Modular Analysis

Joonhyup Lee

April 12, 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

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

Figure 2: Definition of the semantic domains.

$\boxed{\sigma \vdash e \Downarrow v}$

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

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

Figure 3: The big-step operational semantics.

The big-step operational semantics is *deterministic* up to α-equivalence.

$$
\begin{array}{rccll}
\text{Environment} & \sigma & \in & \text{MEnv} \triangleq \text{Var} \xrightarrow{\text{fin}} \text{Loc} & \\
\text{Memory} & m & \in & \text{Mem} \triangleq \text{Loc} \xrightarrow{\text{fin}} \text{MVal} & \\
\text{Allocated set} & L & \subseteq & \text{Loc} & \\
\text{Value} & v & \in & \text{MVal} \triangleq \text{MEnv} + \text{Var} \times \text{Expr} \times \text{MEnv} & \\
\text{Environment} & \sigma & \rightarrow & \bullet & \text{empty stack} \\
 & & \mid & (x, \ell) :: \sigma & \text{location binding} \\
\text{Value} & v & \rightarrow & \sigma & \text{exported environment} \\
 & & \mid & \langle \lambda x.e, \sigma \rangle & \text{closure}
\end{array}
$$

Figure 4: Definition of the semantic domains with memory.

$$\boxed{\sigma, m, L \vdash e \Downarrow v, m', L'}$$

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

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

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

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

Figure 5: The big-step operational semantics with memory.

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

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

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

Figure 6: The equivalence relation between weak values in the original semantics and values in the semantics with memory. $f \in \text{Loc} \xrightarrow{\text{fin}} \text{Loc}$ tells what the free locations in w that were *opened* should be mapped to in memory. m is omitted for brevity.

1.3 Reconciling with Conventional Backpatching

The semantics in Figure 3 makes sense due to similarity with a conventional backpatching semantics as presented in Figure 5. We have defined a relation $\sim$ that satisfies:

$$\sim \subseteq \text{WVal} \times (\text{MVal} \times \text{Mem} \times \mathcal{P}(\text{Loc})) \qquad \bullet \sim (\bullet, \varnothing, \varnothing)$$

The following theorem holds:

Theorem 1.1 (Equivalence of semantics)**.** For all $\sigma \in \mathrm{Env}, \sigma' \in \mathrm{MEnv} \times \mathrm{Mem} \times \mathcal{P}(\mathrm{Loc}), v \in \mathrm{Val}, v' \in \mathrm{MVal} \times \mathrm{Mem} \times \mathcal{P}(\mathrm{Loc})$, we have:

$$\sigma \sim \sigma' \text{ and } \sigma \vdash e \Downarrow v \Rightarrow \exists v' : v \sim v' \text{ and } \sigma' \vdash e \Downarrow v'$$
$$\sigma \sim \sigma' \text{ and } \sigma' \vdash e \Downarrow v' \Rightarrow \exists v : v \sim v' \text{ and } \sigma \vdash e \Downarrow v$$

The definition for $w \sim (\sigma, m, L)$ is:

$$w \sim_{\perp} (\sigma, m) \text{ and } \mathrm{FLoc}(w) \subseteq L$$

where the definition for $\sim_f$ is given in Figure 6.

The proof of Theorem 1.1 uses some useful lemmas, such as:

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

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

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

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

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)**.**

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

2 Generating and Resolving Events

Now we formulate the semantics for generating events.

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

Figure 7: Definition of the semantic domains with events. All other semantic domains are equal to Figure 2.

