+k}' : x_{n+k}\}) - \{x_1' : x_1, \ldots, x_n' : x_n\}$ is defined as $\rho + \{x_{n+1}' : x_{n+1}, \ldots, x_{n+k}' : x_{n+k}\}$.

We define type type context $\Gamma|s = \Gamma|_{\{x_1, \ldots, x_n\}}$ where $s = \{x_1' : x_1, \ldots, x_n' : x_n\}$.

We define the pair $\Gamma(\rho + s) = (\rho : ((\widetilde{\Gamma})' - s), \widetilde{\Gamma}|_s)$ consisting of the record variable type declaration $\rho : ((\widetilde{\Gamma})' - s)$ and the type context $\widetilde{\Gamma}|_s$.

$K(e)$ is defined by $\bot(e) = e$ and $(K, \kappa)(e) = K(\kappa[e])$.

We will write $\lambda\Gamma.e = \lambda x_1 \ldots x_n.e$ and $\Gamma \to B = A_1 \to \ldots \to A_n \to B$ when $\Gamma = (x_1 : A_1, \ldots, x_n : A_n)$. We assume a fixed order in variables.

$(\rho_0 : B_0, \Gamma_0), \ldots, (\rho_n : B_n, \Gamma_n) \vdash (e, K) : A$ is defined as $\vdash \kappa_0[\lambda\rho_0\Gamma_0.\kappa_1[\lambda\rho_1\Gamma_1.\ldots\kappa_n[\lambda\rho_n\Gamma_n.e]\ldots]] : B_0 \to \Gamma_0 \to B_1 \to \Gamma_1 \to \ldots \to B_n \to \Gamma_n \to A$ where $K = (\kappa_0^1, \ldots, \kappa_m^1)$, $\kappa_i = [\cdot]$ for $0 \leq i < n - m$, and $\kappa_{n-m+i} = \kappa_i^1$ for $0 \leq i \leq m$. The context stack $(\kappa_0, \ldots, \kappa_n)$ is obtained from $(\kappa_0^1, \ldots, \kappa_m^1)$ by padding the dummy context $[\cdot]$ to the left so that its length becomes $n + 1$.

We explain the meaning of $\Gamma(\rho + s)$. Let $r_0, \ldots, r_n \vdash e \mapsto (\underline{e}, K)$. Assume $r_i = \rho + s$ and $s = \{x_1' : x_1, \ldots, x_n' : x_n\}$. ρ refers to global type information given outside the box and s refers to variables bound locally in the box. Suppose $\Gamma_0, \ldots, \Gamma_n \vdash e : A$. Then the type context for level i is Γ_i. Type information for s in Γ_i is for bound variables in the box, and the rest in Γ_i is for global type information outside the box. They are given by the type context $\widetilde{\Gamma}_i|_s$ and the record typing $\rho : ((\widetilde{\Gamma}_i)' - s)$ respectively, which are defined by $\Gamma_i(\rho + s)$.

We explain the meaning of $(\rho_0 : B_0, \Gamma_0), \ldots, (\rho_n : B_n, \Gamma_n) \vdash (e, (\kappa_0, \ldots, \kappa_n)) : A$. Let $\kappa_i = (\lambda h_i.[\cdot])e_i$. e_{i+1} is the body of an unbox at level $i + 1$ and evaluated at level i by using $h_0, \ldots, h_{i-1}, \rho_i, \Gamma_i$. e is evaluated at level n by using $h_0, \ldots, h_n, \rho_n, \Gamma_n$. When we express this dependency by lambda abstraction, the expression becomes $\kappa_0[\lambda\rho_0\Gamma_0.\ldots\kappa_n[\lambda\rho_n\Gamma_n.e]\ldots]$.

We will use vector notation, for example, $\overrightarrow{\Gamma}$ for $\Gamma_0, \ldots, \Gamma_{n-1}$.

Lemma 2.1 *If $\overrightarrow{\Gamma(r)} \vdash (e_1, K_1) : A \to B$ and $\overrightarrow{\Gamma(r)} \vdash (e_2, K_2) : A$, then $\overrightarrow{\Gamma(r)} \vdash (e_1 e_2, K_1 \bowtie K_2) : B$.*

Proof. Let $\overrightarrow{\Gamma(r)} = ((\rho_0 : B_0, \Pi_0), \ldots, (\rho_n : B_n, \Pi_n))$. Let $K_j = (\kappa_0^j, \ldots, \kappa_n^j)$ and $\kappa_i^j = (\lambda h_i^j.[\cdot])e_i^j$ for $j = 1, 2$ and $0 \leq i \leq n$. By the first assumption, we have $\vdash \kappa_0^1(\lambda\rho_0\Pi_0.\ldots\kappa_n^1(\lambda\rho_n\Pi_n.e_1)\ldots) : B_0 \to \Pi_0 \to \ldots \to B_n \to \Pi_n \to A \to B$. By the generation lemma, we have $h_0^1 : C_0^1, \rho_0 : B_0, \Pi_0, \ldots, h_i^1 : C_i^1, \rho_i : B_i, \Pi_i \vdash e_{i+1}^1 : C_{i+1}^1$ and $h_0^1 : C_0^1, \rho_0 : B_0, \Pi_0, \ldots, h_n^1 : C_n^1, \rho_n : B_n, \Pi_n \vdash e_1 : A \to B$ for some $C_0^1, \ldots, C_n^1$.

