Homework 4

Due: Monday, April 02, 11:59pm
Extension: Wednesday, April 04, 11:59pm

Preface

Discussing high-level approaches to homework problems with your peers is encouraged. You must include at the top of all of your assignment documents a Collaboration Statement which declares any other people with whom you discussed homework problems. For example:

> *Collaboration Statement: I discussed problems 1 and 3 with Jamie Smith. I discussed problem 2 with one of the TAs. I discussed problem 4 with a personal tutor.*

If you did not discuss the assignment with anyone, you still must declare:

> *Collaboration Statement: I did not discuss homework problems with anyone.*

Copying answers or doing the work for another student is not allowed. Write your Collaboration Statement at the top of your hw4.ml file. Please do not write your Collaboration Statement in the text of an email; you must include it directly it in your assignment.

For the programming portion, you must write your name at the top *and the bottom* of your submitted *.ml file in an OCaml comment. E.g.:

```
<beginning of file>
(* Name: Jamie Smith *)
... your solutions to the assignment ...
(* Name: Jamie Smith *)
<end of the file>
```

You may do this assignment either by yourself or in a group of two people. If you choose to do this assignment as a group, submit *only one* copy of the assignment, with the names of both team member at the top and bottom of the file.

Submitting

Programming Portion

Prepare the programming portion of your assignment in a text editor of your choice. You will be working from a provided file named `hw4.ml`. Do not change the file name when you submit your assignment.

Submit the programming portion of your assignment by emailing your `hw4.ml` file to me: David.Darais@uvm.edu with "CS 225 HW4" in the subject line.

If you are doing this assignment with a partner, only submit one file for both team members. In the email where you submit your assignment, let me know that it is a group submission, *e.g.*, by writing "This submission is for Jamie Smith and Alex Williams"

Programming Portion (100 Points)

Download hw4.zip from the course webpage and extract it to a folder on your machine. Ensure that you are able to run the following commands from the directory without error:

```
> make
> make hw4
```

Running `make` should print out a log of compiling various OCaml files. Running `make hw4` should run a bunch of tests and report that 7 passed, 0 failed, and 43 are still "todo". In order for the `merlin` plugin to work, you must first run `make`.

Your assignment is to complete the definitions of `step` and `infer` basd on the small-step semantics and typing relations defined in the next section of this writeup. These are also described in the book (TAPL) Chapter 13. The primary difference between the book and this writeup is that the book includes both a type environment Γ and store typeing Σ in its typing rules. In this assignment we have simplified the language such that a type environment Γ is not necessary because there are no variables, let expressions or lambda expressions. The book notates the small-step rules:

$$t \mid \mu \longrightarrow t \mid \mu$$

for expression t and stores μ, whereas in this writeup we instead write:

$$\langle e, \sigma \rangle \longrightarrow \langle e, \sigma \rangle$$

for expression e and stores σ.

Look in the assignment file for `raise TODO` expressions and replace them with your solution.

At the end of `hw4.ml` is a test suite which will test your functions on a few test cases. Feel free to add your own tests. Just because you pass the provided tests doesn't guarantee your solution is correct.

Reference

Syntax for expressions with boolean expressions, products, and reference:

$$\begin{array}{rcl} \ell \in \texttt{loc} & ::= & \{\ldots, -1, 0, 1, \ldots\} \approx \mathbb{N} \\ e \in \texttt{exp} & ::= & \texttt{T} \mid \texttt{F} \mid \texttt{if}(e)\{e\}\{e\} \\ & \mid & \langle e, e\rangle \mid \texttt{projl}(e) \mid \texttt{projr}(e) \\ & \mid & \texttt{ref}(e) \mid !e \mid e := e \mid e\ ;\ e \\ & \mid & \texttt{loc}(\ell) \end{array}$$

Syntax for values:

$$\begin{array}{rcl} v \in \texttt{val} & ::= & \texttt{T} \mid \texttt{F} \\ & \mid & \langle v, v\rangle \\ & \mid & \texttt{loc}(\ell) \end{array}$$

A store is a partial map from locations ℓ to values v:

$$\sigma \in \texttt{store} := \texttt{loc} \rightharpoonup \texttt{val}$$

The notation $\sigma[\ell \mapsto v]$ constructs a new store σ' s.t.:

$$\begin{array}{lll} \sigma'(\ell') = v & \text{when} & \ell' = \ell \\ \sigma'(\ell') = \sigma(\ell') & \text{when} & \ell' \neq \ell \end{array}$$

Fresh locations in the store are defined deterministically as the smallest location not currently in the store, which is equivalent to $1 + \ell$ when ℓ is the largest location in the store (or 0 if nothing is in the store):

$$\text{fresh}(\sigma) := 1 + \max(\{\ell \mid \ell \in \text{dom}(\sigma)\} \cup \{0 \mid \sigma = \varnothing\})$$

