Modular Analysis

Joonhyup Lee

February 19, 2024

1 Syntax and Semantics

1.1 Abstract Syntax

$$
\begin{array}{rccll}
\text{Identifiers} & x & \in & \text{Var} & \\
\text{Expression} & e & \rightarrow & x \mid \lambda x.e \mid e\ e & \lambda\text{-calculus} \\
 & & \mid & e \rtimes e & \text{linked expression} \\
 & & \mid & \varepsilon & \text{empty module} \\
 & & \mid & x = e\ ;\ e & \text{(recursive) binding}
\end{array}
$$

Figure 1: Abstract syntax of the language.

1.2 Operational Semantics

$$
\begin{array}{rccll}
\text{Environment} & \sigma & \in & \text{Env} & \\
\text{Location} & \ell & \in & \text{Loc} & \\
\text{Value} & v & \in & \text{Val} \triangleq \text{Env} + \text{Var} \times \text{Expr} \times \text{Env} & \\
\text{Weak Value} & w & \in & \text{WVal} \triangleq \text{Val} + \text{Loc} \times \text{Val} & \\
\text{Environment} & \sigma & \rightarrow & \bullet & \text{empty stack} \\
 & & \mid & (x, w) :: \sigma & \text{weak value binding} \\
 & & \mid & (x, \ell) :: \sigma & \text{free location binding} \\
\text{Value} & v & \rightarrow & \sigma & \text{exported environment} \\
 & & \mid & \langle \lambda x.e, \sigma \rangle & \text{closure} \\
\text{Weak Value} & w & \rightarrow & v & \text{value} \\
 & & \mid & \mu\ell.v & \text{recursive value}
\end{array}
$$

Figure 2: Definition of the semantic domains.

$$\boxed{(e, \sigma) \Downarrow v}$$

$$
\textsc{Id}\ \frac{\sigma(x) = v}{(x, \sigma) \Downarrow v}
\qquad
\textsc{RecId}\ \frac{\sigma(x) = \mu\ell.v}{(x, \sigma) \Downarrow v[\mu\ell.v/\ell]}
\qquad
\textsc{Fn}\ \frac{}{(\lambda x.e, \sigma) \Downarrow \langle \lambda x.e, \sigma \rangle}
\qquad
\textsc{App}\ \frac{(e_1, \sigma) \Downarrow \langle \lambda x.e, \sigma_1 \rangle \quad (e_2, \sigma) \Downarrow v_2 \quad (e, (x, v_2) :: \sigma_1) \Downarrow v}{(e_1\ e_2, \sigma) \Downarrow v}
$$

$$
\textsc{Link}\ \frac{(e_1, \sigma) \Downarrow \sigma_1 \quad (e_2, \sigma_1) \Downarrow v}{(e_1 \rtimes e_2, \sigma) \Downarrow v}
\qquad
\textsc{Empty}\ \frac{}{(\varepsilon, \sigma) \Downarrow \bullet}
\qquad
\textsc{Bind}\ \frac{\ell \notin \text{FLoc}(\sigma) \quad (e_1, (x, \ell) :: \sigma) \Downarrow v_1 \quad (e_2, (x, \mu\ell.v_1) :: \sigma) \Downarrow \sigma_2}{(x = e_1; e_2, \sigma) \Downarrow (x, \mu\ell.v_1) :: \sigma_2}
$$

Figure 3: The big-step operational semantics.

We use the locally nameless representation, and enforce that all values be *locally closed*. As a consequence, the big-step operational semantics will be *deterministic*, no matter what ℓ is chosen in the Bind rule.

1.3 Adding Memory

The first step towards abstraction is reformulating the semantics into a version with memory.

