

Homework 6

15-814: Type Systems for Programming Languages

TA: Kumar Avijit (kavijit@cs.cmu.edu)

Out: Oct. 25, 2009

Due: Oct. 31, 2009 (11:59 PM)

1 Reynolds' IA with references

Please refer to the appendix for the full static and dynamic semantics of the language.

In this question, you will have to prove type safety for Reynolds' IA extended with reference types.

Question 1. *Prove progress theorem for IA:*

Theorem 1. 1. If $\cdot \vdash_{\Sigma} e : \tau$, then either $e \text{ val}_{\Sigma}$, or $e \xrightarrow{\Sigma} e'$ for some e' .

2. If $m @ \mu \sim_{\Sigma} \tau$, then either $m = \text{ret}(e)$, where $e \text{ val}_{\Sigma}$ and $\cdot \vdash_{\Sigma} e : \tau$, or $m @ \mu \xrightarrow{\Sigma} m' @ \mu'$ for some m', μ' .

The state $m @ \mu$ is well-formed, i.e. $m @ \mu \sim_{\Sigma} \tau$ iff $\cdot \vdash_{\Sigma} m \sim \tau$ and $\mu : \Sigma$, and $\Sigma \text{ ok}$. The memory typing judgment $\mu : \Sigma$ is defined as follows:

$$\frac{\cdot \vdash \cdot}{\cdot \vdash \cdot} \quad \frac{\mu : \Sigma \quad \cdot \vdash e : \tau}{\mu \otimes \langle a := e \rangle : \Sigma, a : \tau}$$

The judgment $\Sigma \text{ ok}$ is defined as

$$\frac{\cdot \vdash \cdot}{\cdot \text{ ok}} \quad \frac{\Sigma \text{ ok} \quad \tau \text{ mobile} \quad a \notin \text{dom}(\Sigma)}{\Sigma, a : \tau \text{ ok}}$$

Note that since we require all declarations to be of mobile type, the expression e should be typable under the empty memory.

Question 2. *Prove preservation for IA:*

Theorem 2 (Preservation). 1. If $e \xrightarrow{\Sigma} e'$ and $\cdot \vdash_{\Sigma} e : \tau$ then $\cdot \vdash_{\Sigma} e' : \tau$.

2. If $m @ \mu \xrightarrow{\Sigma} m' @ \mu'$ and $m @ \mu \sim_{\Sigma} \tau$ then $m' @ \mu' \sim_{\Sigma} \tau$.

In order to prove preservation, you will first have to prove the following Lemma regarding universal typability of values of mobile type:

Lemma 1 (Mobility). *If $e \text{ val}_\Sigma$ and $\Gamma \vdash_\Sigma e : \tau$ and $\tau \text{ mobile}$, then $\Gamma \vdash. e : \tau$.*

Another lemma that you need is substitution lemma for substituting expressions in expressions as well as commands. You can assume this without proof:

Lemma 2 (Substitution). *[---do not prove---]*

1. *If $\Gamma \vdash_\Sigma e_1 : \tau_1$ and $\Gamma, x:\tau_1 \vdash_\Sigma e_2 : \tau_2$ then $\Gamma \vdash_\Sigma [e_1/x]e_2 : \tau_2$*
2. *If $\Gamma \vdash_\Sigma e : \tau_1$ and $\Gamma, x:\tau_1 \vdash_\Sigma m \sim \tau_2$ then $\Gamma \vdash_\Sigma [e/x]m \sim \tau_2$*

Question 3. *Explain by a counterexample that violates safety, why the restriction $\tau' \text{ mobile}$ is necessary in Rule (dcl) for typing declarations.*

1.1 Monad axioms

We will now study some properties of the lax modality $\tau \text{ cmd}$. The first property says that every function of type $\tau_1 \rightarrow \tau_2$ can be mapped to a function of type $\tau_1 \text{ cmd} \rightarrow \tau_2 \text{ cmd}$ which essentially is the same function, but lifted to commands. This proposition is expressed by the type

$$(\tau_1 \rightarrow \tau_2) \rightarrow (\tau_1 \text{ cmd} \rightarrow \tau_2 \text{ cmd}) \quad (1)$$

Question 4. *Give an expression, say $\text{map}_{\tau_1, \tau_2}$, in Reynolds' IA, s.t. $\cdot \vdash. \text{map}_{\tau_1, \tau_2} : (\tau_1 \rightarrow \tau_2) \rightarrow (\tau_1 \text{ cmd} \rightarrow \tau_2 \text{ cmd})$.*

Question 5. *In fact, a stronger property, commonly called **K** by modal logicians, also holds:*

$$(\tau_1 \rightarrow \tau_2) \text{ cmd} \rightarrow (\tau_1 \text{ cmd} \rightarrow \tau_2 \text{ cmd}) \quad (2)$$

Give a closed expression $\mathbf{K}_{\tau_1, \tau_2}$ in IA that witnesses the above type.

