Modular Analysis

Joonhyup Lee

May 7, 2024

Abstract

We present a framework of modular static analysis that analyzes open program fragments in advance and complete the whole analysis later when the fragments are closed. The framework is defined for a call-by-value lambda calculus extended with constructs for defining and linking first-class modules (a collection of bindings) that support recursive bindings for values, modules, and functors(module functions). Thanks to the abstract interpretation framework, our modular analysis framework's key is how to define the semantics, called "shadow" semantics of open program fragments so that the computation with free variables should be captured in advance and be completed later at link-time. A modular static analysis is abstractions of this shadow semantics and of the linking operation. The safety of the framework is proven in Coq. Two instances of the framwork are presented: value analyses for an applicative language(control-flow, closure analysis) and for an imperative language.

1 Problem

By modular analysis we mean a static analysis technology that analyzes programs in the compositional way. Modular analysis partially analyzes program fragments separately and then complete the analysis when all the fragments are available and linked.

Designing a modular analysis decides on two factors: how to build sound partial semantic summaries of program fragments and how to complete them at link time. A particular modular analysis strikes a balance between the two factors. Depending on how much analysis work is invested on each factor, modular analyses range from one extreme, a trivial one (wait until the whole code is available and do the whole-program analysis) to another extreme, non-trivial yet hardly-automatic ones (abduct assumptions about free variables and check them at link-time).

However, designing a modular analysis has been elusive. There is a lack of a general framework by which we can design sound modular analyses with varying balance and accuracy of our choice. We need a general framework by which static analysis designers can design non-trivial sound modular analyses.

2 Framework Sketch

Thanks to the abstract interpretation framework, the key in our modular analysis framework is how to define the semantics (we call it "shadow" semantics) of open program fragments so that the computation with free variables should be captured in advance and be completed later at link-time. A modular static analysis is then nothing but abstractions of this shadow semantics and of the linking operation.

Shadow semantics is the semantics for the computation involving free variables. For exmaple, consider the following program fragment.

```
let rec map f l =
  match l with
  | [] -> []
  | hd :: tl -> ¹(³f ⁴hd) :: ²(map f tl)
in
map ⁵g (1 :: 2 :: 3 :: [])
```

The shadow semantics of the above fragment is

$$\mathsf{Call}(\mathsf{G},1) :: \mathsf{Call}(\mathsf{G},2) :: \mathsf{Call}(\mathsf{G},3) :: [] \text{ where } \mathsf{G} = \mathsf{Read}(\mathsf{Init},\mathsf{g})$$

Shadows such as Call, Read correspond to semantic operations, like function application and reading from the environment. The Init shadow is the unknown initial environment that the fragment will execute under.

Computing a sound and finite approximation of the shadow semantics of program fragments corresponds to building a sound partial semantic summaries in advance. For the above example, we may use an abstraction

that stores summaries of each input *environment* and output *value* per program point. Using the program points that are labelled in the example, we may say that the abstract shadow that is returned is

$$\ell_1 :: \ell_2$$

where each program point stores

$$\ell_1.\text{out} \mapsto \{\mathsf{Call}^{\#}(\ell_3, \ell_4)\} \quad \ell_2.\text{out} \mapsto \{[], \ell_1 :: \ell_2\} \quad \ell_3.\text{out} \mapsto \{\mathsf{Read}^{\#}(\ell_5, \mathtt{g})\} \quad \ell_4.\text{out} \mapsto [1, 3] \quad \ell_5.\text{in} \mapsto \{\mathsf{Init}^{\#}\}$$

The shadow semantics become actual when the involved free variables are known at link time. The linking semantics, the semantics of the link operation, defines this actualization operation. For the above fragment, let's consider that a closing fragment is available.

```
let g ^1x = ^2(x + 1)
```

This is a module which returns the environment $\sigma_{0,1} = [\mathtt{g} \mapsto \langle \lambda \mathtt{x.x + 1}, [] \rangle]$. Linking this with the concrete shadow gives:

$$2 :: 3 :: 4 :: []$$

The function `g` might also be a foreign function.

```
external g : int -> int = "incr"
```