Similarly the second assumption and the generation lemma give $h_0^2 : C_0^2, \rho_0 : B_0, \Pi_0, \ldots, h_i^2 : C_i^2, \rho_i : B_i, \Pi_i \vdash e_{i+1}^2 : C_{i+1}^2$ and $h_0^2 : C_0^2, \rho_0 : B_0, \Pi_0, \ldots, h_n^2 : C_n^2, \rho_n : B_n, \Pi_n \vdash e_2 : A$ for some $C_0^2, \ldots, C_n^2$. Hence $h_0^1 : C_0^1, h_0^2 : C_0^2, \rho_0 : B_0, \Pi_0, \ldots, h_n^1 : C_n^1, h_n^2 : C_n^2, \rho_n : B_n, \Pi_n \vdash e_1 e_2 : B$. Hence we have the claim. $\square$

Theorem 2.2 *If $\Gamma_0, \ldots, \Gamma_n \vdash e : A$ in λ_S and $r_0, \ldots, r_n \vdash e \mapsto (\underline{e}, K)$, then $\Gamma_0(r_0), \ldots, \Gamma_n(r_n) \vdash (\underline{e}, K) : \widetilde{A}$ in λ_R.*

Proof. $\Pi, (\rho_0 : B_0, \Gamma_0), \ldots, (\rho_n : B_n, \Gamma_n) \vdash (e, K) : A$ is defined as $\Pi \vdash \kappa_0[\lambda\rho_0\Gamma_0.\kappa_1[\lambda\rho_1\Gamma_1.\ldots\kappa_n[\lambda\rho_n\Gamma_n.e]\ldots]] : B_0 \to \Gamma_0 \to B_1 \to \Gamma_1 \to \ldots \to B_n \to \Gamma_n \to A$ where $K = (\kappa_0^1, \ldots, \kappa_m^1)$, $\kappa_i = [\cdot]$ for $0 \leq i < n - m$, and $\kappa_{n-m+i} = \kappa_i^1$ for $0 \leq i \leq m$.

The claim is proved by induction on e.

Case x. Suppose $\overrightarrow{\Gamma}, \Gamma_n \vdash x : A$, $\Gamma_n(x) = A$, and $\overrightarrow{r}, r_n \vdash x \mapsto (r_n(x'), \bot)$. Let $r_n = \rho + s$. $r_n(x')$ is x or $\rho \cdot x'$. We have $\Gamma_n(r_n) = (\rho : (\widetilde{\Gamma_n})', \widetilde{\Gamma_n}|_s)$. Since $\widetilde{\Gamma_n}(x) = \widetilde{A}$, we have $\rho : (\widetilde{\Gamma_n})', \widetilde{\Gamma_n}|_s \vdash r_n(x') : \widetilde{A}$. Hence we have the claim.

Case $\lambda x.e$. Suppose

$$\frac{\overrightarrow{\Gamma}, \Gamma_n + (x : A) \vdash e : B}{\overrightarrow{\Gamma}, \Gamma_n \vdash \lambda x.e : A \to B} \qquad \frac{\overrightarrow{r}, r_n + \{x' : x\} \vdash e \mapsto (\underline{e}, K)}{\overrightarrow{r}, r_n \vdash \lambda x.e \mapsto (\lambda x.\underline{e}, K)}$$

By induction hypothesis, $\overrightarrow{\Gamma(r)}, (\Gamma_n + (x : A))(r_n + \{x' : x\}) \vdash (\underline{e}, K) : \widetilde{B}$. Hence $\overrightarrow{\Gamma(r)}, \Gamma_n(r_n) \vdash (\lambda x.\underline{e}, K) : \widetilde{A} \to \widetilde{B}$.

