Concrete Semantics for Pushdown Analysis

The Essence of Summarization

J. Ian Johnson and David Van Horn
{ianj,dvanhorn}@ccs.neu.edu
Northeastern University
Boston, MA, USA

Two things:

Two things:

Pushdown analysis is easy

Two things:

Pushdown analysis is easy

You should model your analyses concretely

Two things:

Pushdown analysis is easy

You should model your analyses concretely

Regular v Pushdown

Regular

Store:

```
(define (id x) x)

(<= (id 0) (id 1))
```

Regular

Store:

x	{0}

```
(define (id x) x)

(<= (id 0) (id 1))
```

Regular

Store:

x	{0}

```
(define (id x) x)

(<= (id 0) (id 1))
```

Regular

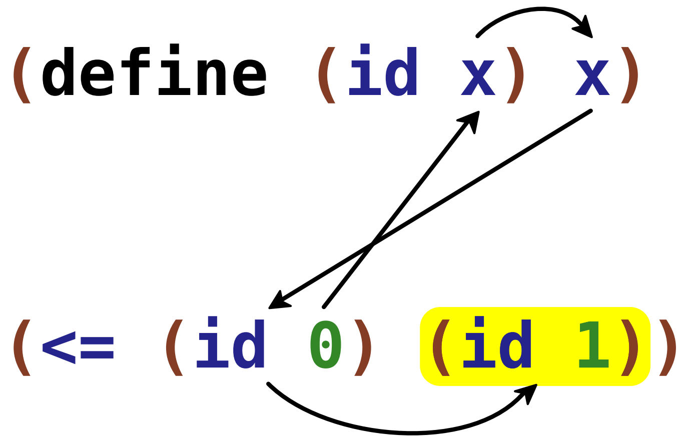
Store:

x	{0}
(id 0)	{0}

```
(define (id x) x)

(<= (id 0) (id 1))
```

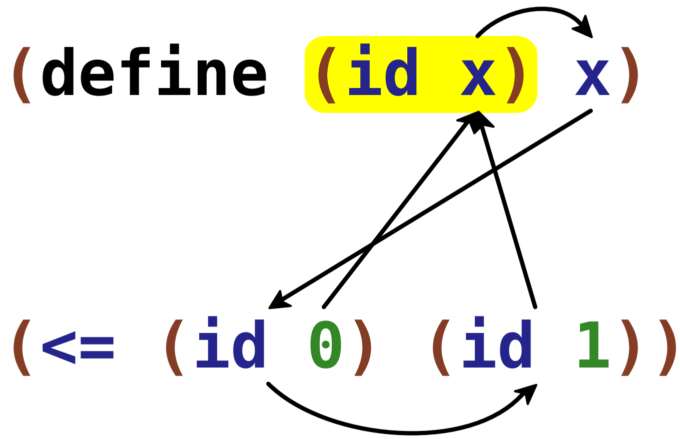
Regular



Store:

x	{0}
(id 0)	{0}

Regular



Store:

x	{0, 1}
(id 0)	{0}

Regular

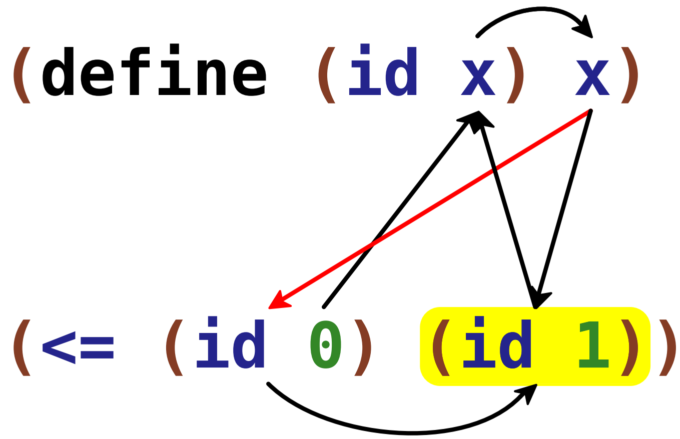
```
(define (id x) x)

(<= (id 0) (id 1))
```