The second property is expressed by the following proposition:

$$\tau \text{ cmd cmd} \rightarrow \tau \text{ cmd} \quad (3)$$

In terms of IA, this states that a command which returns another command of type τ can be “flattened” to a single command of type τ (instead of type $\tau \text{ cmd}$). This is always possible because we can sequentialize the execution of the subsequent commands, to get a single command.

Question 6. *Give an expression, say join_τ , s.t. $\cdot \vdash. \text{join}_\tau : \tau \text{ cmd cmd} \rightarrow \tau \text{ cmd}$.*

1.2 Destination-passing with IA

With a runtime stack at our disposal in IA, we can transform certain functions to commands so that instead of returning a value, the functions instead put that value in a memory location on the stack. For this we will assume product types $\tau_1 \times \tau_2$ and **1** are added to our language. A function f of type $\tau_1 \rightarrow \tau_2$,

where the return type τ_2 is mobile can be transformed into a function of type $\tau_1 \times \tau_2 \text{ ref} \rightarrow \mathbf{1} \text{ cmd}$ as:

$$\lambda x:\tau_1 \times \tau_2 \text{ ref. cmd}(\text{do}\{- \leftarrow \text{setv}(\text{snd}(x)); \text{ap}(f; \text{fst}(x)); \text{ret}(\langle \rangle)\})$$

(Please refer to Chapter 37 of PFPL for a definition of command-sequencing operation $\text{do}\{x \leftarrow m_1; m_2\}$.)

We can also write recursive functions using the above style. The idea is the same — before making a recursive call, allocate a stack location for the callee to put the return value in, and pass a reference to the location to the callee. After the recursive call finishes, get the result from the location, and proceed.

Question 7. For this question, we will add `fix` from Plotkin’s PCF to our language (refer to Chapter 15 of PFPL for PCF). Write the factorial function in the destination-passing style:

$$\text{factorial} = \text{fix } f : \text{nat} \times \text{nat ref} \rightarrow \mathbf{1} \text{ cmd is } \dots$$

A IA: Statics and dynamics

Types	$\tau ::= \text{nat} \mid \tau_1 \rightarrow \tau_2 \mid \tau \text{ cmd} \mid \tau \text{ ref}$
Expressions	$e ::= x \mid \mathbf{z} \mid \mathbf{s}(e) \mid \text{ifz}(e; e_0; x.e_1) \mid \lambda x:\tau.e \mid \text{ap}(e_1; e_2) \mid \text{cmd}(m) \mid \text{ref}[a]$
Commands	$m ::= \text{ret}(e) \mid \text{dcl}_\tau(e; a.m) \mid \text{seq}(e; x.m)$ $\mid \text{get}[a] \mid \text{set}[a](e) \mid \text{getv}(e) \mid \text{setv}(e_1; e_2)$
Memory locations	a
Memory context	$\Sigma ::= \cdot \mid \Sigma, a:\tau$
Memory	$\mu ::= \cdot \mid \mu \otimes \langle a := e \rangle$

By abuse of the notation, we shall use $\mu \otimes \mu'$ to denote the successive concatenation of μ with elements of μ' .

