Assignment #4

Diego Oniarti - 257835

1 Progress

Write the proof for the progress theorem for the cases related to locations, allocations, dereferencing, and update.

> **Progress:** if $H : \Sigma$ and $\Sigma, \emptyset, \emptyset \vdash t : \tau$ then either $\vdash t.val$ or $\exists t', H'.H \triangleright t \rightsquigarrow H' \triangleright t'$
> Proven by induction

1. locations

$$\begin{array}{c}
\text{w.h. } H : \Sigma \text{ and } \Sigma, \emptyset, \emptyset \vdash l : ref(\tau) \\
\text{t.s.} \begin{array}{ll} \text{either} & \vdash l.val \\ \text{or} & \exists t', H'.H \triangleright l \rightsquigarrow H' \triangleright t' \end{array} \\
\text{by def. } l.val\square
\end{array}$$

2. allocation $t = \text{ new } t_1$

$$\begin{array}{rlr}
 & \text{w.h. } H : \Sigma \text{ and } \Sigma, \emptyset, \emptyset \vdash \text{ new } t_1 : ref(\tau) & \\
\text{by IH w.h. either} & t_1.val & \\
\text{or} & \exists t'_1, H'.\ H \triangleright t_1 \rightsquigarrow H' \triangleright t'_1 & \\
\hline
\text{case } t_1.val: & & \\
\text{by def. of } new: & H \triangleright \text{ new } t_1 \rightsquigarrow H, l \mapsto t_1 \triangleright l\square & \\
\hline
\text{case } \exists t'_1, H'.\ H \triangleright t_1 \rightsquigarrow H' \triangleright t'_1: & & \\
\text{by inversion} & t_1 \equiv \mathbb{E}[t_0] & (HE1) \\
 & t'_1 \equiv \mathbb{E}[t'_0] & (HE2) \\
 & H \triangleright t_0 \rightsquigarrow^p H' \triangleright t'_0 & (HPR) \\
\text{by HE1, HE2 t.s.} & H \triangleright \text{ new } \mathbb{E}[t_0] \rightsquigarrow H' \triangleright \text{ new } \mathbb{E}[t'_0] & \\
\text{by ctx. with HPR and } & \mathbb{E}' = [\text{ new } \mathbb{E}]\square &
\end{array}$$

3. dereferencing $t =!t_1$

$$\begin{array}{rlr}
\text{by HP w.h.} & H : \Sigma \text{ and } \Sigma, \emptyset, \emptyset \vdash t : \tau & (HP1) \\
\text{by typing of ! w.h.} & H : \Sigma \text{ and } \Sigma, \emptyset, \emptyset \vdash t_1 : ref(\tau) & (HP2) \\
\text{t.s.} & \begin{array}{ll} \text{either} & t.val \\ \text{or} & \exists t', H'.\ H \triangleright t \rightsquigarrow H' \triangleright t' \end{array} & \\
\text{by IH w.h.} & \begin{array}{ll} \text{either} & t_1.val \\ \text{or} & \exists t'_1, H'.\ H \triangleright t_1 \rightsquigarrow H' \triangleright t'_1 \end{array} & \\
\hline
\text{case } & t_1.val & \\
\text{by canonicity w.h.} & t_1 \equiv l & \\
\text{by HP2:} & H(l) = v & \\
\text{by def. of !:} & H \triangleright !t_1 \rightsquigarrow H \triangleright v\ \square & \\
\hline
\text{case } & \exists t'_1, H'.\ H \triangleright t_1 \rightsquigarrow H' \triangleright t'_1 & \\
\text{by inversion} & t_1 \equiv \mathbb{E}[t_0] & (HE1) \\
 & t'_1 \equiv \mathbb{E}[t'_0] & (HE2) \\
 & H \triangleright t_0 \rightsquigarrow^p H' \triangleright t'_0 & (HPR) \\
\text{by HE1, HE2 t.s.} & H \triangleright !\mathbb{E}[t_0] \rightsquigarrow H' \triangleright !\mathbb{E}[t'_0] & \\
\text{by ctx with HPR and } \mathbb{E}' =!\mathbb{E}: & H \triangleright \mathbb{E}'[t_0] \rightsquigarrow H' \triangleright E'[t'_0]\square &
\end{array}$$