Store:

x	{0, 1}
(id 0)	{0}

Regular



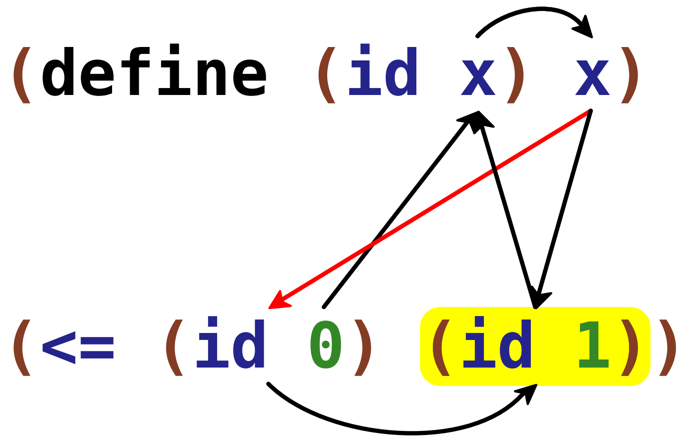
Store:

x	{0, 1}
(id 0)	{0, 1}
(id 1)	{0, 1}

Regular

Store:

`x`	`{0, 1}`
`(id 0)`	`{0, 1}`
`(id 1)`	`{0, 1}`



Result: true or false

Pushdown

Store:

```
(define (id x) x)

(<= (id 0) (id 1))
```

Pushdown

Store:

x	{0}

```
(define (id x) x)

(<= (id 0) (id 1))
```

Pushdown

Store:

x	{0}

```
(define (id x) x)

(<= (id 0) (id 1))
```

Pushdown

```
(define (id x) x)

(<= (id 0) (id 1))
```

Store:

`x`	`{0}`
`(id 0)`	`{0}`

Pushdown

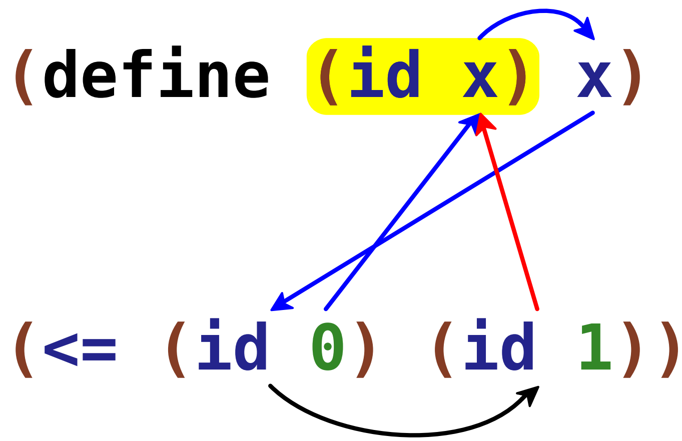
Store:

`x`	`{0}`
`(id 0)`	`{0}`

```
(define (id x) x)

(<= (id 0) (id 1))
```

Pushdown



Store:

x	{0, 1}
(id 0)	{0}

Pushdown

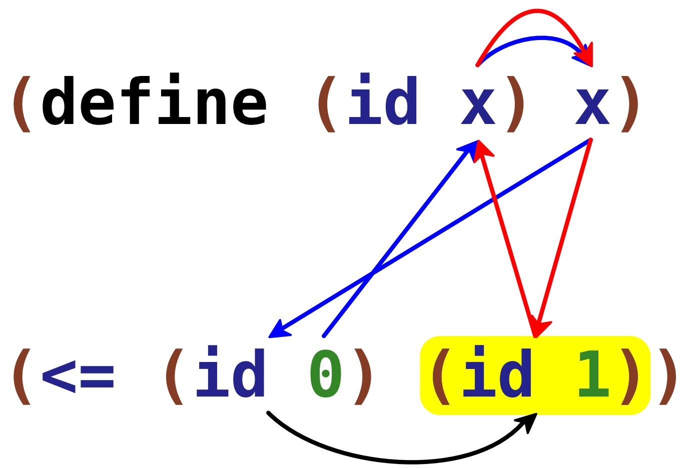
```
(define (id x) x)

(<= (id 0) (id 1))
```

