1 Typed λ_{lvar} calculus

Given a set D, let $\mathbb{D}$ be a 4-tuple $(D, \sqcup_D, \bot_D, \top_D)$, and let there be a function $incomp(d) := \forall d' \in D.(d' \neq d \Rightarrow d \sqcup d' = \top_D)$ that models $\sqcup$-incompatibility among elements of $\mathbb{J}$, i.e $\mathbb{J} = \{d \in D | incomp(d)\}$.

1.1 Syntax

Types and environments

$$
\begin{array}{rcll}
T, U & := & \mathbf{1} & \text{unit} \\
 & | & T \times U & \text{product} \\
 & | & T \to U & \lambda \text{ abstraction} \\
 & | & \mathcal{J} & \text{threshold set, where } \mathcal{J} \subseteq \mathbb{J} \\
 & | & \mathcal{D}^d & \text{values } d \text{ in } D \text{ indexed by } \bigsqcup d \\
 & | & \mathcal{L}^d_{\mathcal{D}} & \text{locations (indexed by } d\text{) of values in } \mathcal{D}^d
\end{array}
$$

$$
\begin{array}{rcll}
\Gamma & := & \cdot & \text{empty environment} \\
 & | & x : T & \text{environment extension}
\end{array}
$$

Terms and Status bits

$$
\begin{array}{rcll}
L, M, N & := & x & \text{variables} \\
 & | & V & \text{values} \\
 & | & B & \text{status bits} \\
 & | & K & \text{constants} \\
 & | & M\ N & \text{parallel application} \\
 & | & () & \text{unit introduction} \\
 & | & \mathbf{let}\ () = M\ \mathbf{in}\ N & \text{unit elimination} \\
 & | & (M, N) & \text{product introduction} \\
 & | & \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N & \text{product elimination}
\end{array}
$$

$$
\begin{array}{rcll}
B & := & 1 & \text{frozen LVar} \\
 & | & 0 & \text{unfrozen LVar}
\end{array}
$$

Stores and Configurations

$$
\begin{array}{rcll}
S & := & \cdot & \text{empty store} \\
 & | & S, l \mapsto (B, V) & \text{store extension} \\
 & | & \top & \text{top}
\end{array}
$$

$$
\begin{array}{rcll}
C & := & \langle M \mid S \rangle & \text{programs are pairs of terms and store} \\
 & | & \mathbf{error} & \text{runtime crashes}
\end{array}
$$

Values

$$
\begin{array}{rcll}
V, W & := & l & \text{locations} \\
& | & J & \text{threshold set} \\
& | & (B, d) & \text{states, where } D \text{ is the distinguished set, and } d \in D \\
& | & \lambda x.M & \lambda \text{ abstraction} \\
& | & () & \text{unit} \\
& | & (M, N) & \text{product}
\end{array}
$$

Constants

$$
\begin{array}{rcll}
K & := & \mathbf{new} & \text{allocate new LVar} \\
& | & \mathbf{freeze} & \text{freeze LVar} \\
& | & \mathbf{get} & \text{read threshold from LVar} \\
& | & \mathbf{put} & \text{add value to LVar}
\end{array}
$$

Evaluation Context

$$
\begin{array}{rcl}
E & := & \square \\
& | & V\ E \\
& | & E\ V \\
& | & (V, E) \\
& | & (E, V) \\
& | & \mathbf{let}\ () = E\ \mathbf{in}\ M \\
& | & \mathbf{let}\ (x, y) = E\ \mathbf{in}\ M
\end{array}
$$

1.2 Type System

$$
\textsc{T-Var}\ \frac{}{x : T \vdash x : T}
\qquad
\textsc{T-Lam}\ \frac{\Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x.M : T \to U}
\qquad
\textsc{T-App}\ \frac{\Gamma \vdash M : T \to U \qquad \Gamma \vdash N : T}{\Gamma \vdash M\ N : U}
$$

$$
\textsc{T-Unit}\ \frac{}{\cdot \vdash () : \mathbf{1}}
\qquad
\textsc{T-LetUnit}\ \frac{\Gamma \vdash M : \mathbf{1} \qquad \Gamma \vdash N : T}{\Gamma \vdash \mathbf{let}\ () = M\ \mathbf{in}\ N : T}
$$

$$
\textsc{T-Pair}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : U}{\Gamma \vdash (M, N) : T \times U}
\qquad
\textsc{T-LetPair}\ \frac{\Gamma \vdash M : T \times T' \qquad \Gamma, x : T, y : T' \vdash N : U}{\Gamma \vdash \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N : U}
$$

$$
\textsc{T-New}\ \frac{}{\Gamma, l : \mathcal{L}_{\mathcal{D}}^{\perp} \vdash \mathbf{new} : \mathcal{L}_{\mathcal{D}}^{\perp}}
\qquad
\textsc{T-Freeze}\ \frac{\Gamma \vdash l : \mathcal{L}_{\mathcal{D}}^{d}}{\Gamma \vdash \mathbf{freeze}\ l : \mathcal{D}^{d}}
$$