We extend how to read weak values given an environment.

$$\bullet(x) \triangleq \bot \qquad ((x', \ell) :: \sigma)(x) \triangleq (x = x' ? \ell : \sigma(x))$$

$$[E](x) \triangleq \mathsf{Read}(E, x) \qquad ((x', w) :: \sigma)(x) \triangleq (x = x' ? w : \sigma(x))$$

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

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

Now we need to formulate the *concrete linking* rules. The concrete linking rule $\sigma_0 \infty w$, given an answer σ_0 to the Init event, resolves all events within w to obtain a set of final results.

$$\boxed{\infty \in \text{Env} \to \text{Event} \to \mathcal{P}(\text{Val})}$$

$$\begin{aligned} \sigma_0 \infty \mathsf{Init} &\triangleq \{\sigma_0\} \\ \sigma_0 \infty \mathsf{Read}(E, x) &\triangleq \{v_+ | \sigma_+ \in \sigma_0 \infty E \wedge \sigma_+(x) = v_+\} \\ &\cup \{v_+[\mu\ell.v_+/\ell] | \sigma_+ \in \sigma_0 \infty E \wedge \sigma_+(x) = \mu\ell.v_+\} \\ \sigma_0 \infty \mathsf{Call}(E, v) &\triangleq \{v'_+ | \langle \lambda x.e, \sigma_+ \rangle \in \sigma_0 \infty E \wedge v_+ \in \sigma_0 \infty v \wedge (x, v_+) :: \sigma_+ \vdash e \Downarrow v'_+\} \\ &\cup \{\mathsf{Call}(E_+, v_+) | E_+ \in \sigma_0 \infty E \wedge v_+ \in \sigma_0 \infty v\} \end{aligned}$$

$$\boxed{\infty \in \text{Env} \to \text{Env} \to \mathcal{P}(\text{Env})}$$

$$\begin{aligned} \sigma_0 \infty \bullet &\triangleq \{\bullet\} \\ \sigma_0 \infty (x, \ell) :: \sigma &\triangleq \{(x, \ell) :: \sigma_+ | \sigma_+ \in \sigma_0 \infty \sigma\} \\ \sigma_0 \infty (x, w) :: \sigma &\triangleq \{(x, w_+) :: \sigma_+ | w_+ \in \sigma_0 \infty w \wedge \sigma_+ \in \sigma_0 \infty \sigma\} \\ \sigma_0 \infty [E] &\triangleq \{\sigma_+ | \sigma_+ \in \sigma_0 \infty E\} \cup \{[E_+] | E_+ \in \sigma_0 \infty E\} \end{aligned}$$

$$\boxed{\infty \in \text{Env} \to \text{Val} \to \mathcal{P}(\text{Val})}$$

$$\sigma_0 \infty \langle \lambda x.e, \sigma \rangle \triangleq \{\langle \lambda x.e, \sigma_+ \rangle | \sigma_+ \in \sigma_0 \infty \sigma\}$$

$$\boxed{\infty \in \text{Env} \to \text{WVal} \to \mathcal{P}(\text{WVal})}$$

$$\sigma_0 \infty \mu\ell.v \triangleq \{\mu\ell'.v_+ | \ell' \notin \text{FLoc}(v) \cup \text{FLoc}(\sigma_0) \wedge v_+ \in \sigma_0 \infty v[\ell'/\ell]\}$$

Concrete linking makes sense because of the following theorem. First define:

$$\text{eval}(e, \sigma) \triangleq \{v | \sigma \vdash e \Downarrow v\} \qquad \text{eval}(e, \Sigma) \triangleq \bigcup_{\sigma \in \Sigma} \text{eval}(e, \sigma) \qquad \Sigma_0 \infty W \triangleq \bigcup_{\substack{\sigma_0 \in \Sigma_0 \\ w \in W}} (\sigma_0 \infty w)$$

Then the following holds:

Theorem 2.1 (Advance)**.** Given $e \in \text{Expr}, \Sigma_0, \Sigma \subseteq \text{Env}$,

$$\text{eval}(e, \Sigma_0 \infty \Sigma) \subseteq \Sigma_0 \infty \text{eval}(e, \Sigma)$$

The proof of Theorem 2.1 uses some useful lemmas, such as:

Lemma 2.1 (Linking distributes under substitution)**.** Let σ_0 be the external environment that is linked with weak values w and u. For all $\ell \notin \mathrm{FLoc}(\sigma_0)$, we have:

$$\forall w_+, u_+ : w_+ \in \sigma_0 \propto w \wedge u_+ \in \sigma_0 \propto u \Rightarrow w_+[u_+/\ell] \in \sigma_0 \propto w[u/\ell]$$

Lemma 2.2 (Linking is compatible with reads)**.** Let σ_0 be the external environment that is linked with some environment σ. Let w be the value obtained from reading x from σ. Let unfold : WVal $\to$ Val be defined as:

$$\mathrm{unfold}(\mu\ell.v) \triangleq v[\mu\ell.v/\ell] \qquad \mathrm{unfold}(v) \triangleq v$$

Then for all $\sigma_+ \in \sigma_0 \propto \sigma$, we have:

$$\exists w_+ \in \mathrm{WVal} : \sigma_+(x) = w_+ \wedge \mathrm{unfold}(w_+) \in \sigma_0 \propto \mathrm{unfold}(w)$$

Now we can formulate modular analysis. A modular analysis consists of two requirements: an abstraction for the semantics with events and an abstraction for the semantic linking operator.

Theorem 2.2 (Modular analysis)**.** Assume:

1. An abstract domain $\mathrm{WVal}^\#$ that is concretized by a monotonic $\gamma \in \mathcal{P}(\mathrm{WVal}) \to \mathrm{WVal}^\#$
2. A sound $\mathrm{eval}^\#$: $\Sigma_0 \subseteq \gamma(\sigma_0^\#) \Rightarrow \mathrm{eval}(e, \Sigma_0) \subseteq \gamma(\mathrm{eval}^\#(e, \sigma_0^\#))$
3. A sound $\propto^\#$: $\Sigma_0 \subseteq \gamma(\sigma_0^\#)$ and $W \subseteq \gamma(w^\#) \Rightarrow \Sigma_0 \propto W \subseteq \gamma(\sigma_0^\# \propto^\# w^\#)$

then we have:

$$\Sigma_0 \subseteq \gamma(\sigma_0^\#) \text{ and } \Sigma \subseteq \gamma(\sigma^\#) \Rightarrow \mathrm{eval}(e, \Sigma_0 \propto \Sigma) \subseteq \gamma(\sigma_0^\# \propto^\# \mathrm{eval}^\#(e, \sigma^\#))$$

Corollary 2.1 (Modular analysis of linked program)**.**

$$\Sigma_0 \subseteq \gamma(\sigma_0^\#) \text{ and } [\mathsf{Init}] \in \gamma(\mathsf{Init}^\#) \Rightarrow \mathrm{eval}(e_1 \rtimes e_2, \Sigma_0) \subseteq \gamma(\mathrm{eval}^\#(e_1, \sigma_0^\#) \propto^\# \mathrm{eval}^\#(e_2, \mathsf{Init}^\#))$$

3 CFA

3.1 Collecting semantics

$$\begin{array}{rccl}
\text{Program point} & p & \in & \mathbb{P} \triangleq \{\text{finite set of program points}\} \\
\text{Labelled expression} & pe & \in & \mathbb{P} \times \text{Expr} \\
\text{Labelled location} & \ell^p & \in & \mathbb{P} \times \text{Loc} \\
\text{Collecting semantics} & t & \in & \mathbb{T} \triangleq \mathbb{P} \to \mathcal{P}(\text{Env} + \text{Env} \times \text{Val}) \\
\text{Labelled expression} & pe & \to & \{p : e\} \\
\text{Expression} & e & \to & x \mid \lambda x.pe \mid pe\ pe \mid pe \rtimes pe \mid \varepsilon \mid x = pe; pe
\end{array}$$