$$\boxed{e, \sigma, K \to e, \sigma, K}$$

$$\begin{array}{rcll}
e_1\; e_2, \sigma, K & \to & e_1, \sigma, K \circ (\_\; (e_2, \sigma)) & \\
e_1 \rtimes e_2, \sigma, K & \to & e_1, \sigma, K \circ (\_ \rtimes e_2) & \\
x = e_1; e_2, \sigma, K & \to & e_1, (x, \ell) :: \sigma, K \circ (x = \ell; (e_2, \sigma)) & \ell \notin \mathrm{FLoc}(\sigma)
\end{array}$$

$$\boxed{v, K \to e, \sigma, K}$$

$$\begin{array}{rcll}
\langle \lambda x.e, \sigma_1 \rangle, K \circ (\_\; (e_2, \sigma)) & \to & e_2, \sigma, K \circ (\langle \lambda x.e, \sigma_1 \rangle\; \_) \\
\sigma_1, K \circ (\_ \rtimes e_2) & \to & e_2, \sigma_1, K \\
v_1, K \circ (x = \ell; (e_2, \sigma)) & \to & e_2, (x, \mu\ell.v_1) :: \sigma, K \circ (x = \mu\ell.v_1; \_) \\
v_2, K \circ (\langle \lambda x.e, \sigma_1 \rangle\; \_) & \to & e, (x, v_2) :: \sigma_1, K
\end{array}$$

$$\boxed{v, K \to v, K}$$

$$\begin{array}{rcl}
\sigma_2, K \circ (x = w_1; \_) & \to & (x, w_1) :: \sigma_2, K
\end{array}$$

$$\boxed{e, \sigma, K \to v, K}$$

$$\begin{array}{rcll}
x, \sigma, K & \to & v, K & v = \sigma(x) \\
x, \sigma, K & \to & v[\mu\ell.v/\ell], K & \mu\ell.v = \sigma(x) \\
\lambda x.e, \sigma, K & \to & \langle \lambda x.e, \sigma \rangle, K & \\
\varepsilon, \sigma, K & \to & \bullet, K &
\end{array}$$

Figure 4: The equivalent small-step operational semantics.

$$\begin{array}{rclll}
\text{Environment} & \sigma & \in & \mathrm{Env} & \\
\text{Location} & \ell & \in & \mathrm{Loc} & \\
\text{Memory} & m & \in & \mathrm{Mem} \triangleq \mathrm{Loc} \xrightarrow{\text{fin}} \mathrm{Val} & \\
\text{Value} & v & \in & \mathrm{Val} \triangleq \mathrm{Env} + \mathrm{Var} \times \mathrm{Expr} \times \mathrm{Env} & \\
\text{Environment} & \sigma & \to & \bullet & \text{empty stack} \\
 & & \mid & (x, \ell) :: \sigma & \text{location binding} \\
\text{Value} & v & \to & \sigma & \text{exported environment} \\
 & & \mid & \langle \lambda x.e, \sigma \rangle & \text{closure}
\end{array}$$

Figure 5: Definition of the semantic domains with memory.

$$\boxed{e, \sigma, m, K \to e, \sigma, m, K}$$

$$\begin{array}{rcll}
e_1\; e_2, \sigma, m, K & \to & e_1, \sigma, m, K \circ (\_\; (e_2, \sigma)) & \\
e_1 \rtimes e_2, \sigma, m, K & \to & e_1, \sigma, m, K \circ (\_ \rtimes e_2) & \\
x = e_1; e_2, \sigma, m, K & \to & e_1, (x, \ell) :: \sigma, m, K \circ (x = \ell; (e_2, \sigma)) & \ell \notin \mathrm{dom}(m) \cup \mathrm{FLoc}(K)
\end{array}$$

$$\boxed{v, m, K \to e, \sigma, m, K}$$

$$\begin{array}{rcll}
\langle \lambda x.e, \sigma_1 \rangle, m, K \circ (\_\; (e_2, \sigma)) & \to & e_2, \sigma, m, K \circ (\langle \lambda x.e, \sigma_1 \rangle\; \_) & \\
\sigma_1, m, K \circ (\_ \rtimes e_2) & \to & e_2, \sigma_1, m, K & \\
v_1, m, K \circ (x = \ell; (e_2, \sigma)) & \to & e_2, (x, \ell) :: \sigma, m[\ell \mapsto v_1], K \circ (x = \ell; \_) & \\
v_2, m, K \circ (\langle \lambda x.e, \sigma_1 \rangle\; \_) & \to & e, (x, \ell) :: \sigma_1, m[\ell \mapsto v_2], K & \ell \notin \mathrm{dom}(m) \cup \mathrm{FLoc}(K)
\end{array}$$

$$\boxed{v, m, K \to v, m, K}$$

$$\begin{array}{rcl}
\sigma_2, m, K \circ (x = \ell; \_) & \to & (x, \ell) :: \sigma_2, m, K
\end{array}$$

$$\boxed{e, \sigma, m, K \to v, m, K}$$

$$\begin{array}{rcll}
x, \sigma, m, K & \to & v, m, K & \ell = \sigma(x), v = m(\ell) \\
\lambda x.e, \sigma, m, K & \to & \langle \lambda x.e, \sigma \rangle, m, K & \\
\varepsilon, \sigma, m, K & \to & \bullet, m, K &
\end{array}$$