$$
\textsc{T-Get}\ \frac{\Gamma \vdash l : \mathcal{L}_{\mathcal{D}}^{d} \qquad \Gamma \vdash J : \mathcal{J} \qquad \exists d' \in \mathcal{J}.d \sqsubseteq d'}{\Gamma \vdash \mathbf{get}\ l\ J : \mathcal{D}^{d}}
\qquad
\textsc{T-Put}\ \frac{\Gamma \vdash l : \mathcal{L}_{\mathcal{D}}^{d} \qquad \Gamma \vdash M : \mathcal{D}^{d'}}{\Gamma, l : \mathcal{L}_{\mathcal{D}}^{d \sqcup d'} \vdash \mathbf{put}\ l\ M : 1}
$$

1.3 Operational semantics

$$
\begin{array}{llcl}
\textsc{E-Lam} & \langle(\lambda x.M)\ V \mid S\rangle & \longrightarrow & \langle M\{V/x\} \mid S\rangle \\
\textsc{E-Unit} & \langle \mathbf{let}\ () = ()\ \mathbf{in}\ M \mid S\rangle & \longrightarrow & \langle M \mid S\rangle \\
\textsc{E-Pair} & \langle \mathbf{let}\ (x,y) = (V,W)\ \mathbf{in}\ M \mid S\rangle & \longrightarrow & \langle M\{V/x\}\{W/y\} \mid S\rangle \\
\textsc{E-New} & \langle \mathbf{new} \mid S\rangle & \longrightarrow & \langle l \mid S, l \mapsto (0,\bot)\rangle \\
\textsc{E-Freeze} & \langle \mathbf{freeze} \mid S, l \mapsto (b,d)\rangle & \longrightarrow & \langle d \mid S, l \mapsto (1,d)\rangle
\end{array}
$$

$$
\textsc{E-Put}\quad \frac{s = (b,d) \qquad s' = (b,d') \qquad s \sqcup s' \neq \top}{\langle \mathbf{put}\ l\ d' \mid S, l \mapsto s\rangle \longrightarrow \langle s' \mid S, l \mapsto s'\rangle}
$$

$$
\textsc{E-Put-Err}\quad \frac{s = (b,d) \qquad s' = (b,d') \qquad s \sqcup s' = \top}{\langle \mathbf{put}\ l\ d' \mid S, l \mapsto s\rangle \longrightarrow \mathbf{error}}
$$

$$
\textsc{E-Get}\quad \frac{J \in \mathcal{J} \qquad d' \in J \qquad d' \sqsubseteq d}{\langle \mathbf{get}\ l\ J \mid S, l \mapsto (b,d)\rangle \longrightarrow \langle d' \mid S, l \mapsto (b,d)\rangle}
$$

$$
\textsc{E-Pair'}\quad \frac{\langle M \mid S\rangle \longrightarrow \langle M' \mid S'\rangle \qquad \langle N \mid S\rangle \longrightarrow \langle N' \mid S''\rangle}{\langle (M,N) \mid S\rangle \longrightarrow \langle (M',N') \mid S' \sqcup S''\rangle}
$$

$$
\textsc{E-App}\quad \frac{\langle M \mid S\rangle \longrightarrow \langle M' \mid S'\rangle \qquad \langle N \mid S\rangle \longrightarrow \langle N' \mid S''\rangle}{\langle M\ N \mid S\rangle \longrightarrow \langle M'\ N' \mid S' \sqcup S''\rangle}
$$

$$
\textsc{E-Lift}\quad \frac{M \longrightarrow M'}{\langle E[M] \mid S\rangle \longrightarrow_E \langle E[M'] \mid S\rangle}
$$

1.4 Syntax sugar

$$
\textsc{T-RunLVar}\quad \frac{\Gamma \vdash M : \mathcal{L}^d_{\mathcal{D}} \to ()}{\Gamma \vdash \mathbf{runLVar}\ M : \mathcal{D}^d}
$$

$$
\textsc{E-RunLVar}\quad \mathbf{runLVar}\ M \quad \longrightarrow \quad (\lambda l.\mathbf{let}\ () = M\ l\ \mathbf{in\ freeze}\ l)\ \mathbf{new}
$$

2 Metatheory of Typed $\lambda_{\mathbf{lvar}}$ calculus

2.1 Translation to $\lambda_{\mathbf{LVar}}$ from typed $\lambda_{\mathbf{lvar}}$

Definition 1. A translation is a function $\zeta : C \to \sigma$, such that:

Add partial-order rules for state s, where $s = (b,d)$.

