15-312 Foundations of Programming Languages
Recitation 9: More on Storage

Daniel Spoonhower
spoons+@cs

October 29, 2003

1 Assignment 5

First, note a couple of common mistakes.

In the general monadic framework, we have two forms of transitions,

$$e \mapsto e'$$
$$\langle w, m\rangle \mapsto \langle w', m'\rangle$$

The first is for pure expressions, and the second for monadic expressions. In lecture, we discussed a monadic formulation of mutable store, so our worlds w took the form of maps M from locations to values called stores. In this assignment, we are attempting to model input and output, and so our worlds correspond to states of the inputs and output streams. While stores could contain values of different types, our input and output streams contain only integers. So while we required a context Λ for giving types to each location in the store, we don't require any context for the input and output.

Second, despite my warnings about verifying that your programs typecheck, many people submitted programs that did not.

```
if eof then ...
```

However, recall the typing rule for `if`:

$$\frac{\Gamma \vdash e : \texttt{bool} \quad \cdots}{\Gamma \vdash \texttt{if}(e, e_1, e_2) : \tau}$$

And our typing rule for `eof`,

$$\frac{}{\Gamma \vdash \texttt{eof} \div \texttt{bool}}$$

In other words, `if` expects a pure expression of type `bool`, not a monadic expression, such as `eof`.

2 More on the G Machine

Recall our transition rules for the G machine.

$$\frac{}{(H_f[l = V], S \cup \{l\}, H_t) \mapsto_{\mathsf{g}} (H_f, S \cup FLLV(V), H_t[l = V])}\ Copy$$

$$\frac{}{(H_f, S \cup \{l\}, H_t[l = V]) \mapsto_{\mathsf{g}} (H_f, S, H_t[l = V])}\ Discard$$

We would like some assurance that our collector is doing the right thing. First, we would like to show that it terminates, but before we do that, we should take note of the following fact.

Theorem 1 (Consistency). *For all $l \in \mathrm{dom}(H)$, either $l \in H_f$ or $l \in H_t$ but not both.*

Proof. By inspection of the transition rules for the G machine.

Notice that we begin with $H_f = H$ and $H_t = \emptyset$, so the condition is clearly true when we invoke the G machine. Then note that the first rule moves a location l from H_f to H_t, while the second rule leaves both H_f and H_t unchanged. □

Theorem 2 (Termination). *For well-formed H, k, η and e, the G machine terminates in a finite number of steps.*

Proof. By induction on the sizes of H_f and S, ordered lexicographically. (Noting that the state of our machine is always, by construction, of finite size.)

Case $H_f = \emptyset$ and $S = \emptyset$. Then the algorithm terminates, as we have finished as soon as $S = \emptyset$.

Case $H_f = \emptyset$ and $S = S' \cup \{l\}$. In this case only the second rule applies, in which case we decrease the size of the scan set by one. By applying the induction hypothesis with H_f and S', we see that the algorithm terminates.

Case $H_f = H'_f[l = V]$ and $S = \emptyset$. Again, since S is empty, the algorithm has finished.

Case $H_f = H'_f[l = V]$ and $S = S' \cup \{l'\}$. Both rules might apply, but according to our previous theorem, a given location l appears in exactly one of H_f and H_t.

- Subcase *Copy*. In this case the size of the *from-space* is decreasing by one, so (regardless of changes to the scan set) we can apply the induction hypothesis and conclude that the algorithm terminates.

- Subcase *Discard*. Here the size of H_f remains the same, while the size of the scan set decreases by one. Again, we apply the induction hypothesis with H_f and S' and conclude that the algorithm terminates.

□

In describing termination, we required that the machine state be "well-formed" but we haven't yet defined well-formed. We will use the judgment "s ok" to indicate that a machine state s is properly constructed and well-typed. This judgment will be expanded in the proof below.

In order to proof preservation, we will need to give a type to each location; just as we gave a type to locations in our description of references we will add a new type to describe locations in the heap and a form of typing to describe the heaps themselves.

$$\begin{array}{llll} \textit{(types)} & \tau & ::= & \ldots \mid \tau\ \texttt{loc} \\ \textit{(heap typing)} & \Lambda & ::= & \cdot \mid \Lambda, l : \tau \end{array}$$

In the small part of the proof we will see below, we will also need a few specific rules about frames that manipulate locations and about the heap.

$$\frac{}{\texttt{fst}(\Box) : (\tau_1 * \tau_2)\ \texttt{loc} \rightarrow \tau_1\ \mathsf{frame}}$$

$$\frac{(l : \tau) \in \Lambda \quad H : \Lambda \quad (l = V) \in H}{\Lambda; \cdot \vdash V : \tau} \qquad \frac{(l : \tau) \in \Lambda}{\Lambda; \Gamma \vdash l : \tau\ \texttt{loc}}$$

Finally, we will need to extend our value inversion lemma with the following clause:

- If v value and $\Lambda; \cdot \vdash v : \tau\ \texttt{loc}$ then $v = l$ for some location l and $(l : \tau) \in \Lambda$.

Our statement of preservation will be similar to that for the E machine.

Theorem 3 (Preservation). *If s ok and $s \mapsto_{\mathsf{a}} s'$ then s' ok.*

Proof. By rule induction on the derivation of $s \mapsto_{\mathsf{a}} s'$.

We will consider just one case, the case for `fst`.

$$\frac{}{H[l = \texttt{pair}(v_1, v_2)] \mid k \triangleright \texttt{fst}(\Box) \mid \eta < l \mapsto_{\mathsf{a}} H[l = \texttt{pair}(v_1, v_2)] \mid k \mid \eta < v_1}\ \mathit{Fst}$$

Case *Fst* $s = H[l = \texttt{pair}(v_1, v_2)] \mid k \triangleright \texttt{fst}(\Box) \mid \eta < l$
$s' = H[l = \texttt{pair}(v_1, v_2)] \mid k \mid \eta < v_1$

s ok	Assumption
$\eta : \Gamma$	Definition of s ok*
$H[l = \texttt{pair}(v_1, v_2)] : \Lambda$	"*
$\Lambda; \Gamma \vdash k \triangleright \texttt{fst}(\Box) : \tau$ stack	"

l value	”
$\Lambda; \cdot \vdash l : \tau$	”
$\Lambda; \Gamma \vdash k : \tau_1$ stack	Inversion on stack typing*
$\Lambda; \Gamma \vdash \texttt{fst}(\square) : \tau \rightarrow \tau_1$ frame	”
$\tau = (\tau_1 * \tau_2)$ `loc`	Inspection of frame typing rule
l is a location and $(l : \tau_1 * \tau_2) \in \Lambda$	By value inversion
$\Lambda; \cdot \vdash \texttt{pair}(v_1, v_2) : \tau_1 * \tau_2$	By rule
v_1 value	By construction of pairs*
$\Lambda; \cdot \vdash v_1 : \tau_1$	Inversion on pair typing*
s' ok	From the marked statements above

□

Finally, we would also like to know that our A machine preserves the meaning of programs according to our previous definitions of the dynamic semantics of MinML. To do so, we will need to show, for example, that for each step taken by a the E machine is *simulated* by one or more steps of the A machine.