Figure 6: The small-step operational semantics with memory.

1.4 Reconciling the Two Semantics

We need to prove that the two semantics simulate each other. Thus, we need to define a notion of equivalence between the two semantic domains.

Some useful lemmas.

$$\boxed{w \sim_f v, m}$$

$$\textsc{Eq-Nil}\ \frac{}{\bullet \sim_f \bullet} \qquad \textsc{Eq-ConsFree}\ \frac{\ell \notin \mathrm{dom}(f) \quad \ell \notin \mathrm{dom}(m) \quad \sigma \sim_f \sigma'}{(x,\ell) :: \sigma \sim_f (x,\ell) :: \sigma'} \qquad \textsc{Eq-ConsBound}\ \frac{f(\ell) = \ell' \quad \ell' \in \mathrm{dom}(m) \quad \sigma \sim_f \sigma'}{(x,\ell) :: \sigma \sim_f (x,\ell') :: \sigma'}$$

$$\textsc{Eq-ConsWVal}\ \frac{m(\ell') = v' \quad w \sim_f v' \quad \sigma \sim_f \sigma'}{(x,w) :: \sigma \sim_f (x,\ell') :: \sigma'} \qquad \textsc{Eq-Clos}\ \frac{\sigma \sim_f \sigma'}{\langle \lambda x.e, \sigma\rangle \sim_f \langle \lambda x.e, \sigma'\rangle} \qquad \textsc{Eq-Rec}\ \frac{m(\ell') = v' \quad v \sim_{f[\ell \mapsto \ell']} v'}{\mu\ell.v \sim_f v'}$$

Figure 7: The equivalence relation between weak values in the original semantics and values in the semantics with memory. $f \in \mathrm{Loc} \xrightarrow{\mathrm{fin}} \mathrm{Loc}$ tells what the free locations in w that were *opened* should be mapped to in memory. The memory component in the right-hand side is omitted.

Lemma 1.1 (Free locations not in f are free in memory).

$$w \sim_f v', m \Rightarrow m|_{\mathrm{FLoc}(w)-\mathrm{dom}(f)} = \bot$$

Lemma 1.2 (Equivalence is preserved by extension of memory).

$$w \sim_f v', m \text{ and } m \sqsubseteq m' \text{ and } m'|_{\mathrm{FLoc}(w)-\mathrm{dom}(f)} = \bot \Rightarrow w \sim_f v', m$$

Lemma 1.3 (Equivalence only cares about f on free locations).

$$w \sim_f v', m \text{ and } f|_{\mathrm{FLoc}(w)} = f|_{\mathrm{FLoc}(w)} \Rightarrow w \sim_{f'} v', m$$

Lemma 1.4 (Extending equivalence on free locations).

$$w \sim_f v', m \text{ and } \ell \notin \mathrm{dom}(f) \text{ and } \ell \notin \mathrm{dom}(m) \Rightarrow \forall u', w \sim_{f[\ell \mapsto \ell]} v', m[\ell \mapsto u']$$

Lemma 1.5 (Substitution of values).

$$w \sim_f v', m \text{ and } f(\ell) = \ell' \text{ and } m(\ell') = u' \text{ and } u \sim_{f-\ell} u', m \Rightarrow w[u/\ell] \sim_{f-\ell} v', m$$

Lemma 1.6 (Substitution of locations).

$$w \sim_f v', m \text{ and } \ell \in \mathrm{dom}(f) \text{ and } \nu \notin \mathrm{FLoc}(w) \Rightarrow w[\nu/\ell] \sim_{f\circ(\nu \leftrightarrow \ell)} v', m$$

It turns out, we only need to reason about what free locations are in the continuation of the semantics of Figure 6. Thus, we can formulate a big-step semantics as in Figure 8 and reason about the equivalence between the big-step semantics directly.