- it should maintain the same number of steps in C when translated into σ;
- it should not introduce sequentialisation.

$$
\begin{array}{lcll}
\zeta(\mathbf{error}) & = & \mathbf{error} & \\
\zeta(\langle \mathbf{get}\ l\ J \mid S\rangle) & = & \langle S\ ;\ \mathbf{get}\ l\ P\rangle & \text{where } p_1 \cong s \text{ and } P \cong J \\
\zeta(\langle \mathbf{put}\ l\ d' \mid S\rangle) & = & \langle S\ ;\ \mathbf{put}_i\ l\rangle & \text{where } u_{p_i} \coloneqq \lambda d_i.d \sqcup d_i \\
\zeta(\langle \mathbf{new} \mid S\rangle) & = & \langle S\ ;\ \mathbf{new}\rangle & \\
\zeta(\langle \mathbf{freeze}\ l \mid S\rangle) & = & \langle S\ ;\ \mathbf{freeze}\ l\rangle & \\
\zeta(\langle (\lambda x.M)\ V \mid S\rangle) & = & \langle S\ ;\ (\lambda x.e)\ v\rangle & \\
\zeta(\langle M\ N \mid S\rangle) & = & \langle S\ ;\ e\ e'\rangle & \\
\zeta(\langle () \mid S\rangle) & = & \langle S\ ;\ ()\rangle & \\
\zeta(\langle \mathbf{let}\ () = M\ \mathbf{in}\ N \mid S\rangle) & = & \langle S\ ;\ (\lambda().e)\ e'\rangle & \\
\zeta(\langle (M, N) \mid S\rangle) & = & \langle S\ ;\ (\lambda x.\lambda y.\lambda f.fxy)\ e\ e'\rangle & \\
\zeta(\langle \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N \mid S\rangle) & = & \langle S\ ;\ e\ (\lambda x.\lambda y.e')\rangle & \\
\zeta(\langle M \mid S, l \mapsto (0, d)\rangle) & = & \langle S[l \mapsto (d, \mathtt{false})]\ ;\ e\rangle & \\
\zeta(\langle M \mid S, l \mapsto (1, d)\rangle) & = & \langle S[l \mapsto (d, \mathtt{true})]\ ;\ e\rangle &
\end{array}
$$

Lemma 1 (Translation, Typed $\lambda_{\text{lvar}} \rightsquigarrow \lambda_{\text{LVar}}$)**.**
For any translation ζ,

- *if $C \longrightarrow C'$ and $\sigma \hookrightarrow \sigma'$ and $\zeta(C) = \sigma$, then $\zeta(C') = \sigma'$;*
- *if $C \longrightarrow_E C'$ and $\sigma \mapsto \sigma'$ and $\zeta(C) = \sigma$, then $\zeta(C') = \sigma'$.*

Proof. By induction on the structure of C.

Case. $C = \langle \mathbf{error} \mid S\rangle$, $\sigma = \langle S\ ;\ \mathbf{error}\rangle$.
C and σ cannot step. Hence, the translation is vacuously valid.

Case. $C = \langle \mathbf{get}\ l\ J \mid S\rangle$, $\sigma = \langle S\ ;\ \mathbf{get}\ l\ P\rangle$.
Given the operational semantics, C steps to $C' = \langle s' \mid S, l \mapsto (b, d)\rangle$. And given λ_{LVar}'s operational semantics, σ steps to $\sigma' = \langle S\ ;\ p_2\rangle$. Applying $\zeta(C')$, we get $\langle S\ ;\ p_2\rangle$. Hence, the translation is valid.

Case. $C = \langle \mathbf{put}\ l\ d' \mid S\rangle$, $\sigma = \langle S\ ;\ \mathbf{put}_i\ l\rangle$.
Given the operational semantics, C can either error or take a step.

Sub-case. $C' = \langle s' \mid S, l \mapsto s'\rangle$
Given λ_{LVar}'s operational semantics, σ steps to $\sigma' = \langle S\ ;\ p_2\rangle$ if $d \sqcup d_i \neq \top$, which is exactly the same as applying ζ to C'.

Sub-case. $C' = \mathbf{error}$
Given λ_{LVar}'s operational semantics, σ steps to $\sigma' = \mathbf{error}$ if $d \sqcup d_i = \top$, which

is exactly the same as applying ζ to C'.

Hence, the translation is valid.

Case. $C = \langle \mathbf{new} \mid S \rangle$, $\sigma = \langle S\ ;\ \mathbf{new} \rangle$

Case. $C = \langle \mathbf{freeze} \mid S \rangle$, $\sigma = \langle S\ ;\ \mathbf{freeze} \rangle$

Case. $C = \langle (M, N) \mid S \rangle$, $\sigma = \langle S\ ;\ (\lambda x.\lambda y.\lambda f.fxy)\ e\ e' \rangle$

Case. $C = \langle \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N \mid S \rangle$, $\sigma = \langle S\ ;\ e'\ (\lambda x.\lambda y.e) \rangle$

□

2.2 Determinism

Determinism proof as stated in Kuper'15.

Definition 2. Permutation

Definition 3. Permutation of an expression

Definition 4. Permutation of a store

Definition 5. Permutation of configurations

Lemma 2.
Permutability

Lemma 3.
Internal Determinism

Lemma 4.
Strong Confluence

Lemma 5.
Confluence

Theorem 1.
Determinism

2.3 Type safety

Theorem 2.
Progress

Theorem 3.
Preservation

Corollary 1.
Type Safety