Modular Analysis

Joonhyup Lee

April 29, 2024

1 Introduction

"There's a library for everything" is one way of summarizing modern software development. A library is a reusable component of software that is packaged into a module, which can be imported and used by others that require similar features. Since libraries promote code reuse and modular code design, both recommended practices, the number of libraries continues to grow.

Libraries, however, mean trouble for static program analyzers. Using values from libraries that cannot be resolved from analyzing the local code prevent the analyzer from creating accurate summaries of the program behavior. Realistic program analyzers resort to making either angelic assumptions about unknown functions (such as Infer), which may lead to false negatives, or demonic assumptions, which greatly drops the precision of the analysis.

Thus, a method of analysis that allows uninterpreted function symbols is needed. Even in the presence of unresolved variables, a program analyzer should be able to produce a result that can be reused when the names are resolved *separately.*

We present such a framework powered by *abstract interpretation.* What abstract interpretation teaches us is that given a concrete semantics of a program and semantic operators that act on the semantic domains, approximating each with a sound abstract version leads to a sound analysis. Thus we present a semantics of programs that works even when the initial environment the program is evaluated under is not fully given. We call this the "shadow semantics", since it computes a shadow of the expected program execution. Also, we present a semantic linking operator that can fill in the missing parts later.

Example 1: Error handling	Example 2: React-like state management
`let _ =`	`let make = fun cur ->`
`  match f 42 with`	`  let x, setX =`
`  | Ok result -> result`	`    useState { cur with val = 0 }`
`  | Err trace -> trace - 1`	`  in setX (x + 1)`
Shadow 1: value of the `let` binding	**Shadow 2: body of the `make` function**
$\mathsf{Match}(\mathsf{ret}, \texttt{Ok})$ or	$\mathsf{Call}(\mathsf{Get}(\mathsf{ret}, 1), \mathsf{Add}(\mathsf{Get}(\mathsf{ret}, 0), 1))$
$\mathsf{Sub}(\mathsf{Match}(\mathsf{ret}, \texttt{Err}), 1)$	where $\mathsf{ret} = \mathsf{Call}(\mathsf{useState}, (\texttt{val}, 0) :: [\mathsf{cur}])$
where $\mathsf{ret} = \mathsf{Call}(\mathsf{f}, 42)$	and $\mathsf{useState} = \mathsf{Read}(\mathsf{Init}, \texttt{useState})$
and $\mathsf{f} = \mathsf{Read}(\mathsf{Init}, \texttt{f})$	and $\mathsf{cur} = \mathsf{Read}(\mathsf{Init}, \texttt{cur})$

As a preview, see the examples. The first example calls an unknown function `f` with argument `42`, and performs different actions depending on the result of the function call. There are two shadows for this program, one from the `Ok` branch and another from the `Err` branch. When the code for the `f` function is given in the initial environment, the shadow can be colored in to a specific branch. For example, when

$$\sigma_0 = (\texttt{f}, \langle \lambda\texttt{x.if x = 0 then Err -1 else Ok (42 / x)}, []\rangle) :: []$$

is given as an answer to the Init shadow, the linked result will contain only $42/42 = 1$.

The second example is adapted from how JavaScript programmers use React, a library for building reactive web applications. In the example, the function `make` is called with the *current state* of the reactive component `x` each time the web page is rendered, and updates `x` with its incremented value. What is noticable is that idioms such as functions that returns functions (`useState` returning `setX`) is used to hide how the library manipulates the local state. To analyze this program, one has to record what calls affect the result in what order. The answer can be obtained by linking the details of how React schedules state updates, which can change over library versions.

The example also shows how shadows are no different from "normal" values. Note the record $(\texttt{val}, 0) :: [\texttt{cur}]$ that is given as the argument to `useState`. `cur` is a shadow, yet it is consed with a concrete value $(\texttt{val}, 0)$. That is, the shadows we create live in a semantic domain, which can be subjected to regular techniques in abstract interpretation.

Our modular analysis framework consists of two parts; abstracting the shadow semantics and abstracting the semantic linking operator.

2 Syntax and Semantics

2.1 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.

2.2 Operational Semantics

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

Figure 2: Definition of the semantic domains.

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

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

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

Figure 3: The big-step operational semantics.

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

2.3 Reconciling with Conventional Backpatching

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

Figure 4: Definition of the semantic domains with memory.

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

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

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

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

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

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

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

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

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

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

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:

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

The following theorem holds:

Theorem 2.1 (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.

3 Generating and Resolving Events

Now we formulate the semantics for generating events.