The one-step semantics relation:

$$\boxed{\langle e, \sigma\rangle \longrightarrow \langle e, \sigma\rangle}$$

$$\frac{}{\langle \texttt{if}(\texttt{T})\{e_2\}\{e_3\}, \sigma\rangle \longrightarrow \langle e_2, \sigma\rangle} \qquad \frac{}{\langle \texttt{if}(\texttt{F})\{e_2\}\{e_3\}, \sigma\rangle \longrightarrow \langle e_3, \sigma\rangle}$$

$$\frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle \texttt{if}(e_1)\{e_2\}\{e_3\}, \sigma\rangle \longrightarrow \langle \texttt{if}\{e_1'\}\{e_2\}\{e_3\}, \sigma'\rangle} \qquad \frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle\langle e_1, e_2\rangle, \sigma\rangle \longrightarrow \langle\langle e_1', e_2\rangle, \sigma'\rangle}$$

$$\frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle}{\langle\langle v, e\rangle, \sigma\rangle \longrightarrow \langle\langle v, e'\rangle, \sigma'\rangle} \qquad \frac{}{\langle \texttt{projl}(\langle v_1, v_2\rangle), \sigma\rangle \longrightarrow \langle v_1, \sigma'\rangle} \qquad \frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle}{\langle \texttt{projl}(e), \sigma\rangle \longrightarrow \langle \texttt{projl}(e'), \sigma'\rangle}$$

$$\frac{}{\langle \texttt{projr}(\langle v_1, v_2\rangle), \sigma\rangle \longrightarrow \langle v_2, \sigma'\rangle} \qquad \frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle}{\langle \texttt{projr}(e), \sigma\rangle \longrightarrow \langle \texttt{projr}(e'), \sigma'\rangle}$$

$$\frac{\ell = \text{fresh}(\sigma)}{\langle \texttt{ref}(v), \sigma\rangle \longrightarrow \langle \texttt{loc}(\ell), \sigma[\ell \mapsto v]\rangle} \qquad \frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle}{\langle \texttt{ref}(e), \sigma\rangle \longrightarrow \langle \texttt{ref}(e'), \sigma'\rangle} \qquad \frac{}{\langle !\texttt{loc}(\ell), \sigma\rangle \longrightarrow \langle \sigma(\ell), \sigma\rangle}$$

$$\frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle}{\langle !e, \sigma\rangle \longrightarrow \langle !e', \sigma'\rangle} \qquad \frac{}{\langle \texttt{loc}(\ell) := v, \sigma\rangle \longrightarrow \langle \texttt{T}, \sigma[\ell \mapsto v]\rangle} \qquad \frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle e_1 := e_2, \sigma\rangle \longrightarrow \langle e_1' := e_2, \sigma'\rangle}$$

$$\frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma'\rangle}{\langle \texttt{loc}(\ell) := e, \sigma\rangle \longrightarrow \langle \texttt{loc}(\ell) := e', \sigma'\rangle} \qquad \frac{}{\langle v\ ;\ e, \sigma\rangle \longrightarrow \langle e, \sigma\rangle} \qquad \frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma'\rangle}{\langle e_1\ ;\ e_2, \sigma\rangle \longrightarrow \langle e_1'\ ;\ e_2, \sigma'\rangle}$$

Syntax for types:

$$\tau \in \texttt{type} ::= \texttt{bool} \mid \tau \times \tau \mid \texttt{ref}(\tau)$$

A store typing is a partial map from locations ℓ to types τ:

$$\Sigma \in \texttt{store-typing} := \texttt{loc} \rightharpoonup \texttt{type}$$

The notation $\Sigma[\ell \mapsto \tau]$ constructs a new store Σ' s.t.:

$$\begin{array}{lll} \Sigma'(\ell') = \tau & \text{when} & \ell' = \ell \\ \Sigma'(\ell') = \Sigma(\ell') & \text{when} & \ell' \neq \ell \end{array}$$

The type system:

$\boxed{\Sigma \vdash e : \tau}$

$$\frac{}{\Sigma \vdash \texttt{T} : \texttt{bool}} \qquad \frac{}{\Sigma \vdash \texttt{F} : \texttt{bool}} \qquad \frac{\Sigma \vdash e_1 : \texttt{bool} \qquad \Sigma \vdash e_2 : \tau \qquad \Sigma \vdash e_2 : \tau}{\Sigma \vdash \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

$$\frac{\Sigma \vdash e_1 : \tau_1 \qquad \Sigma \vdash e_2 : \tau_2}{\Sigma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \qquad \frac{\Sigma \vdash e : \tau_1 \times \tau_2}{\Sigma \vdash \texttt{projl}(e) : \tau_1} \qquad \frac{\Sigma \vdash e : \tau_1 \times \tau_2}{\Sigma \vdash \texttt{projr}(e) : \tau_2} \qquad \frac{\Sigma \vdash e : \tau}{\Sigma \vdash \texttt{ref}(e) : \texttt{ref}(\tau)}$$

$$\frac{\Sigma \vdash e : \texttt{ref}(\tau)}{\Sigma \vdash !e : \tau} \qquad \frac{\Sigma \vdash e_1 : \texttt{ref}(\tau) \qquad \Sigma \vdash e_2 : \tau}{\Sigma \vdash e_1 := e_2 : \texttt{bool}} \qquad \frac{\Sigma \vdash e_1 : \tau_1 \qquad \Sigma \vdash e_2 : \tau_2}{\Sigma \vdash (e_1 \; ; e_2) : \tau_2} \qquad \frac{\Sigma(\ell) = \tau}{\Sigma \vdash \texttt{loc}(\ell) : \tau}$$