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

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

2. allocation

$$\begin{aligned} &\text{w.h. } H : \Sigma \text{ and } \Sigma, \emptyset, \emptyset \vdash \text{ new } t_0 : ref(\tau) \\ &\text{by IH w.h. if } H_0 : \Sigma_0 \text{ and } \Sigma_0, \emptyset, \emptyset \vdash t_0 : \tau \text{ then either } t_0.val \text{ or } \exists t_0', H_0'.H_0 \triangleright t_0 \rightsquigarrow H_0' \triangleright t_0' \\ &\text{case } t_0.val\text{: } H \triangleright \text{ new } t_0 \rightsquigarrow H, l \mapsto t_0 \triangleright l \square \\ &\text{case } \exists t_0', H_0'.H_0 \triangleright t_0 \rightsquigarrow H_0' \triangleright t_0'\text{: } H \triangleright \text{ new } t_0 \rightsquigarrow H \triangleright \text{ new } t_0' \square \end{aligned}$$

3. dereferencing
4. update

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

   Not equivalent. Can be distinguished by this context:

$$\begin{aligned} &\text{let } d = \text{ new } 0 \text{ in} \\ &\text{let } z = \lambda x : \mathbb{N}.d := 1; 12 \text{ in} \\ &t_{1/2}\ 0;\ \text{if } !d == 0 \text{ then } 0 \text{ else } \omega \end{aligned}$$

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

3. - $f : (Ref\ \mathbb{N}) \to \mathbb{N}$
   - $t_1 =$ let $x = new\ 0$ in $f\ x; !x$
   - $t_2 =$ let $x = new\ 1$ in $f\ (new\ 0); x := (!x - 1)$

4. - $r : Ref(\mathbb{N})$

- $t_1 =$
$$\begin{array}{l} \text{let } x = !r \text{ in} \\ \text{let } y = \text{ new } x \text{ in} \\ r := !y; \\ !y \end{array}$$
- $t_2 =$
$$\begin{array}{l} \text{let } x = \text{ new } 0 \text{ in} \\ \text{let } y = !x; !r \text{ in} \\ y \end{array}$$

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.

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