Store:

x	{0, 1}
(id 0)	{0}

Pushdown



Store:

x	{0, 1}
(id 0)	{0}
(id 1)	{0, 1}

Pushdown

Store:

x	{0, 1}
(id 0)	{0}
(id 1)	{0, 1}

```
(define (id x) x)

(<= (id 0) (id 1))
```

Result: true

That was first-order [Sharir & Pnueli 1981]

That was first-order [Sharir & Pnueli 1981]

We *can* do higher-order [Vardoulakis & Shivers 2010]

That was first-order [Sharir & Pnueli 1981]

We *can* do higher-order [Vardoulakis & Shivers 2010]

That was … 981]

We *can* … is & Shivers 2010]

```
Summary, Callers, TCallers, Final ⟵ ∅
Seen, W ⟵ {(Ĩ(pr), Ĩ(pr))}
while W ≠ ∅
  remove (ς̃1, ς̃2) from W
  switch ς̃2
    case ς̃2 of Entry, CApply, Inner-CEval
      for each ς̃3 in succ(ς̃2) Propagate(ς̃1, ς̃3)
    case ς̃2 of Call
      for each ς̃3 in succ(ς̃2)
        Propagate(ς̃3, ς̃3)
        insert (ς̃1, ς̃2, ς̃3) in Callers
        for each (ς̃3, ς̃4) in Summary  Update(ς̃1, ς̃2, ς̃3, ς̃4)
    case ς̃2 of Exit-CEval
      if ς̃1 = Ĩ(pr) then
        Final(ς̃2)
      else
        insert (ς̃1, ς̃2) in Summary
        for each (ς̃3, ς̃4, ς̃1) in Callers  Update(ς̃3, ς̃4, ς̃1, ς̃2)
        for each (ς̃3, ς̃4, ς̃1) in TCallers Propagate(ς̃3, ς̃2)
    case ς̃2 of Exit-TC
      for each ς̃3 in succ(ς̃2)
        Propagate(ς̃3, ς̃3)
        insert (ς̃1, ς̃2, ς̃3) in TCallers
        for each (ς̃3, ς̃4) in Summary Propagate(ς̃1, ς̃4)

Propagate(ς̃1, ς̃2) ≜
  if (ς̃1, ς̃2) not in Seen then insert (ς̃1, ς̃2) in Seen and W

Update(ς̃1, ς̃2, ς̃3, ς̃4) ≜
  ς̃1 of the form  ([[(λl1 (u1 k1) call1)]] , d̂1, h1)
  ς̃2 of the form  ([[(f e2 (λγ2 (u2) call2))^l2]], tf2, h2)
  ς̃3 of the form  ([[(λl3 (u3 k3) call3)]] , d̂3, h2)
  ς̃4 of the form  ([[(k4 e4)^γ4]], tf4, h4)
  d̂ ⟵  Âu(e4, γ4, tf4, h4)
  tf ⟵ { tf2[f ↦ {[[(λl3 (u3 k3) call3)]]}]   S?(l2, f)
       { tf2                                  H?(l2, f) ∨ Lam?(f)
  ς̃ ⟵  ([[(λγ2 (u2) call2)]], d̂, tf, h4)
  Propagate(ς̃1, ς̃)

Final(ς̃) ≜
  ς̃ of the form  ([[(k e)^γ]], tf, h)
  insert  (halt, Âu(e, γ, tf, h), ∅, h) in Final
```