The return value of this module is $\sigma_{0,2} = [\mathtt{g} \mapsto \mathsf{Prim}(\mathtt{incr})]$, where Prim stands for a *primitive* value. Linking this with the concrete shadow gives:

$$\mathsf{PrimCall}(\mathtt{incr}, 1) :: \mathsf{PrimCall}(\mathtt{incr}, 2) :: \mathsf{PrimCall}(\mathtt{incr}, 3) :: []$$

Computing a sound and finite approximation of the linking semantics corresponds to completing partial analysis summaries at link time when the analysis results for the involved free variable are available. For the above example,

$$\sigma^{\#}_{0,1} = [\mathtt{g} \mapsto \{\langle \lambda \mathtt{x}.\ell_{-2}, \ell_{-1} \rangle\}] \text{ where } \ell_{-1}.\text{in} \mapsto []$$

is the abstract shadow for the first closing fragment, and

$$\sigma^{\#}_{0,2} = [\mathtt{g} \mapsto \{\mathsf{Prim}(\mathtt{incr})\}]$$

is the abstract shadow for the second closing fragment.

Applying a sound abstract version of the linking operator will result in

$$\ell_1.\text{out} \mapsto [2, 4] \quad \ell_2.\text{out} \mapsto \{[], \ell_1 :: \ell_2\} \quad \ell_3.\text{out} \mapsto \{\langle \lambda \mathtt{x}.\ell_{-2}, \ell_{-1} \rangle\} \quad \ell_4.\text{out} \mapsto [1, 3] \quad \ell_5.\text{in} \mapsto \sigma^{\#}_{0,1}$$

for the first shadow, and

$$\ell_1.\text{out} \mapsto \{\mathsf{PrimCall}^{\#}(\mathtt{incr}, \ell_4)\} \quad \ell_2.\text{out} \mapsto \{[], \ell_1 :: \ell_2\} \quad \ell_3.\text{out} \mapsto \{\mathsf{Prim}(\mathtt{incr})\} \quad \ell_4.\text{out} \mapsto [1, 3] \quad \ell_5.\text{in} \mapsto \sigma^{\#}_{0,2}$$

for the second shadow.

We present our framework for a call-by-value lambda calculus extended with constructs for defining and linking first-class modules (a collection of bindings) that support recursive bindings for values, modules, and functors(module functions).

The framework shows two points: how to define the shadow semantics and what to prove for the soundness of consequent modular analysis. The safety of the framework is proven in Coq. We present wwo instances of the framework: for high-order applicative language we show modular closure analysis design, and for imperative languages we show modular [TODO] analysis design.

3 Syntax and Semantics

3.1 Abstract Syntax

3.2 Operational Semantics

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

Identifiers	x	$\in$	Var	
Expression	e	$\to$	$x \mid \lambda x.e \mid e\ e$	λ-calculus
		$\mid$	$e \rtimes e$	linked expression
		$\mid$	ε	empty module
		$\mid$	$x = e\ ;\ e$	(recursive) binding

Figure 1: Abstract syntax of the language.

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

Figure 2: Definition of the semantic domains.

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

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

$$\textsc{App}\ \frac{\sigma \vdash e_1 \Downarrow \langle \lambda x.e, \sigma_1 \rangle \qquad \sigma \vdash e_2 \Downarrow v_2 \qquad (x, v_2) :: \sigma_1 \vdash e \Downarrow v}{\sigma \vdash e_1\ e_2 \Downarrow v}$$

$$\textsc{Link}\ \frac{\sigma \vdash e_1 \Downarrow \sigma_1 \qquad \sigma_1 \vdash e_2 \Downarrow v}{\sigma \vdash e_1 \rtimes e_2 \Downarrow v} \qquad \textsc{Empty}\ \frac{}{\sigma \vdash \varepsilon \Downarrow \bullet}$$

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

Figure 3: The big-step operational semantics.

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

Figure 4: Definition of the semantic domains with memory.