4. update $t = t_1 := t_2$

by HP w.h. $H : \Sigma$ and $\Sigma, \emptyset, \emptyset \vdash t : \mathbb{N}$ $(HP1)$

by typing of $:=$ w.h. $H : \Sigma$ and $\Sigma, \emptyset, \emptyset \vdash t_1 : ref(\tau)$ $(HP2)$

and $H : \Sigma$ and $\Sigma, \emptyset, \emptyset \vdash t_2 : \tau$ $(HP3)$

by IH w.h. either $t_1.val$ $(A1)$
or $\exists H', t'_1.\ H \rhd t_1 \leadsto H' \rhd t'_1$ $(A2)$

by IH w.h. either $t_2.val$ $(B1)$
or $\exists H', t'_2.\ H \rhd t_2 \leadsto H' \rhd t'_2$ $(B2)$

---

case $A1\ B1$

by canonicity and A1: $t_1 \equiv l$

by definition of $:=$ w.h. $H \rhd t \leadsto H[{}^{l \mapsto t_2}/_{l \mapsto \_}] \rhd 0$ □

---

case $A2$

by inversion $t_1 \equiv \mathbb{E}[t_0]$ $(HE1)$

$t'_1 \equiv \mathbb{E}[t'_0]$ $(HE2)$

$H \rhd t_0 \leadsto^p H' \rhd t'_0$ (HPR)

by HE1,HE2 t.s. $H \rhd \mathbb{E}[t_0] := t_2 \leadsto H' \rhd \mathbb{E}[t'_0] := t_2$

by HPR with $\mathbb{E}' = \mathbb{E} := t_2$: $H \rhd \mathbb{E}'[t_0] \leadsto H' \rhd \mathbb{E}'[t'_0]$□

---

case $B2$

by inversion $t_2 \equiv \mathbb{E}[t_0]$ $(HE1)$

$t'_2 \equiv \mathbb{E}[t'_0]$ $(HE2)$

$H \rhd t_0 \leadsto^p H' \rhd t'_0$ (HPR)

by HE1,HE2 t.s. $H \rhd t_1 := \mathbb{E}[t_0] \leadsto H' \rhd t_1 := \mathbb{E}[t'_0]$

by HPR with $\mathbb{E}' = t_1 := \mathbb{E}$: $H \rhd \mathbb{E}'[t_0] \leadsto H' \rhd \mathbb{E}'[t'_0]$□

2 Program equivalence

For each of these programs, tell if they are equivalent or not. If they are equivalent, show what they reduce to, no matter the input. If they are not, argue why and if possible, show a context that tells them apart.

1. - $z : Ref(\mathbb{N} \to \mathbb{N})$
   - $t_1 = \lambda x : \mathbb{N}.!z\ 0; 2 + x$
   - $t_2 = \lambda x : \mathbb{N}.\ \text{if } x > 0 \text{ then } x + 2 \text{ else } !z\ x; x + 2$

   Not equivalent. Can be distinguished by this context:

   $$\begin{aligned}&\text{let } d = \text{ new } 0 \text{ in}\\&\text{let } z = \text{ new } (\lambda x : \mathbb{N}.d := 1; 12) \text{ in}\\&[\cdot]\ 0;\ \text{if } !d == 0 \text{ then } 0 \text{ else } \omega\end{aligned}$$

2. - $t_1 = \text{ let } x = \lambda y : \forall\alpha.\alpha \to \alpha.\lambda z : \mathbb{N}.y[\mathbb{N}]\ (z + 1) \text{ in } x$
   - $t_2 = \lambda y : \forall\alpha.\alpha \to \alpha.\lambda x : \mathbb{N}.(y[\mathbb{N}]\ x) + 1$

   Equivalent. In both cases the function passed to t is an identity function, and we can't differentiate the two since they both return the input increased by 1.
   We could try differentiating the two on the basis on of a side effect provoked inside the function we pass to the term but:

   - The function is always called, so we can't use the same method as in case 1.
   - The function is polymorphic, so we can't inspect the argument that it receives

3. - $f : (Ref\ \mathbb{N}) \to \mathbb{N}$
   - $t_1 = \left|\begin{array}{l}\text{let } x = new\ 0 \text{ in}\\ f\ x;\\ !x\end{array}\right.$
   - $t_2 = \left|\begin{array}{l}\text{let } x = new\ 1 \text{ in}\\ f\ (new\ 0);\\ x := (!x - 1)\end{array}\right.$

   Not equivalent.

   $$\begin{aligned}&\text{let } f = \lambda d : Ref\ \mathbb{N}.d := 12; 1 \text{ in}\\&\text{if } [\cdot] == 0 \text{ then } 0 \text{ else } \omega\end{aligned}$$

   Assuming the value of the update is 0 as stated in class.