A.1 Statics

$\boxed{\Gamma \vdash_{\Sigma} e : \tau}$

$$\begin{array}{c}
\frac{}{\Gamma, x:\tau \vdash_{\Sigma} x : \tau}(\text{hyp}) \qquad \frac{\Gamma, x:\tau \vdash_{\Sigma} e : \tau'}{\Gamma \vdash_{\Sigma} \lambda x:\tau. e : \tau \rightarrow \tau'}(\rightarrow\text{-I}) \\
\\
\frac{\Gamma \vdash_{\Sigma} e_1 : \tau_2 \rightarrow \tau_1 \quad \Gamma \vdash_{\Sigma} e_2 : \tau_2}{\Gamma \vdash_{\Sigma} \text{ap}(e_1; e_2) : \tau_1}(\rightarrow\text{-E}) \qquad \frac{}{\Gamma \vdash_{\Sigma} \mathbf{z} : \text{nat}}(\text{nat-I}_1) \\
\\
\frac{\Gamma \vdash_{\Sigma} e : \text{nat}}{\Gamma \vdash_{\Sigma} \mathbf{s}(e) : \text{nat}}(\text{nat-I}_2) \\
\\
\frac{\Gamma \vdash_{\Sigma} e : \text{nat} \quad \Gamma \vdash_{\Sigma} e_0 : \tau \quad \Gamma, x:\text{nat} \vdash_{\Sigma} e_1 : \tau}{\Gamma \vdash_{\Sigma} \text{ifz}(e; e_0; x.e_1) : \tau}(\text{nat-E}) \\
\\
\frac{\Gamma \vdash_{\Sigma} m \sim \tau}{\Gamma \vdash_{\Sigma} \text{cmd}(m) : \tau \text{ cmd}}(\text{cmd-I}) \qquad \frac{}{\Gamma \vdash_{\Sigma, a:\tau} \text{ref}[a] : \tau \text{ ref}}(\text{ref-I})
\end{array}$$

$\boxed{\Gamma \vdash_{\Sigma} m \sim \tau}$

$$\begin{array}{c}
\frac{\Gamma \vdash_{\Sigma} e : \tau}{\Gamma \vdash_{\Sigma} \text{ret}(e) \sim \tau}(\text{ret}) \\
\\
\frac{\Gamma \vdash_{\Sigma} e : \tau' \quad \tau' \text{ mobile} \quad \tau \text{ mobile} \quad \Gamma \vdash_{\Sigma, a:\tau'} m \sim \tau \quad a \notin \text{dom}(\Sigma)}{\Gamma \vdash_{\Sigma} \text{dcl}_{\tau'}(e; a.m) \sim \tau}(\text{dcl}) \\
\\
\frac{\Gamma \vdash_{\Sigma} e : \tau' \text{ cmd} \quad \Gamma, x:\tau' \vdash_{\Sigma} m \sim \tau}{\Gamma \vdash_{\Sigma} \text{seq}(e; x.m) \sim \tau}(\text{seq}) \qquad \frac{}{\Gamma \vdash_{\Sigma, a:\tau} \text{get}[a] \sim \tau}(\text{get}) \\
\\
\frac{\Gamma \vdash_{\Sigma, a:\tau} e : \tau}{\Gamma \vdash_{\Sigma, a:\tau} \text{set}[a](e) \sim \tau}(\text{set}) \qquad \frac{\Gamma \vdash_{\Sigma} e : \tau \text{ ref}}{\Gamma \vdash_{\Sigma} \text{getv}(e) \sim \tau}(\text{getv}) \\
\\
\frac{\Gamma \vdash_{\Sigma} e_1 : \tau \text{ ref} \quad \Gamma \vdash_{\Sigma} e_2 : \tau}{\Gamma \vdash_{\Sigma} \text{setv}(e_1; e_2) \sim \tau}(\text{setv})
\end{array}$$

$\boxed{\tau \text{ mobile}}$

$\frac{}{\text{nat mobile}}$

A.2 Dynamics

$e \text{ val}_\Sigma$

$$\frac{}{z \text{ val}_\Sigma} (\text{val-z}) \quad \frac{e \text{ val}_\Sigma}{s(e) \text{ val}_\Sigma} (\text{val-s}) \quad \frac{}{\lambda x:\tau.e \text{ val}_\Sigma}$$

$$\frac{}{\text{cmd}(m) \text{ val}_\Sigma} (\text{val-cmd}) \quad \frac{}{\text{ref}[a] \text{ val}_{\Sigma,a;\tau}} (\text{val-ref})$$

$e \xrightarrow{\Sigma} e'$

$$\frac{e \xrightarrow{\Sigma} e'}{s(e) \xrightarrow{\Sigma} s(e')} (\text{s} \rightarrow) \quad \frac{e \xrightarrow{\Sigma} e'}{\text{ifz}(e; e_0; x.e_1) \xrightarrow{\Sigma} \text{ifz}(e'; e_0; x.e_1)} (\text{ifz} \rightarrow)$$

$$\frac{}{\text{ifz}(z; e_0; x.e_1) \xrightarrow{\Sigma} e_0} (\text{ifz}_z) \quad \frac{}{\text{ifz}(s(e); e_0; x.e_1) \xrightarrow{\Sigma} [e/x]e_1} (\text{ifz}_s)$$

