Scribe: Srijuntongsiri, Hackett Lecturer: Radu Rugina

1 References

In this lecture, we explore the semantics of dynamic allocation and imperative features. For this, we extend our applicative language F with ML-style reference cells and statement sequences; call the new language F_{ref}. The syntax for constructs that manipulate references is:

$$e \;::=\; \dots \;\mid\; \mathsf{ref}\; e \;\mid\; !e \;\mid\; e_0 := e_1 \;\mid\; e_0; e_1$$

Informally, these expressions have the following meaning:

$$\begin{array}{rcl}
\mathsf{ref}\; e & \Rightarrow & \text{Dynamically creates a location and initializes it with the value of } e \\
!e & \Rightarrow & \text{Dereferences } e \text{ and yields the value of the location that it represents} \\
e_0 := e_1 & \Rightarrow & \text{Updates the location } e_0 \text{ with the value of } e_1, \\
 & & \text{but the overall expression does not yield a result} \\
e_0; e_1 & \Rightarrow & \text{Evaluates } e_0 \text{ and then } e_1 \text{, does not yield a result}
\end{array}$$

The fact that locations are mutable (they can be updated using assignments) introduces the concept of *state*, which is a map from locations to their values. Note that a number of constructs have *side-effects*, because they modify the state of the program: dynamic allocation constructs ($\mathsf{ref}\; e$) extend the state with mappings for the newly created locations; and assignments ($e_0 := e_1$) update the values of existing cells. A statement sequence $e_0; e_1$ is evaluated only for its side-effects.

A natural question is what statements like sequences evaluate to? We don't want their value to be an error, but we don't want to get an F value (integer, boolean, pair, or function) either. Therefore, we introduce a new value unit and include it in the syntax:

$$e ::= \dots \;\mid\; \mathsf{unit}$$

The constant value unit represents the result for expressions that are evaluated for their side-effects, such as assignments or sequences.

2 Operational Semantics

We present here a small-step operational semantics for F_{ref}. The operational semantics must now take into account the concept of state. We therefore use the following evaluation relation:

$$(e, \sigma) \rightarrow_1 (e', \sigma')$$

The evaluation of all the constructs common to F and F_{ref} is straightforward. For these constructs, the F_{ref} evaluation rules are similar to the evaluations in F, except that now we must keep track of state changes because of expressions with side-effects. We give below the evaluation rules for pairs – the rules for the other constructs are similar. For comparison purposes, we present the evaluations in both languages (the annotation shows what language they refer to).

$$\frac{e_0 \rightarrow_1 e_0'}{\langle e_0, e_1\rangle \rightarrow_1 \langle e_0', e_1\rangle}\; F \qquad\qquad \frac{(e_0, \sigma) \rightarrow_1 (e_0', \sigma')}{(\langle e_0, e_1\rangle, \sigma) \rightarrow_1 (\langle e_0', e_1\rangle, \sigma')}\; F_{\text{ref}}$$

$$\frac{e_1 \rightarrow_1 e_1'}{\langle v, e_1\rangle \rightarrow_1 \langle v, e_1'\rangle}\; F \qquad\qquad \frac{(e_1, \sigma) \rightarrow_1 (e_1', \sigma')}{(\langle v, e_1\rangle, \sigma) \rightarrow_1 (\langle v, e_1'\rangle, \sigma')}\; F_{\text{ref}}$$

The semantics for creating, accessing, or updating reference cells, and for sequences follow their informal description. For each of them, we evaluate the subexpressions to locations or values, then perform the appropriate update or access to the state information. For assignments and sequences, the semantics enforce a left-to-right evaluation.

- Allocation of reference cells:

$$\frac{(e, \sigma) \to_1 (e', \sigma')}{(\mathsf{ref}\ e, \sigma) \to_1 (\mathsf{ref}\ e', \sigma')} \qquad \frac{}{(\mathsf{ref}\ v, \sigma) \to_1 (l, \sigma[l \to v])}\ l \notin dom(\sigma)$$

- Dereferencing cells:

$$\frac{(e, \sigma) \to_1 (e', \sigma')}{(!e, \sigma) \to_1 (!e', \sigma')} \qquad \frac{}{(!l, \sigma) \to_1 (\sigma(l), \sigma)}$$