Then we can prove the following.

Definition 1.1 (Equivalence of configurations).

$$\sigma \sim \sigma', m, L \triangleq \sigma \sim_\bot \sigma', m \text{ and } \mathrm{FLoc}(\sigma) \subseteq L$$

Theorem 1.1 (Equivalence of semantics).

$$\sigma \sim \sigma', m, L \text{ and } (e,\sigma) \Downarrow v \Rightarrow \exists v', m', L' : v \sim v', m', L' \text{ and } (e, \sigma', m, L) \Downarrow (v', m', L')$$
$$\sigma \sim \sigma', m, L \text{ and } (e, \sigma', m, L) \Downarrow (v', m', L') \Rightarrow \exists v : v \sim v', m', L' \text{ and } (e,\sigma) \Downarrow v$$

2 Generating and Resolving Events

2.1 Without Memory

Now we formulate the semantics for generating events. The point of this semantics is that we are expressing a function that interacts with a stream of answers to external events as a data structure.

$$\boxed{(e, \sigma, m, L) \Downarrow (v, m, L)}$$

$$\textsc{Id}\ \frac{\ell = \sigma(x) \qquad v = m(\ell)}{(x, \sigma, m, L) \Downarrow (v, m, L)} \qquad \textsc{Fn}\ \frac{}{(\lambda x.e, \sigma, m, L) \Downarrow (\langle \lambda x.e, \sigma \rangle, m, L)}$$

$$\textsc{App}\ \frac{\begin{array}{c} (e_1, \sigma, m, L) \Downarrow (\langle \lambda x.e, \sigma_1 \rangle, m_1, L_1) \\ (e_2, \sigma, m_1, L_1) \Downarrow (v_2, m_2, L_2) \qquad \ell \notin \mathrm{dom}(m_2) \cup L_2 \qquad (e, (x, \ell) :: \sigma_1, m_2[\ell \mapsto v_2], L_2) \Downarrow (v, m', L') \end{array}}{(e_1\ e_2, \sigma, m, L) \Downarrow (v, m', L')}$$

$$\textsc{Link}\ \frac{(e_1, \sigma, m, L) \Downarrow (\sigma_1, m_1, L_1) \qquad (e_2, \sigma_1, m_1, L_1) \Downarrow (v, m', L')}{(e_1 \rtimes e_2, \sigma) \Downarrow (v, m', L')} \qquad \textsc{Empty}\ \frac{}{(\varepsilon, \sigma, m, L) \Downarrow (\bullet, m, L)}$$

$$\textsc{Bind}\ \frac{\ell \notin \mathrm{dom}(m) \cup L \qquad (e_1, (x, \ell) :: \sigma, m, L \cup \{\ell\}) \Downarrow (v_1, m_1, L_1) \qquad (e_2, (x, \ell) :: \sigma, m_1[\ell \mapsto v_1], L_1 - \{\ell\}) \Downarrow (\sigma_2, m', L')}{(x = e_1; e_2, \sigma, m, L) \Downarrow ((x, \ell) :: \sigma_2, m', L')}$$

Figure 8: The big-step operational semantics with memory.

$$\begin{array}{rcll}
\text{Environment} & \sigma & \rightarrow & \cdots \\
 & & \mid & [E] & \text{answer to an event} \\
\text{Value} & v & \rightarrow & \cdots \\
 & & \mid & E & \text{answer to an event} \\
\text{Event} & E & \rightarrow & \mathsf{Init} & \text{initial environment} \\
 & & \mid & \mathsf{Read}(E, x) & \text{read event} \\
 & & \mid & \mathsf{Call}(E, v) & \text{call event}
\end{array}$$

Figure 9: Definition of the semantic domains with events. All other semantic domains are equal to Figure 2.