$$\frac{e_1 \xrightarrow{\Sigma} e'_1}{\text{ap}(e_1; e_2) \xrightarrow{\Sigma} \text{ap}(e'_1; e_2)} (\text{ap} \rightarrow_1) \quad \frac{e_2 \xrightarrow{\Sigma} e'_2}{\text{ap}(\lambda x:\tau.e_1; e_2) \xrightarrow{\Sigma} \text{ap}(\lambda x:\tau.e_1; e'_2)} (\text{ap} \rightarrow_2)$$

$$\frac{e_2 \text{ val}_\Sigma}{\text{ap}(\lambda x:\tau.e_1; e_2) \xrightarrow{\Sigma} [e_2/x]e_1} (\text{ap} \rightarrow_3)$$

$$\boxed{m@μ \xrightarrow{\Sigma} m'@μ'}$$

$$\frac{e \xrightarrow{\Sigma} e'}{\text{ret}(e)@μ \xrightarrow{\Sigma} \text{ret}(e')@μ} (\text{ret } \rightarrow)$$

$$\frac{e \xrightarrow{\Sigma} e'}{\text{dcl}_\tau(e; a.m)@μ \xrightarrow{\Sigma} \text{dcl}_\tau(e'; a.m)@μ} (\text{dcl } \rightarrow_1)$$

$$\frac{e \text{ val}_\Sigma \quad m@μ \otimes \langle a := e \rangle \xrightarrow{\Sigma, a:\tau} m'@μ' \otimes \langle a := e' \rangle}{\text{dcl}_\tau(e; a.m)@μ \xrightarrow{\Sigma} \text{dcl}_\tau(e'; a.m')@μ'} (\text{dcl } \rightarrow_2)$$

$$\frac{e' \text{ val}_\Sigma \quad e \text{ val}_\Sigma}{\text{dcl}_\tau(e'; a.\text{ret}(e))@μ \xrightarrow{\Sigma} \text{ret}(e)@μ} (\text{dcl } \rightarrow_3)$$

$$\frac{e \xrightarrow{\Sigma} e'}{\text{seq}(e; x.m)@μ \xrightarrow{\Sigma} \text{seq}(e'; x.m)@μ} (\text{seq } \rightarrow_1)$$

$$\frac{m'@μ \xrightarrow{\Sigma} m''@μ'}{\text{seq}(\text{cmd}(m'); x.m)@μ \xrightarrow{\Sigma} \text{seq}(\text{cmd}(m''); x.m)@μ'} (\text{seq } \rightarrow_2)$$

$$\frac{e \text{ val}_\Sigma}{\text{seq}(\text{cmd}(\text{ret}(e)); x.m)@μ \xrightarrow{\Sigma} [e/x]m@μ} (\text{seq } \rightarrow_3)$$

$$\frac{}{\text{set}[a]@μ \otimes \langle a := e \rangle \otimes μ' \xrightarrow{\Sigma, a:\tau, \Sigma'} \text{ret}(e)@μ \otimes \langle a := e \rangle \otimes μ'} (\text{get } \rightarrow)$$

$$\frac{e \xrightarrow{\Sigma} e'}{\text{set}[a](e)@μ \xrightarrow{\Sigma} \text{set}[a](e')@μ} (\text{set } \rightarrow_1)$$

$$\frac{e \text{ val}_\Sigma \quad \Sigma = \Sigma_1, a:\tau, \Sigma_2}{\text{set}[a](e)@μ_1 \otimes \langle a := _ \rangle \otimes μ_2 \xrightarrow{\Sigma} \text{ret}(e)@μ_1 \otimes \langle a := e \rangle \otimes μ_2} (\text{set } \rightarrow_2)$$

$$\frac{e \xrightarrow{\Sigma} e'}{\text{getv}(e)@μ \xrightarrow{\Sigma} \text{getv}(e')@μ} (\text{getv } \rightarrow_1)$$

$$\frac{}{\text{getv}(\text{ref}[a])@μ \xrightarrow{\Sigma} \text{get}[a]@μ} (\text{getv } \rightarrow_2)$$

$$\frac{e_1 \xrightarrow{\Sigma} e'_1}{\text{setv}(e_1; e_2)@μ \xrightarrow{\Sigma} \text{setv}(e'_1; e_2)@μ} (\text{setv } \rightarrow_1)$$

$$\frac{6}{\text{setv}(\text{ref}[a]; e)@μ \xrightarrow{\Sigma} \text{set}[a](e)@μ} (\text{setv } \rightarrow_2)$$