Figure 8: CFA2 workset algorithm

Deriving Pushdown Analyses

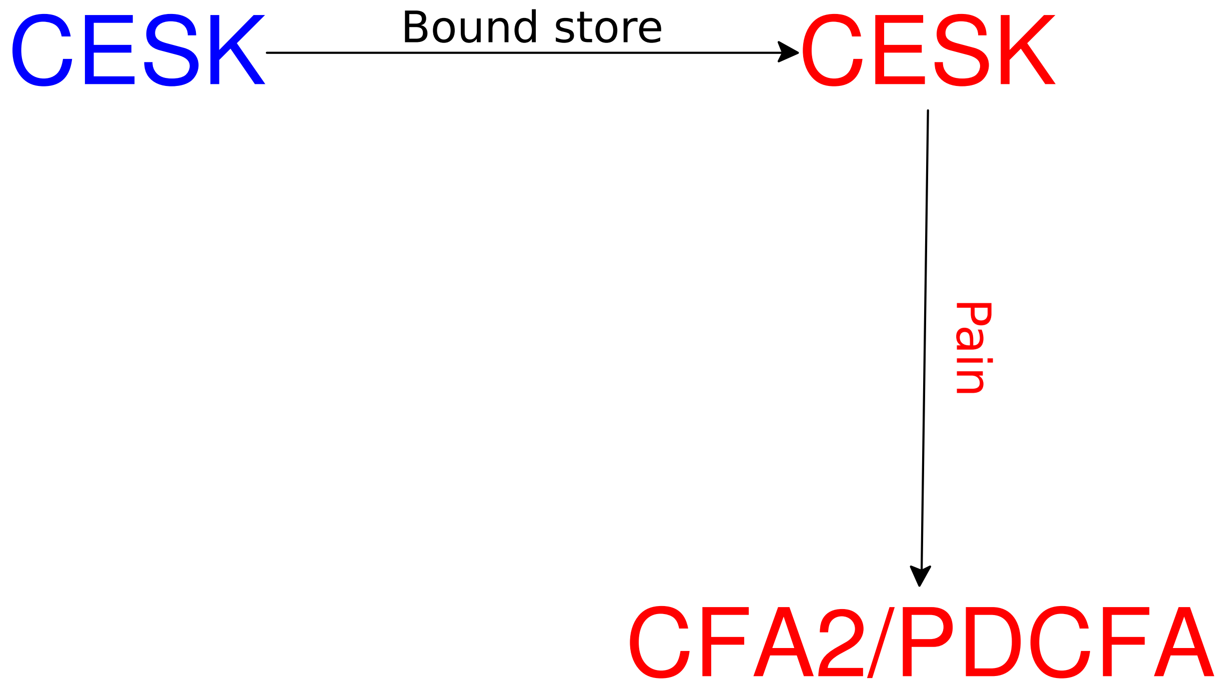
- Transform: memoize functions

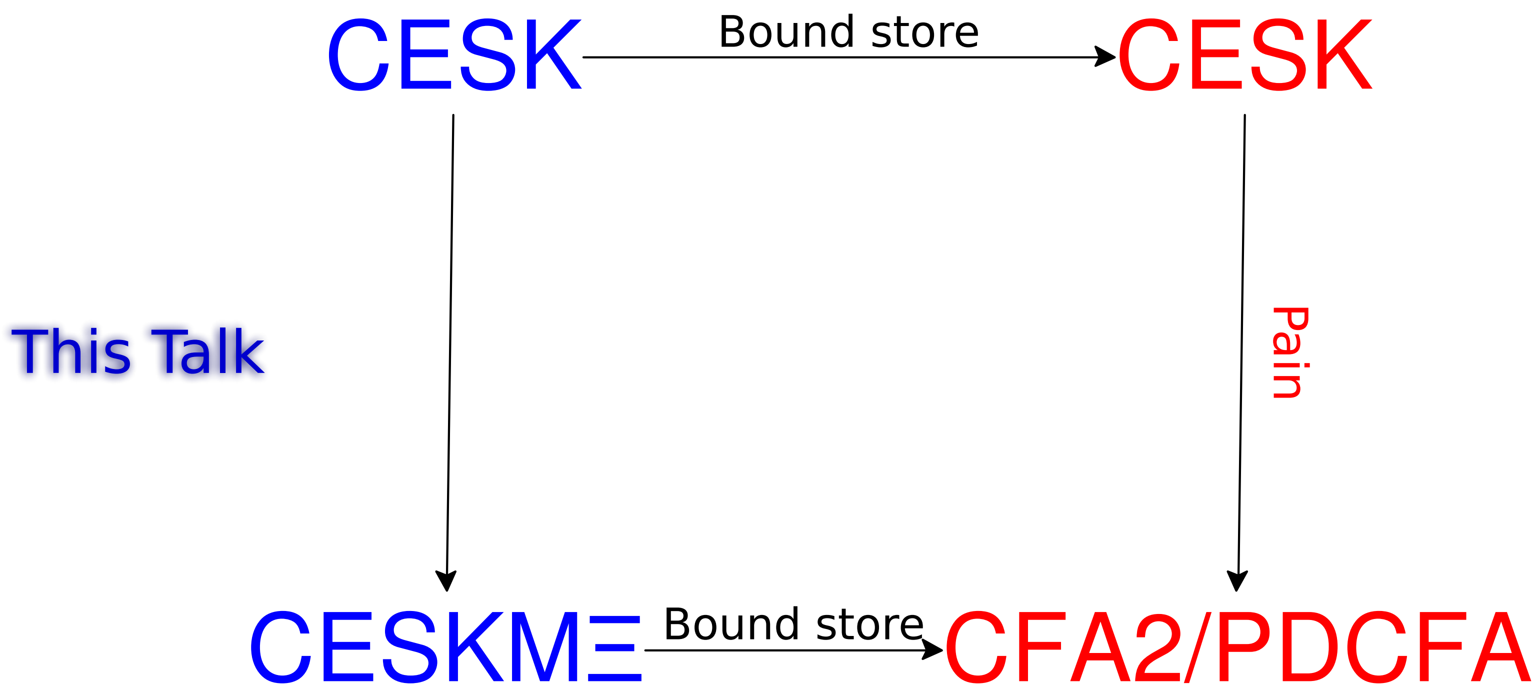