$\boxed{\text{Step} : \mathbb{T} \to \mathbb{T}}$

$$\text{Step}(t) \triangleq \bigcup_{p \in \mathbb{P}} \text{step}(t, p)$$

$\boxed{\text{step} : (\mathbb{T} \times \mathbb{P}) \to \mathbb{T}}$

$$\begin{aligned}
\text{step}(t,p) \triangleq\ & [p \mapsto \{(\sigma, v) | \sigma \in t(p) \text{ and } \sigma(x) = v\}] && \text{when } \{p : x\} \\
& \cup [p \mapsto \{(\sigma, v[\mu\ell^{p'}.v/\ell^{p'}]) | \sigma \in t(p) \text{ and } \sigma(x) = \mu\ell^{p'}.v\}] \\
\text{step}(t,p) \triangleq\ & [p \mapsto \{(\sigma, \langle \lambda x.p', \sigma\rangle) | \sigma \in t(p)\}] && \text{when } \{p : \lambda x.p'\} \\
\text{step}(t,p) \triangleq\ & [p_1 \mapsto \{\sigma \in \text{Env} | \sigma \in t(p)\}] && \text{when } \{p : p_1\ p_2\} \\
& \cup [p_2 \mapsto \{\sigma \in \text{Env} | \sigma \in t(p)\}] \\
& \cup \bigcup_{\sigma \in t(p)} \bigcup_{(\sigma, \langle \lambda x.p', \sigma_1\rangle) \in t(p_1)} [p' \mapsto \{(x, v_2) :: \sigma_1 | (\sigma, v_2) \in t(p_2)\}] \\
& \cup [p \mapsto \bigcup_{\sigma \in t(p)} \bigcup_{(\sigma, \langle \lambda x.p', \sigma_1\rangle) \in t(p_1)} \bigcup_{(\sigma, v_2) \in t(p_2)} \{(\sigma, v) | ((x, v_2) :: \sigma_1, v) \in t(p')\}] \\
& \cup [p \mapsto \bigcup_{\sigma \in t(p)} \{(\sigma, \mathsf{Call}(E_1, v_2)) | (\sigma, E_1) \in t(p_1) \text{ and } (\sigma, v_2) \in t(p_2)\}] \\
\text{step}(t,p) \triangleq\ & [p_1 \mapsto \{\sigma | \sigma \in t(p)\}] && \text{when } \{p : p_1 \rtimes p_2\} \\
& \cup [p_2 \mapsto \bigcup_{\sigma \in t(p)} \{\sigma_1 | (\sigma, \sigma_1) \in t(p_1)\}] \\
& \cup [p \mapsto \bigcup_{\sigma \in t(p)} \bigcup_{(\sigma, \sigma_1) \in t(p_1)} \{(\sigma, v_2) | (\sigma_1, v_2) \in t(p_2)\}] \\
\text{step}(t,p) \triangleq\ & [p \mapsto \{(\sigma, \bullet) | \sigma \in t(p)\}] && \text{when } \{p : \varepsilon\} \\
\text{step}(t,p) \triangleq\ & [p_1 \mapsto \bigcup_{\sigma \in t(p)} \{(x, \ell^{p_1}) :: \sigma | \ell \not\in \text{FLoc}(\sigma)\}] && \text{when } \{p : x = p_1; p_2\} \\
& \cup [p_2 \mapsto \bigcup_{\sigma \in t(p)} \{(x, \mu\ell^{p_1}.v_1) :: \sigma | ((x, \ell^{p_1}) :: \sigma, v_1) \in t(p_1)\}] \\
& \cup [p \mapsto \bigcup_{\sigma \in t(p)} \bigcup_{((x, \ell^{p_1}) :: \sigma, v_1) \in t(p_1)} \{(\sigma, (x, \mu\ell^{p_1}.v_1) :: \sigma_2) | ((x, \mu\ell^{p_1}.v_1) :: \sigma, \sigma_2) \in t(p_2)\}]
\end{aligned}$$

The collecting semantics $[\![p_0]\!]\Sigma_0$ computed by

$$[\![p_0]\!]\Sigma_0 \triangleq \text{lfp}(\lambda t.\text{Step}(t) \cup t_{\text{init}}) \quad \text{where } t_{\text{init}} = [p_0 \mapsto \Sigma_0]$$

contains all derivations of the form $\sigma_0 \vdash p_0 \Downarrow v_0$ for some $\sigma_0 \in \Sigma_0$ and v_0. That is, (σ, v) is contained in $[\![p_0]\!]\Sigma_0(p)$ if and only if $\sigma \vdash p \Downarrow v$ is contained in some derivation for the judgment $\sigma_0 \vdash p_0 \Downarrow v_0$.