- Updating reference cells:

$$\frac{(e_0, \sigma) \to_1 (e_0', \sigma')}{(e_0 := e_1, \sigma) \to_1 (e_0' := e_1, \sigma')} \qquad \frac{(e_1, \sigma) \to_1 (e_1', \sigma')}{(l := e_1, \sigma) \to_1 (l := e_1', \sigma')}$$

$$\frac{}{(l := v, \sigma) \to_1 (\mathsf{unit}, \sigma[l \to v])}$$

- Expression sequences:

$$\frac{(e_0, \sigma) \to_1 (e_0', \sigma')}{(e_0; e_1, \sigma) \to_1 (e_0'; e_1, \sigma')} \qquad \frac{(e_1, \sigma) \to_1 (e_1', \sigma')}{(v_0; e_1, \sigma) \to_1 (v_0; e_1', \sigma')}$$

$$\frac{}{(v_0; v_1, \sigma) \to_1 (\mathsf{unit}, \sigma)}$$

Note that operational semantics handles errors poorly, as it just enters a stuck configuration when one occurs. The denotational semantics presented next handles errors in referencing constructs explicitly.

3 Denotational Semantics

To account for state, locations and the unit value, we extend our semantic domains appropriately. We include locations and unit in the domain of values; the domain for results is a cartesian product of values and states; and functions take as arguments an input value and a state:

$$\begin{aligned}
v \in \mathit{Value} &= \mathbb{Z} + \mathbb{T} + \mathbb{U} + \mathit{Pair} + \mathit{Function} + \mathit{Location} \\
p \in \mathit{Pair} &= \mathit{Value} \times \mathit{Value} \\
f \in \mathit{Function} &= \mathit{Value} \to \mathit{State} \to \mathit{Result} \\
r \in \mathit{Result} &= ((\mathit{Value} \times \mathit{State}) + \mathit{Error})_\perp \\
\mathbb{U} &= \{\mathit{unit}\}
\end{aligned}$$

An environment ρ maps each variable to a value or an error; a state σ maps each location to a value or an error. Intuitively, think of the environment as being the current values of stack variables, and the state as being the current values of all heap locations. The denotation $\mathcal{C}[\![e]\!]$ then takes an environment and a state an produces a result for the evaluation of e:

$$\begin{aligned}
\rho \in \mathit{Env} &= \mathit{Var} \to (\mathit{Value} + \mathit{Error}) \\
\sigma \in \mathit{State} &= \mathit{Location} \to (\mathit{Value} + \mathit{Error}) \\
\mathcal{C}[\![e]\!] &\in \mathit{Env} \to \mathit{State} \to \mathit{Result}
\end{aligned}$$

Using these domains, we define the denotation $\mathcal{C}[\![e]\!]\rho\sigma$ of an expression e, given the environment ρ and the state σ. We use the *ecase* construct for error-handling (recall that *ecase* yields $\perp$ if its argument is $\perp$ and yields an error for unmatched patterns):

$$\begin{array}{rcl}
\mathcal{C}[\![n]\!]\rho\sigma & = & (\mathbb{Z}(n), \sigma) \\[1em]
\mathcal{C}[\![x]\!]\rho\sigma & = & (\rho(x), \sigma) \\[1em]
\mathcal{C}[\![\textsf{if}\ e_0\ \textsf{then}\ e_1\ \textsf{else}\ e_2]\!]\rho\sigma & = & \mathit{ecase}\ \ \mathcal{C}[\![e_0]\!]\rho\sigma\ \mathit{of}\ \ (\mathbb{T}(\mathit{true}), \mathit{State}(\sigma')) \Rightarrow \mathcal{C}[\![e_1]\!]\rho\sigma' \\
 & & \qquad\qquad\qquad\quad \mid\ \ (\mathbb{T}(\mathit{false}), \mathit{State}(\sigma')) \Rightarrow \mathcal{C}[\![e_2]\!]\rho\sigma' \\[1em]
\mathcal{C}[\![\lambda x.e]\!]\rho\sigma & = & \lambda v.\, \lambda\sigma_{dyn}.\, \mathcal{C}[\![e]\!]\rho[x \rightarrow v]\sigma_{dyn} \\[1em]
\mathcal{C}[\![e_0\ e_1]\!]\rho\sigma & = & \mathit{ecase}\ \ \mathcal{C}[\![e_0]\!]\rho\sigma\ \mathit{of}\ \ (\mathit{Function}(f), \mathit{State}(\sigma')) \Rightarrow \\
 & & \mathit{ecase}\ \ \mathcal{C}[\![e_1]\!]\rho\sigma'\ \mathit{of}\ \ (\mathit{Value}(v), \mathit{State}(\sigma'')) \Rightarrow f\ v\ \sigma'' \\[1em]
\mathcal{C}[\![\textsf{rec}\ y\ \ (\lambda x.\ e)]\!]\rho\sigma & = & \mathit{fix}(\lambda f \in \mathit{Function}.\mathcal{C}[\![\lambda x.e]\!]\rho[y \rightarrow f]\sigma)
\end{array}$$