Case $e_1 e_2$. Suppose

$$\frac{\overrightarrow{\Gamma} \vdash e_1 : A \to B \quad \overrightarrow{\Gamma} \vdash e_2 : A}{\overrightarrow{\Gamma} \vdash e_1 e_2 : B} \qquad \frac{\overrightarrow{r} \vdash e_1 \mapsto (\underline{e_1}, K_1) \quad \overrightarrow{r} \vdash e_2 \mapsto (\underline{e_2}, K_2)}{\overrightarrow{r} \vdash e_1 e_2 \mapsto (\underline{e_1}\ \underline{e_2}, K_1 \bowtie K_2)}$$

By induction hypothesis, $\overrightarrow{\Gamma(r)} \vdash (\underline{e_1}, K_1) : \widetilde{A} \to \widetilde{B}$ and $\overrightarrow{\Gamma(r)} \vdash (\underline{e_2}, K_2) : \widetilde{A}$. By Lemma 2.1, we have the claim.

Case box e. Suppose

$$\frac{\overrightarrow{\Gamma}, \Gamma_n \vdash e : A}{\overrightarrow{\Gamma} \vdash \mathrm{box}\ e : \Box(\Gamma_n \rhd A)} \qquad \frac{\overrightarrow{r}, \rho \vdash e \mapsto (\underline{e}, (K, \kappa))}{\overrightarrow{r} \vdash \mathrm{box}\ e \mapsto (\kappa[\lambda\rho.\underline{e}], K)}$$

By induction hypothesis, $\overrightarrow{\Gamma(r)}, (\rho : (\widetilde{\Gamma_n})', \phi) \vdash (\underline{e}, (K, \kappa)) : \widetilde{A}$. It is $\overrightarrow{\Gamma(r)} \vdash (\kappa[\lambda\rho.\underline{e}], K) : (\widetilde{\Gamma_n})' \to \widetilde{A}$. Since $\widetilde{\Box(\Gamma_n \rhd A)} = (\widetilde{\Gamma_n})' \to \widetilde{A}$, we have the claim.

Case unbox e. Suppose

$$\frac{\overrightarrow{\Gamma} \vdash e : \Box(\Gamma_n \rhd A)}{\overrightarrow{\Gamma}, \Gamma_n \vdash \mathrm{unbox}\ e : A} \qquad \frac{\overrightarrow{r} \vdash e \mapsto (\underline{e}, K)}{\overrightarrow{r}, r_n \vdash \mathrm{unbox}\ e \mapsto (hr_n, (K, (\lambda h.[\cdot])\underline{e}))}$$

By induction hypothesis, $\overrightarrow{\Gamma(r)} \vdash (\underline{e}, K) : (\widetilde{\Gamma_n})' \to \widetilde{A}$. Let $r_n = \rho + s$. We have $\rho : ((\widetilde{\Gamma_n})' - s), \widetilde{\Gamma_n}|_s \vdash r_n : (\widetilde{\Gamma_n})'$. Hence $h : (\widetilde{\Gamma_n})' \to \widetilde{A}, \overrightarrow{\Gamma(r)}, (\rho : ((\widetilde{\Gamma_n})' - s), \widetilde{\Gamma_n}|_s) \vdash (hr_n, (K, [\cdot])) : \widetilde{A}$. Hence $\overrightarrow{\Gamma(r)} \vdash ((\lambda h\rho(\Gamma_n|_s).hr_n)\underline{e}, K) : ((\widetilde{\Gamma_n})' - s) \to \widetilde{\Gamma_n}|_s \to \widetilde{A}$. It is $\overrightarrow{\Gamma(r)}, (\rho : ((\widetilde{\Gamma_n})' - s), \widetilde{\Gamma_n}|_s) \vdash (hr_n, (K, (\lambda h.[\cdot])\underline{e})) : \widetilde{A}$.

Other cases are similar. $\square$

Corollary 2.3 *If $(x_1 : A_1, \ldots, x_n : A_n) \vdash e : A$ in λ_S and $\rho + \{x_1' : x_1, \ldots, x_n' : x_n\} \vdash e \mapsto (\underline{e}, K)$, then $x_1 : \widetilde{A}_1, \ldots, x_n : \widetilde{A}_n \vdash K(\underline{e}) : \widetilde{A}$ in λ_R.*

Proof. Let Γ be the type context given by $(x_1 : A_1, \ldots, x_n : A_n)$. By Theorem 2.2, we have $\vdash K(\lambda\rho\widetilde{\Gamma}.e) : \{\} \to \widetilde{\Gamma} \to \widetilde{A}$. Hence $\rho : \{\}, \widetilde{\Gamma} \vdash K(\underline{e}) : \widetilde{A}$. Since ρ does not appear in $K(\underline{e})$, 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 λ_S and $\rho + \{x_1' : x_1, \ldots, x_n' : x_n\}, \rho_1, \ldots, \rho_n \vdash e \mapsto (\underline{e}, K)$, then $x_1 : \widetilde{A}_1, \ldots, x_n : \widetilde{A}_n, \rho_1 : (\widetilde{\Gamma_1})', \ldots, \rho_n : (\widetilde{\Gamma_n})' \vdash K(\underline{e}) : \widetilde{A}$ in λ_R.*