3.2 Abstract semantics

$$\begin{array}{rccl}
\text{Abstract event} & E^\# & \in & \text{Event}^\# \\
\text{Abstract environment} & \sigma^\# & \in & \text{Env}^\# \triangleq (\text{Var} \xrightarrow{\text{fin}} \mathcal{P}(\mathbb{P})) \times \mathcal{P}(\text{Event}^\#) \\
\text{Abstract closure} & \langle \lambda x.p, p'\rangle & \in & \text{Clos}^\# \triangleq \text{Var} \times \mathbb{P} \times \mathbb{P} \\
\text{Abstract value} & v^\# & \in & \text{Val}^\# \triangleq \text{Env}^\# \times \mathcal{P}(\text{Clos}^\#) \\
\text{Abstract semantics} & t^\# & \in & \mathbb{T}^\# \triangleq \mathbb{P} \to \text{Env}^\# \times \text{Val}^\# \\
\text{Abstract event} & E^\# & \to & \mathsf{Init}^\# \mid \mathsf{Read}^\#(p, x) \mid \mathsf{Call}^\#(p, p)
\end{array}$$

$\boxed{\sigma \leq (\sigma^\#, t^\#)}$

$$\textsc{Conc-Nil}\ \frac{}{\bullet \leq \sigma^\#} \qquad \textsc{Conc-ENil}\ \frac{E \leq (\sigma^\#, \varnothing)}{[E] \leq \sigma^\#} \qquad \textsc{Conc-ConsLoc}\ \frac{p \in \sigma^\#.1(x) \quad \sigma \leq \sigma^\#}{(x, \ell^p) :: \sigma \leq \sigma^\#} \qquad \textsc{Conc-ConsWVal}\ \frac{p \in \sigma^\#.1(x) \quad w \leq t^\#(p).2 \quad \sigma \leq \sigma^\#}{(x, w) :: \sigma \leq \sigma^\#}$$

$\boxed{w \leq (v^\#, t^\#)}$

$$\textsc{Conc-Clos}\ \frac{\langle \lambda x.p, p' \rangle \in v^\#.2 \quad \sigma \leq t^\#(p').1}{\langle \lambda x.p, \sigma \rangle \leq v^\#} \qquad \textsc{Conc-Rec}\ \frac{v \leq t^\#(p).2 \quad v \leq v^\#}{\mu \ell^p.v \leq v^\#}$$

$$\textsc{Conc-Init}\ \frac{\mathsf{Init}^\# \in v^\#.1.2}{\mathsf{Init} \leq v^\#} \qquad \textsc{Conc-Read}\ \frac{\mathsf{Read}^\#(p, x) \in v^\#.1.2 \quad [E] \leq t^\#(p).1}{\mathsf{Read}(E, x) \leq v^\#} \qquad \textsc{Conc-Call}\ \frac{\mathsf{Call}^\#(p_1, p_2) \in v^\#.1.2 \quad E \leq t^\#(p_1).2 \quad v \leq t^\#(p_2).2}{\mathsf{Call}(E, v) \leq v^\#}$$

Figure 8: The concretization relation between weak values and abstract values. $t^\#$ is omitted.

The concretization function γ that sends an element of $\mathbb{T}^\#$ to $\mathbb{T}$ is defined as:

$$\gamma(t^\#) \triangleq \lambda p.\{\sigma | \sigma \leq (t^\#(p).1, t^\#)\} \cup \{(\sigma, v) | v \leq (t^\#(p).2, t^\#)\}$$

where $\leq$ is the concretization relation that is inductively defined in Figure 8.