3.3 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 3.1 (Equivalence of semantics)**.** For all $\sigma \in \text{Env}, \sigma' \in \text{MEnv} \times \text{Mem} \times \mathcal{P}(\text{Loc}), v \in \text{Val}, v' \in \text{MVal} \times \text{Mem} \times \mathcal{P}(\text{Loc})$, 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:

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

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

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

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

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

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

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

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

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

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

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

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

4 Generating and Resolving Shadows

Now we formulate the semantics for generating shadows.

$$\begin{array}{rrcll} \text{Shadow} & S & \to & \mathsf{Init} & \text{initial environment} \\ & & | & \mathsf{Read}(S, x) & \text{read shadow} \\ & & | & \mathsf{Call}(S, v) & \text{call shadow} \\ \text{Environment} & \sigma & \to & \cdots & \\ & & | & [S] & \text{answer to an shadow} \\ \text{Value} & v & \to & \cdots & \\ & & | & S & \text{answer to an shadow} \end{array}$$

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

We extend how to read weak values given an environment.

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

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

$$\textsc{LinkShadow}\ \frac{\sigma \vdash e_1 \Downarrow S \qquad [S] \vdash e_2 \Downarrow v}{\sigma \vdash e_1 \rtimes e_2 \Downarrow v} \qquad \textsc{AppShadow}\ \frac{\sigma \vdash e_1 \Downarrow S \qquad \sigma \vdash e_2 \Downarrow v}{\sigma \vdash e_1\ e_2 \Downarrow \mathsf{Call}(S, v)} \qquad \textsc{BindShadow}\ \frac{\ell \notin \mathrm{FLoc}(\sigma) \qquad (x, \ell) :: \sigma \vdash e_1 \Downarrow v_1 \qquad (x, \mu\ell.v_1) :: \sigma \vdash e_1 \Downarrow S_2}{\sigma \vdash x = e_1; e_2 \Downarrow (x, \mu\ell.v_1) :: [S_2]}$$

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

$$\boxed{\rtimes \in \mathrm{Env} \to \mathrm{Shadow} \to \mathcal{P}(\mathrm{Val})}$$

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

$$\boxed{\rtimes \in \mathrm{Env} \to \mathrm{Env} \to \mathcal{P}(\mathrm{Env})}$$

$$\begin{array}{rl} \sigma_0 \rtimes \bullet & \triangleq \{\bullet\} \\ \sigma_0 \rtimes (x, \ell) :: \sigma & \triangleq \{(x, \ell) :: \sigma_+ | \sigma_+ \in \sigma_0 \rtimes \sigma\} \\ \sigma_0 \rtimes (x, w) :: \sigma & \triangleq \{(x, w_+) :: \sigma_+ | w_+ \in \sigma_0 \rtimes w, \sigma_+ \in \sigma_0 \rtimes \sigma\} \\ \sigma_0 \rtimes [E] & \triangleq \{\sigma_+ | \sigma_+ \in \sigma_0 \rtimes S\} \cup \{[S_+] | S_+ \in \sigma_0 \rtimes S\} \end{array}$$

$$\boxed{\rtimes \in \mathrm{Env} \to \mathrm{Val} \to \mathcal{P}(\mathrm{Val})}$$

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

$$\boxed{\rtimes \in \mathrm{Env} \to \mathrm{WVal} \to \mathcal{P}(\mathrm{WVal})}$$

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

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

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

Then the following holds:

Theorem 4.1 (Advance)**.** Given $e \in \mathrm{Expr}, \Sigma_0, \Sigma \subseteq \mathrm{Env}$,

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

The proof of Theorem 4.1 uses some useful lemmas, such as:

Lemma 4.1 (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:

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

Lemma 4.2 (Linking is compatible with reads)**.** Let $\mathrm{unroll} : \mathrm{WVal} \to \mathrm{Val}$ be defined as:

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

For $\sigma_+ \in \sigma_0 \leftthreetimes \sigma$, if $\sigma_+(x) = w_+$, we have:

$$\text{There exists } w \text{ such that } \sigma(x) = w \text{ and } \mathrm{unroll}(w_+) \in \sigma_0 \leftthreetimes \mathrm{unroll}(w)$$

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

Theorem 4.2 (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 $\leftthreetimes^\#$: $\Sigma_0 \subseteq \gamma(\sigma_0^\#)$ and $W \subseteq \gamma(w^\#) \Rightarrow \Sigma_0 \leftthreetimes W \subseteq \gamma(\sigma_0^\# \leftthreetimes^\# w^\#)$

then we have:

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

Corollary 4.1 (Modular analysis of linked program)**.**

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

5 CFA

5.1 Collecting semantics

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

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

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

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

$$
\begin{array}{lr}
\text{step}(t,p) \triangleq [p \mapsto \{(\sigma, v) | \sigma \in t(p) \text{ and } \sigma(x) = v\}] & \text{when } \{p : x\} \\
\quad \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\} \\
\quad \cup [p_2 \mapsto \{\sigma \in \text{Env} | \sigma \in t(p)\}] & \\
\quad \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)\}] & \\
\quad \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')\}] & \\
\quad \cup [p \mapsto \bigcup_{\sigma \in t(p)} \{(\sigma, \mathsf{Call}(S_1, v_2)) | (\sigma, S_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\} \\
\quad \cup [p_2 \mapsto \bigcup_{\sigma \in t(p)} \{\sigma_1 | (\sigma, \sigma_1) \in t(p_1)\}] & \\
\quad \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 \notin \text{FLoc}(\sigma)\}] & \text{when } \{p : x = p_1; p_2\} \\
\quad \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)\}] & \\
\quad \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{array}
$$

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 $\sigma_0 \vdash p_0 \Downarrow v_0$.

5.2 Abstract semantics

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

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

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

$$\textsc{Conc-ConsWVal}\ \frac{p \in \sigma^\#.1(x) \qquad w \preceq t^\#(p).2 \qquad \sigma \preceq \sigma^\#}{(x, w) :: \sigma \preceq \sigma^\#}$$

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

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

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

$$\textsc{Conc-Call}\ \frac{\mathsf{Call}^\#(p_1, p_2) \in v^\#.1.2 \qquad S \preceq t^\#(p_1).2 \qquad v \preceq t^\#(p_2).2}{\mathsf{Call}(S, v) \preceq v^\#}$$

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

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.

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

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

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

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

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

5.3 Abstract linking

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

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

we define:

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

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

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

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

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

for each $w_+ \in \sigma_0 \rtimes w$ 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

$$w_+ \in \sigma_0 \rtimes w$$

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:	$w = \mathsf{Read}(S, x)$,	
Have:	$\mathsf{Read}^\#(p', x) \in v^\#.1.2$ and $[S] \preceq t^\#(p').1$	
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:	$w = \mathsf{Call}(S, v)$,	
Have:	$\mathsf{Call}^\#(p_1, p_2) \in v^\#.1.2$ and $S \preceq t^\#(p_1).2$ 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 $\mathrm{link}^\# = \mathrm{Link}^\#(\sigma_0^\#, t^\#, t_+^\#)$ in		
For each $p \in \mathbb{P}$, when $v^\#, v_+^\# = (t^\#(p).1, \varnothing), (\mathrm{link}^\#(p).1, \varnothing)$		
or when $v^\#, v_+^\# = t^\#(p).2, \mathrm{link}^\#.2$		
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 $p'' \in t_+^\#(p').1.1(x)$
	$v_+^\# \sqsupseteq ((\lceil\rceil, \{\mathsf{Read}^\#(p', x)\}), \varnothing)$	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 $\langle \lambda x.p', p'' \rangle \in t_+^\#(p_1).2.2$
	$v_+^\# \sqsupseteq ((\lceil\rceil, \{\mathsf{Call}^\#(p_1, p_2)\}), \varnothing)$	if $t_+^\#(p_1).2.1.2 \neq \varnothing$
	$\mathrm{link}^\#(p') \sqsupseteq (t_+^\#(p'').1 \sqcup ([x \mapsto \{p_2\}], \varnothing), \varnothing)$	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 $\mathrm{link}^\#$ and the right-hand side does not depend on the value of $\mathrm{link}^\#$.

Some auxiliary lemmas:

Lemma 5.1 (Substitution of values)**.**

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

Lemma 5.2 (Sound $\mathrm{step}^\#$)**.**

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

Lemma 5.3 (Sound $\mathrm{Step}^\#$)**.**

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

6 Conclusion