Deriving Pushdown Analyses

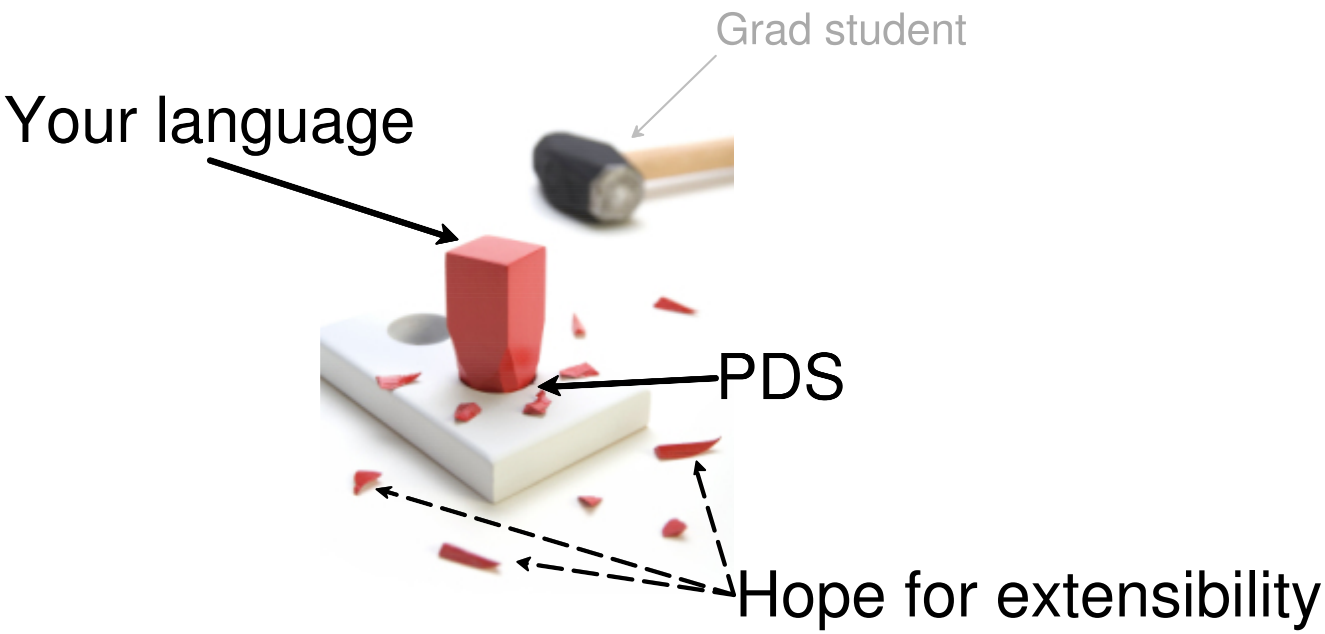
- Transform: memoize functions
- Transform: store return points for ENTIRE states