Now the abstract semantic function can be given.

$\boxed{\mathrm{Step}^\# : \mathbb{T}^\# \to \mathbb{T}^\#}$

$$\mathrm{Step}^\#(t^\#) \triangleq \bigsqcup_{p \in \mathbb{P}} \mathrm{step}^\#(t^\#, p)$$

$\boxed{\mathrm{step}^\# : (\mathbb{T}^\# \times \mathbb{P}) \to \mathbb{T}^\#}$

$$\begin{aligned}
\mathrm{step}^\#(t^\#, p) &\triangleq [p \mapsto \bigsqcup_{p' \in t^\#(p).1.1(x)} (\bot, t^\#(p').2)] && \text{when } \{p : x\} \\
&\sqcup [p \mapsto (\bot, (([], \{\mathsf{Read}^\#(p, x)\}), \varnothing))] && \text{if } t^\#(p).1.2 \neq \varnothing \\
\mathrm{step}^\#(t^\#, p) &\triangleq [p \mapsto (\bot, (\bot, \{\langle \lambda x.p', p \rangle\}))] && \text{when } \{p : \lambda x.p'\} \\
\mathrm{step}^\#(t^\#, p) &\triangleq [p_1 \mapsto (t^\#(p).1, \bot)] && \text{when } \{p : p_1\ p_2\} \\
&\sqcup [p_2 \mapsto (t^\#(p).1, \bot)] \\
&\sqcup \bigsqcup_{\langle \lambda x.p', p'' \rangle \in t^\#(p_1).2.2} [p' \mapsto (t^\#(p'').1 \sqcup ([x \mapsto \{p_2\}], \varnothing), \bot)] \\
&\sqcup [p \mapsto \bigsqcup_{\langle \lambda x.p', _ \rangle \in t^\#(p_1).2.2} (\bot, t^\#(p').2)] \\
&\sqcup [p \mapsto (\bot, (([], \{\mathsf{Call}^\#(p_1, p_2)\}), \varnothing))] && \text{if } t^\#(p_1).2.1.2 \neq \varnothing \\
\mathrm{step}^\#(t^\#, p) &\triangleq [p_1 \mapsto (t^\#(p).1, \bot)] && \text{when } \{p : p_1 \rtimes p_2\} \\
&\sqcup [p_2 \mapsto (t^\#(p_1).2.1, \bot)] \\
&\sqcup [p \mapsto (\bot, t^\#(p_2).2)] \\
\mathrm{step}^\#(t^\#, p) &\triangleq \bot && \text{when } \{p : \varepsilon\} \\
\mathrm{step}^\#(t^\#, p) &\triangleq [p_1 \mapsto (t^\#(p).1 \sqcup ([x \mapsto \{p_1\}], \varnothing), \bot)] && \text{when } \{p : x = p_1; p_2\} \\
&\sqcup [p_2 \mapsto (t^\#(p).1 \sqcup ([x \mapsto \{p_1\}], \varnothing), \bot)] \\
&\sqcup [p \mapsto (\bot, (t^\#(p_2).2.1 \sqcup ([x \mapsto \{p_1\}], \varnothing), \varnothing))]
\end{aligned}$$

The abstract semantics $t^\#$ computed by

$$[\![p_0]\!]^\#(\sigma_0^\#, t_0^\#) \triangleq \mathrm{lfp}(\lambda t^\#.\mathrm{Step}^\#(t^\#) \sqcup t_{\mathrm{init}}^\#) \quad \text{where } t_{\mathrm{init}} = t_0^\# \sqcup [p_0 \mapsto (\sigma_0^\#, \bot)]$$

is a sound abstraction of $[\![p_0]\!]\Sigma_0$ when $\Sigma_0 \subseteq \gamma(\sigma_0^\#, t_0^\#)$.