4. - $r : Ref(\mathbb{N})$
   - $t_1 = \left|\begin{array}{l}\text{let } x =!r \text{ in}\\ \text{let } y = \text{ new } x \text{ in}\\ r :=!y;\\ !y\end{array}\right.$
   - $t_2 = \left|\begin{array}{l}\text{let } x = \text{ new } 0 \text{ in}\\ \text{let } y =!x; !r \text{ in}\\ y\end{array}\right.$

   Equivalent. Both reduce to the initial value of r.

5. - $t_1 = \lambda x : \mathbb{N}.\langle x, 1\rangle.1$
   - $t_2 = \text{ let } x = \Lambda\alpha.\lambda x : \alpha.x \text{ in } x$

   Not equivalent. $t_2\ true$ reduces to $true$ while $t_1\ true$ is ill-typed.

3 A private memory for ASM

Add a private memory to ASM. The domain of the memory becomes integers, so positive and negative numbers. Negative integers represent a private memory.
Modify the semantics of ASM to reflect the following access control policy: If the program counter is within the address range 0 to 100, then any read or write to the private memory succeeds, otherwise any read or write returns 0.

$$\begin{aligned}
t ::=& r := i\\
&|sum\ r\ r\\
&|sub\ r\ r\\
&|cmp\ r\ r\\
&|jmp\ r\\
&|jiz\ r\\
&|jeq\ r\\
&|read\ r\ r\\
&|write\ r\ r\\
r ::=& ar|br|cr|dr|er|fr|gr|hr\\
&|ir|jr|kr|lr|mr|nr|or\\
C ::=& \emptyset|C, \mathbb{I} \mapsto t\\
F ::=& \emptyset|F, \mathbb{I} \mapsto b\\
R ::=& \emptyset|R, r \mapsto \mathbb{I}\\
M ::=& \emptyset|M, \mathbb{I} \mapsto \mathbb{I}
\end{aligned}$$

Judgement:

$$i; C; R; F; M \rightrightarrows i; C; R; F; M$$

Codebase, registers, flags, and memory:

$$\frac{C = C', n \mapsto t}{C(n) = t} \qquad \frac{R = R', r \mapsto n}{R(r) = n} \qquad \frac{F = F', n \mapsto B}{F(n) = B}$$

$$\frac{C = C', n' \mapsto \_ \quad C'(n) = t}{C(n) = t} \qquad \frac{R = R', r' \mapsto \_ \quad R'(r) = n}{R(r) = n} \qquad \frac{F = F', n' \mapsto \_ \quad F'(n) = B}{F(n) = B}$$

$$\frac{M = M', n \mapsto n'}{M(n) = n'} \qquad \frac{M = M', n” \mapsto \_ \quad M'(n) = n'}{M(n) = n'}$$

Rules:

$$\frac{i \geq 0 \quad C(i) = r_1 := i_1 \quad R' = R, r_1 \mapsto i_1 \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R'; F; M}\text{load}$$

$$\frac{i \geq 0 \quad C(i) = sum\ r_1\ r_2\ R(r_1) = i_1\ R(r_2) = i_2\ R' = R, r_1 \mapsto i_1 + i_2\ i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R'; F; M}\text{sum}$$

$$\frac{i \geq 0 \quad C(i) = sub\ r_1\ r_2\ R(r_1) = i_1\ R(r_2) = i_2\ R' = R, r_1 \mapsto i_1 - i_2\ i' = i + 1\ F' = F, 0 \mapsto i_2 > i_1}{i; C; R; F; M \rightrightarrows i'; C; R'; F'; M}\text{sub}$$

$$\frac{i \geq 0 \quad C(i) = cmp\ r_1\ r_2\ R(r_1) = i_1\ R(r_2) = i_2\ F' = F, 1 \mapsto i_1 == i_2\ i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R; F'; M}\text{cmp}$$

$$\frac{i \geq 0 \quad C(i) = jmp\ r_1 \quad R(r_1) = i_1 \quad i' = i_1}{i; C; R; F; M \rightrightarrows i'; C; R; F; M}\text{jmp}$$