Proof. Let $\Pi = (x_1 : A_1, \ldots, x_n : A_n)$. By Theorem 2.2, we have $(\rho : \{\}, \widetilde{\Pi}), \overrightarrow{\Gamma(\rho)} \vdash (\underline{e}, K) : \widetilde{A}$. We have $\Gamma_i(\rho_i) = (\rho_i : (\widetilde{\Gamma_i})', \phi)$. Hence $\vdash \kappa_0[\lambda\rho\widetilde{\Pi}.\kappa_1[\lambda\rho_1.\ldots.\kappa_n[\lambda\rho_n.\underline{e}]\ldots]] : \{\} \to \widetilde{\Pi} \to (\widetilde{\Gamma_1})' \to \ldots \to (\widetilde{\Gamma_n})' \to \widetilde{A}$. Let $\kappa_i = (\lambda h_i : [\cdot])e_i$. By the generation lemma, $h_0 : C_0, \rho : \{\}, \widetilde{\Pi}, h_1 : C_1, \rho_1 : (\widetilde{\Gamma_1})', \ldots, h_{i-1} : C_{i-1}, \rho_{i-1} : (\widetilde{\Gamma_{i-1}})' \vdash e_i : C_i$ for $0 \leq i \leq n$ and $h_0 : C_0, \rho : \{\}, \widetilde{\Pi}, h_1 : C_1, \rho_1 : (\widetilde{\Gamma_1})', \ldots, h_n : C_n, \rho_n : (\widetilde{\Gamma_n})' \vdash e : \widetilde{A}$ for some $C_0, \ldots, C_n$. Hence $\rho : \{\}, \widetilde{\Pi}, \overrightarrow{\rho : (\widetilde{\Gamma})'} \vdash K(\underline{e}) : \widetilde{A}$. Since ρ does not appear in $K(\underline{e})$, we have the claim. $\square$

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.

$\bot \vdash \text{box } ((\lambda x.x)y) \mapsto (\lambda\rho.(\lambda x.x)(\rho \cdot y'), \bot)$,
$\vdash \text{box } ((\lambda x.x)y) : \Box((y : A) \rhd A)$,
$\vdash \lambda\rho.(\lambda x.x)(\rho \cdot y') : \{y' : \widetilde{A}\} \to \widetilde{A}$.

Example 2.

$\vdash 1 : N$,
$\bot \vdash \text{box (unbox } ((\lambda x.x)(\text{box } 1))) \mapsto ((\lambda h\rho_1.h\rho_1)((\lambda x.x)(\lambda\rho_2.1)), \bot)$,
$\vdash \text{box (unbox } ((\lambda x.x)(\text{box } 1))) : \Box N$,
$\vdash (\lambda h\rho_1.h\rho_1)((\lambda x.x)(\lambda\rho_2.1)) : \{\} \to N$.

Example 3.

$\bot \vdash \text{box } (\lambda x.\text{unbox } (\text{box } x)) \mapsto ((\lambda h\rho_1 x.h(\rho_1 + \{x' : x\}))(\lambda\rho_2.\rho_2 \cdot x'), \bot)$,
$\vdash \text{box } (\lambda x.\text{unbox } (\text{box } x)) : \Box(A \to A)$,
$\vdash (\lambda h\rho_1 x.h(\rho_1 + \{x' : x\}))(\lambda\rho_2.\rho_2 \cdot x') : \{\} \to \widetilde{A} \to \widetilde{A}$.

Example 4. Our type translation works well with variable capturing.

$\bot \vdash (\lambda x.\text{box } (\lambda y.(\text{unbox } x)))(\text{box } y) \mapsto ((\lambda x.(\lambda h\rho_1 y.h(\rho_1 + \{y' : y\}))x)(\lambda\rho_2.\rho_2 \cdot y'), \bot)$,
$\vdash (\lambda x.\text{box } (\lambda y.(\text{unbox } x)))(\text{box } y) : \Box(A \to A)$,
$\vdash (\lambda x.(\lambda h\rho_1 y.h(\rho_1 + \{y' : y\}))x)(\lambda\rho_2.\rho_2 \cdot y') : \{\} \to \widetilde{A} \to \widetilde{A}$.

References

[1] B. Aktemur, Type Checking Program Generators Using the Record Calculus, draft, 2009.

[2] I. Kim, K. Yi, and C. Calcagno, A Polymorphic Modal Type System for Lisp-Like Multi-Staged Languages In: *Proceeding of POPL 06* (2006) 257–269.

[3] Kwangkuen Yi, private communication, 2010, March.