3.3 Abstract linking

Now we define a sound linking operator that abstracts $\backpropto$. Assume we have

$$\sigma_0 \leq (\sigma_0^\#, t_0^\#) \quad t \subseteq \gamma(t^\#)$$

we define:

$$\sigma_0 \backpropto t \triangleq \lambda p. \bigcup_{\sigma \in t(p)} (\sigma_0 \backpropto \sigma) \cup \bigcup_{(\sigma, v) \in t(p)} \{(\sigma_+, v_+) | \sigma_+ \in \sigma_0 \backpropto \sigma \text{ and } v_+ \in \sigma_0 \backpropto v\}$$

We want to define $\backpropto^\#$ so that the following holds:

$$\sigma_0 \backpropto t \subseteq \gamma((\sigma_0^\#, t_0^\#) \backpropto^\# t^\#)$$

This is equivalent to saying that the linked result $t_+^\# = (\sigma_0^\#, t_0^\#) \backpropto^\# t^\#$ satisfies:

$$\sigma_0 \leq (\sigma_0^\#, t_0^\#) \text{ and } w \leq (v^\#, t^\#) \Rightarrow w_+ \leq (v_+^\#, t_+^\#)$$

for each $w_+ \in \sigma_0 \backpropto w$ and $p \in \mathbb{P}$, where $[v^\#, v_+^\#] \in \{[(t^\#(p).1, \varnothing), (t_+^\#(p).1, \varnothing)], [t^\#(p).2, t_+^\#(p).2]\}$.

The condition for $t_+^\#$ can be deduced by attempting the proof of the above in advance.

We proceed by induction on the derivation for

$$w_+ \in \sigma_0 \backpropto w$$

and inversion on $w \leq (v^\#, t^\#)$.

When:	$w = \mathsf{Init}$,	
Have:	$\mathsf{Init}^\# \in v^\#.1.2$	
Need:	$v_+^\# \sqsupseteq \sigma_0^\#$	
	$t_+^\# \sqsupseteq t_0^\#$	
When:	$w = \mathsf{Read}(E, x)$,	
Have:	$\mathsf{Read}^\#(p', x) \in v^\#.1.2$ and $[E] \leq t^\#(p').1$	
Need:	$v_+^\# \sqsupseteq t_+^\#(p'').2$	for $p'' \in t_+^\#(p').1.1(x)$
	$v_+^\# \sqsupseteq (([], \{\mathsf{Read}^\#(p', x)\}), \varnothing)$	if $t_+^\#(p').1.2 \neq \varnothing$
When:	$w = \mathsf{Call}(E, v)$,	
Have:	$\mathsf{Call}^\#(p_1, p_2) \in v^\#.1.2$ and $E \leq t^\#(p_1).2$ and $v \leq t^\#(p_2).2$	
Need:	$v_+^\# \sqsupseteq t_+^\#(p').2$	for $\langle \lambda x.p', p'' \rangle \in t_+^\#(p_1).2.2$
	$v_+^\# \sqsupseteq (([], \{\mathsf{Call}^\#(p_1, p_2)\}), \varnothing)$	if $t_+^\#(p_1).2.1.2 \neq \varnothing$
	$t_+^\#(p') \sqsupseteq (t_+^\#(p'').1 \sqcup ([x \mapsto \{p_2\}], \varnothing), \varnothing)$	for $\langle \lambda x.p', p'' \rangle \in t_+^\#(p_1).2.2$
	$t_+^\# \sqsupseteq \mathrm{Step}^\#(t_+^\#)$	
When:	$w = (x, \ell^{p'}) :: \sigma$,	
Have:	$p' \in v^\#.1.1(x)$ and $\sigma \leq v^\#$	
Need:	$v_+^\#.1.1(x) \ni p'$	
When:	$w = (x, w') :: \sigma$,	
Have:	$p' \in v^\#.1.1(x)$ and $w' \in t^\#(p').1$ and $\sigma \leq v^\#$	
Need:	$v_+^\#.1.1(x) \ni p'$	
When:	$w = \langle \lambda x.p', \sigma \rangle$,	
Have:	$\langle \lambda x.p', p'' \rangle \in v^\#.2$ and $\sigma \leq t^\#(p'').1$	
Need:	$v_+^\#.2 \ni \langle \lambda x.p', p'' \rangle$	