$$\frac{i \geq 0 \quad C(i) = jiz\ r_1 \quad R(r_1) = i_1 \quad F(1) = b \quad i' = \text{ if } b \text{ then } i_1 \text{ else } i + 1}{i; C; R; F; M \rightrightarrows i'; C; R; F; M}\text{jiz}$$

$$\frac{i \geq 0 \quad C(i) = jeq\ r_1 \quad R(r_1) = i_1 \quad F(0) = b \quad i' = \text{ if } b \text{ then } i_1 \text{ else } i + 1}{i; C; R; F; M \rightrightarrows i'; C; R; F; M}\text{jeq}$$

$$\frac{0 \leq i \leq 100 \quad C(i) = read\ r_1\ r_2 \quad R(r_2) = i_2 \quad M(i_2) = i” \quad R' = R, r_1 \mapsto i” \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R'; F; M}\text{read-private}$$

$$\frac{i > 100 \quad C(i) = read\ r_1\ r_2 \quad R(r_2) = i_2 \quad i_2 \geq 0 \quad M(i_2) = i” \quad R' = R, r_1 \mapsto i” \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R'; F; M}\text{read-public-☺}$$

$$\frac{i > 100 \quad C(i) = read\ r_1\ r_2 \quad R(r_2) = i_2 \quad i_2 < 0 \quad R' = R, r_1 \mapsto 0 \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R'; F; M}\text{read-public-☹}$$

$$\frac{0 \leq i \leq 100 \quad C(i) = write\ r_1\ r_2 \quad R(r_1) = i_1 \quad R(r_2) = i_2 \quad M' = M, i_1 \mapsto i_2 \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R; F; M'}\text{write-private}$$

$$\frac{i > 100 \quad C(i) = write\ r_1\ r_2 \quad R(r_1) = i_1 \quad R(r_2) = i_2 \quad i_1 \geq 0 \quad M' = M, i_1 \mapsto i_2 \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R; F; M'}\text{write-public-☺}$$

$$\frac{i > 100 \quad C(i) = write\ r_1\ r_2 \quad R(r_1) = i_1 \quad R(r_2) = i_2 \quad i_1 < 0 \quad i' = i + 1}{i; C; R; F; M \rightrightarrows i'; C; R; F; M}\text{write-public-☹}$$

4 Labelled ASM Functions

Add named functions to ASM. ASM programs become a list mapping names to codebases. ASM instructions now include

1. calling a function whose name is statically known
2. calling a function by jumping to the address where it starts (the address value is read from a register)
3. returning from a called function

$$\begin{aligned} t &::= \cdots | fun\ r | name | ret \\ \mathcal{F} &::= \emptyset | \mathcal{F}, name \mapsto \mathbb{I} \\ CS &::= \emptyset | CS, \mathbb{I} \\ name &::= a|b|\ldots|z|namename \end{aligned}$$

$$\frac{\mathcal{F} = \mathcal{F}', name \mapsto i}{\mathcal{F}(name) = i} \qquad \frac{\mathcal{F} = \mathcal{F}', name' \mapsto \_ \quad \mathcal{F}'(name) = i}{\mathcal{F}(name) = i}$$

Judgement:

$$i; C; R; F; M; CF \rightrightarrows i; C; R; F; M; CS$$

Old rules get a call stack and a function names list added to them that are never used or modified

$$\frac{i \geq 0 \quad C(i) = fun\ r_1 \quad R(r_1) = i_1 \quad i” = i + 1 \quad CS' = CS, i” \quad i' = i_1}{i; C; R; F; M; \mathcal{F}; CS \rightrightarrows i'; C; R; F; M; \mathcal{F}; CS'}\text{func-jmp}$$

$$\frac{i \geq 0 \quad C(i) = ret \quad CS = CS_0, i_1 \quad CS' = CS_0 \quad i' = i_1}{i; C; R; F; M; \mathcal{F}; CS \rightrightarrows i'; C; R; F; M; \mathcal{F}; CS'}\text{ret}$$

$$\frac{i \geq 0 \quad C(i) = name \quad \mathcal{F}(name) = i_1 \quad i” = i + 1 \quad CS' = CS, i” \quad i' = i_1}{i; C; R; F; M; \mathcal{F}; CS \rightrightarrows i'; C; R; F; M; \mathcal{F}; CS'}\text{func}$$