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

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.

$$
\begin{aligned}
\bullet(x) &\triangleq \bot & ((x',\ell)::\sigma)(x) &\triangleq (x = x'?\ell:\sigma(x)) \\
[E](x) &\triangleq \mathsf{Read}(E,x) & ((x',w)::\sigma)(x) &\triangleq (x = x'?w:\sigma(x))
\end{aligned}
$$

Then we need to add only three rules to the semantics in Figure 3 for the semantics to incorporate events.

$$
\textsc{LinkEvent}\ \frac{\sigma\vdash e_1 \Downarrow E \qquad [E]\vdash e_2\Downarrow v}{\sigma\vdash e_1 \rtimes e_2 \Downarrow v}
\qquad
\textsc{AppEvent}\ \frac{\sigma\vdash e_1 \Downarrow E \qquad \sigma\vdash e_2\Downarrow v}{\sigma\vdash e_1\, e_2 \Downarrow \mathsf{Call}(E,v)}
\qquad
\textsc{BindEvent}\ \frac{\ell\notin \mathrm{FLoc}(\sigma) \qquad (x,\ell)::\sigma\vdash e_1\Downarrow v_1 \qquad (x,\mu\ell.v_1)::\sigma\vdash e_1\Downarrow E_2}{\sigma\vdash x \mathbin{=} e_1; e_2 \Downarrow (x,\mu\ell.v_1)::[E_2]}
$$

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

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

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

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

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

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

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

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

$$\sigma_0 \backpropto \mu\ell.v \triangleq \{\mu\ell'.v_+ \mid \ell'\notin\mathrm{FLoc}(v)\cup\mathrm{FLoc}(\sigma_0), v_+\in\sigma_0\backpropto v[\ell'/\ell]\}$$

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

$$
\mathrm{eval}(e,\sigma)\triangleq\{v \mid \sigma\vdash e\Downarrow v\} \qquad
\mathrm{eval}(e,\Sigma)\triangleq\bigcup_{\sigma\in\Sigma}\mathrm{eval}(e,\sigma) \qquad
\Sigma_0\backpropto W\triangleq\bigcup_{\substack{\sigma_0\in\Sigma_0\\ w\in W}}(\sigma_0\backpropto w)
$$

Then the following holds:

Theorem 3.1 (Advance)**.** Given $e\in\text{Expr}, \Sigma_0,\Sigma\subseteq\text{Env}$,

$$\mathrm{eval}(e,\Sigma_0\backpropto\Sigma)\subseteq\Sigma_0\backpropto\mathrm{eval}(e,\Sigma)$$

The proof of Theorem 3.1 uses some useful lemmas, such as:

Lemma 3.1 (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:

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

Lemma 3.2 (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:

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

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 3.2 (Modular analysis)**.** Assume:

1. An abstract domain $\mathrm{WVal}^\#$ that is concretized by a monotonic $\gamma \in \mathrm{WVal}^\# \to \mathcal{P}(\mathrm{WVal})$
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 3.1 (Modular analysis of linked program)**.**

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

4 CFA

4.1 Collecting semantics

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

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

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

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

$$\begin{aligned}
\mathrm{step}(t,p) \triangleq{} & [p \mapsto \{(\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\}] &&\\
\mathrm{step}(t,p) \triangleq{} & [p \mapsto \{(\sigma, \langle \lambda x.p', \sigma\rangle) | \sigma \in t(p)\}] && \text{when } \{p : \lambda x.p'\}\\
\mathrm{step}(t,p) \triangleq{} & [p_1 \mapsto \{\sigma \in \mathrm{Env} | \sigma \in t(p)\}] && \text{when } \{p : p_1\ p_2\}\\
& \cup [p_2 \mapsto \{\sigma \in \mathrm{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)\}] &&\\
\mathrm{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)\}] &&\\
\mathrm{step}(t,p) \triangleq{} & [p \mapsto \{(\sigma, \bullet) | \sigma \in t(p)\}] && \text{when } \{p : \varepsilon\}\\
\mathrm{step}(t,p) \triangleq{} & [p_1 \mapsto \bigcup_{\sigma \in t(p)} \{(x, \ell^{p_1}) :: \sigma | \ell \notin \mathrm{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

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

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 $\sigma_0 \vdash p_0 \Downarrow v_0$.

4.2 Abstract semantics

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

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

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

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

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

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

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:

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

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

Now the abstract semantic function can be given.

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

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

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

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

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 $\Sigma_0 \subseteq \gamma(\sigma_0^{\#}, t_0^{\#})$.

4.3 Abstract linking

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

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

we define:

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

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

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

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

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

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

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

We proceed by induction on the derivation for

$$w_+ \in \sigma_0 \propto w$$

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

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 $[E] \preceq t^\#(p').1$	
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 $E \preceq t^\#(p_1).2$ and $v \preceq t^\#(p_2).2$	
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 $\sigma \preceq v^\#$	
Need:	$v_+^\#.1.1(x) \ni p'$	
When:	$w = (x, w') :: \sigma$,	
Have:	$p' \in v^\#.1.1(x)$ and $w' \preceq t^\#(p').2$ and $\sigma \preceq v^\#$	
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 $\sigma \preceq t^\#(p'').1$	
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 4.1 (Substitution of values)**.**

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

Lemma 4.2 (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 4.3 (Sound $\text{Step}^{\#}$)**.**

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