The above conditions can be summarized by saying $t_+^\#$ is a post-fixed point of:

$$\lambda t_+^\#.\mathrm{Step}^\#(t_+^\#) \sqcup \mathrm{Link}^\#(\sigma_0^\#, t^\#, t_+^\#) \sqcup t_0^\#$$

where $\mathrm{Link}^\#(\sigma_0^\#, t^\#, t_+^\#)$ is the least function that satisfies:

Let $\text{link}^\# = \text{Link}^\#(\sigma_0^\#, t^\#, t_+^\#)$ in		
	For each $p \in \mathbb{P}$, when $v^\#, v_+^\# = (t^\#(p).1, \varnothing), (\text{link}^\#(p).1, \varnothing)$	
	or when $v^\#, v_+^\# = t^\#(p).2, \text{link}^\#.2$	
If:	$\mathsf{Init}^\# \in v^\#.1.2$	
Then:	$v_+^\# \sqsupseteq \sigma_0^\#$	
If:	$\mathsf{Read}^\#(p', x) \in v^\#.1.2$	
Then:	$v_+^\# \sqsupseteq t_+^\#(p'').2$	for $p'' \in t_+^\#(p').1.1(x)$
	$v_+^\# \sqsupseteq (([], \{\mathsf{Read}^\#(p', x)\}), \varnothing)$	if $t_+^\#(p').1.2 \neq \varnothing$
If:	$\mathsf{Call}^\#(p_1, p_2) \in v^\#.1.2$	
Then:	$v_+^\# \sqsupseteq t_+^\#(p').2$	for $\langle \lambda x.p', p'' \rangle \in t_+^\#(p_1).2.2$
	$v_+^\# \sqsupseteq (([], \{\mathsf{Call}^\#(p_1, p_2)\}), \varnothing)$	if $t_+^\#(p_1).2.1.2 \neq \varnothing$
	$\text{link}^\#(p') \sqsupseteq (t_+^\#(p'').1 \sqcup ([x \mapsto \{p_2\}], \varnothing), \varnothing)$	for $\langle \lambda x.p', p'' \rangle \in t_+^\#(p_1).2.2$
If:	$p' \in v^\#.1.1(x)$	
Then:	$v_+^\#.1.1(x) \ni p'$	
If:	$p' \in v^\#.1.1(x)$	
Then:	$v_+^\#.1.1(x) \ni p'$	
If:	$\langle \lambda x.p', p'' \rangle \in v^\#.2$	
Then:	$v_+^\#.2 \ni \langle \lambda x.p', p'' \rangle$	

Note that the left-hand side contains only $\text{link}^\#$ and the right-hand side does not depend on the value of $\text{link}^\#$.

Some auxiliary lemmas:

Lemma 3.1 (Substitution of values)**.**

$$w \leq (v^\#, t^\#) \text{ and } u \leq (t^\#(p).2, t^\#) \Rightarrow w[u/\ell^p] \leq (v^\#, t^\#)$$

Lemma 3.2 (Sound $\text{step}^\#$)**.**

$$\forall p, t, t^\# : t \subseteq \gamma(t^\#) \Rightarrow \text{step}(t, p) \cup t \subseteq \gamma(\text{step}^\#(t^\#, p) \sqcup t^\#)$$

Lemma 3.3 (Sound $\text{Step}^\#$)**.**

$$\forall t_{\text{init}}, t^\# : t_{\text{init}} \subseteq \gamma(t^\#) \text{ and } \text{Step}^\#(t^\#) \sqsubseteq t^\# \Rightarrow \text{lfp}(\lambda t.\text{Step}(t) \cup t_{\text{init}}) \subseteq \gamma(t^\#)$$