We redefine how to read weak values given an environment.

$$\begin{array}{rclcrcll}
\bullet(x) & \triangleq & \bot & & ((x, w) :: \sigma)(x) & \triangleq & w & \\
((x, \ell) :: \sigma)(x) & \triangleq & \bot & & ((x', \_) :: \sigma)(x) & \triangleq & \sigma(x) & (x \neq x') \\
[E](x) & \triangleq & \mathsf{Read}(E, x) & & & & &
\end{array}$$

Then we need to add only one rule to the semantics in Figure 3 for the semantics to incorporate events.

$$\textsc{AppEvent}\ \frac{(e_1, \sigma) \Downarrow E \qquad (e_2, \sigma) \Downarrow v}{(e_1\ e_2, \sigma) \Downarrow \mathsf{Call}(E, v)}$$

Now we need to formulate the *event resolution* rules. The event resolution rule $w \downarrow_{\sigma_0} W$, given an answer σ_0 to the Init event, resolves all events within w to obtain W. Note that $\mathrm{FLoc}(\sigma_0) \cap \mathrm{FLoc}(w) = \emptyset$ is required for the resolution to work.

For the event resolution to be sound, we expect the following theorem to hold.

Theorem 2.1 (Soundness of resolution).

$$\sigma \downarrow_{\sigma_0} \Sigma \text{ and } (e, \sigma) \Downarrow v \text{ and } (e, \Sigma) \Downarrow V \Rightarrow v \downarrow_{\sigma_0} V$$

Now we present the definition for resolution.

Note that substitution of free locations that happens when unfolding a recursive value may generate multiple copies of the same event. In our simple module language, resolution is not impacted by neither the order nor the redundancy of resolution. However, in the future, to support stateful events which depend on both the

$$\boxed{w \downarrow_{\sigma_0} W}$$

$$\textsc{R-Init}\ \frac{}{\mathsf{Init} \downarrow \sigma_0} \qquad \textsc{R-Read}\ \frac{E \downarrow \Sigma \quad \Sigma(x) = V}{\mathsf{Read}(E, x) \downarrow V} \qquad \textsc{R-ReadRec}\ \frac{E \downarrow \Sigma \quad \Sigma(x) = \mu\ell.V}{\mathsf{Read}(E, x) \downarrow V[\mu\ell.V/\ell]} \qquad \textsc{R-CallV}\ \frac{E \downarrow \langle \lambda x.e, \Sigma \rangle \quad v \downarrow V \quad (e, (x, V) :: \Sigma) \Downarrow V'}{\mathsf{Call}(E, v) \downarrow V'}$$

$$\textsc{R-CallE}\ \frac{E \downarrow E' \quad v \downarrow V}{\mathsf{Call}(E, v) \downarrow \mathsf{Call}(E', V)} \qquad \textsc{R-Nil}\ \frac{}{\bullet \downarrow \bullet} \qquad \textsc{R-ConsLoc}\ \frac{\sigma \downarrow \Sigma}{(x, \ell) :: \sigma \downarrow (x, \ell) :: \Sigma} \qquad \textsc{R-ConsWVal}\ \frac{w \downarrow W \quad \sigma \downarrow \Sigma}{(x, w) :: \sigma \downarrow (x, W) :: \Sigma}$$

$$\textsc{R-HoleE}\ \frac{E \downarrow E'}{[E] \downarrow [E']} \qquad \textsc{R-HoleEnv}\ \frac{E \downarrow \Sigma}{[E] \downarrow \Sigma} \qquad \textsc{R-Clos}\ \frac{\sigma \downarrow \Sigma}{\langle \lambda x.e, \sigma \rangle \downarrow \langle \lambda x.e, \Sigma \rangle} \qquad \textsc{R-Rec}\ \frac{v \downarrow V}{\mu\ell.v \downarrow \mu\ell.V}$$

Figure 10: The resolution relation. The subscript σ_0 in $\downarrow_{\sigma_0}$ is omitted for brevity.

order and redundancy, we must assign *unique numbers* to the events. Furthermore, for abstraction of the data structure, it is much easier if everything is threaded through the memory through memory locations. This motivates us to:

1. Lift $\downarrow$ in the case for memory in a way that is *compatible* with the equivalence relation $\sim$.
2. Formulate a semantics where events are assigned a unique number, and their dependencies are tracked by a *memory for events*.

We expect by doing so, we may obtain a sound approximation for the semantics by simply abstracting the *evaluation relation* and the *resolution relation*.

2.2 With Memory

2.3 With Event Memory