Deriving Pushdown Analyses

- Transform: memoize functions
- Transform: store return points for ENTIRE states
- Analysis: bound store


CESK
Bound store
CESK
Pain
CFA2/PDCFA


CESK
Bound store
CESK
This Talk
Pain
CESKMΞ
Bound store
CFA2/PDCFA


Grad student
Your language
PDS
Hope for extensibility

$$e ::= x \mid (e\ e) \mid \lambda x.e$$

$$v ::= \lambda x.e$$

$$E ::= [\,] \mid (E\ e) \mid (v\ E)$$

$$E[(\lambda x.e\ v)] \mapsto_{\beta v} E[e\{x{:=}v\}]$$

$$\rho \in \text{Env} = \text{Var} \to (\text{Value} \times \text{Env})$$

$$\kappa \in \text{Kont} = \text{Frame}^*$$

$$[\,] \longrightarrow [\,]$$

$$(E\ e) \longrightarrow \mathbf{ar}(e, \rho) : \kappa$$

$$(v\ E) \longrightarrow \mathbf{fn}(v, \rho) : \kappa$$

$$\rho \in \text{Env} \quad = \quad \text{Var} \to (\text{Value} \times \text{Env})$$

$$\langle x, \rho, \kappa \rangle \mapsto \langle v, \rho', \kappa \rangle$$

$$\text{if } (v, \rho') = \rho(x)$$

$$\langle (e_0\ e_1), \rho, \kappa \rangle \mapsto \langle e_0, \rho, \mathbf{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \mathbf{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \mathbf{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \mathbf{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto (v, \rho)]$$

$$\rho \in Env = Var \to Addr$$
$$\sigma \in Store = Addr \to (Value \times Env)$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$
$$\text{if } (v, \rho') = \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, ar(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, fn(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, fn(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$
$$\text{where } \rho'' = \rho'[x \mapsto a]$$
$$\sigma' = \sigma[a \mapsto (v, \rho)]$$
$$a \text{ fresh}$$

$$\rho \in \mathit{Env} = \mathit{Var} \to \mathit{Addr}$$

$$\sigma \in \mathit{Store} = \mathit{Addr} \to \wp(\mathit{Value} \times \mathit{Env})$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathtt{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathtt{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathtt{alloc}(\varsigma)$$

$$\rho \in \text{Env} = \text{Var} \to \text{Addr}$$

$$\sigma \in \text{Store} = \text{Addr} \to \wp(\text{Value} \times \text{Env})$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle$ …

$\langle v, \rho$ … $(v, \rho) : \kappa \rangle$

$\langle v, \rho, \sigma, \text{fr}$ … $, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \text{alloc}(\varsigma)$$

(1) memoize functions

$$\rho \in Env = Var \to Addr$$

$$\sigma \in Store = Addr \to \wp(Value \times Env)$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathbf{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathbf{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathbf{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathbf{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathbf{alloc}(\varsigma)$$

$$\rho \in \mathit{Env} = \mathit{Var} \rightarrow \mathit{Addr}$$

$$\sigma \in \mathit{Store} = \mathit{Addr} \rightarrow \wp(\mathit{Value} \times \mathit{Env})$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathtt{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathtt{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathtt{alloc}(\varsigma)$$

$$\rho \in \mathit{Env} = \mathit{Var} \to \mathit{Addr}$$

$$\sigma \in \mathit{Store} = \mathit{Addr} \to \wp(\mathit{Value} \times \mathit{Env})$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathtt{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathtt{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathtt{alloc}(\varsigma)$$

$$\rho \in \mathit{Env} = \mathit{Var} \to \mathit{Addr}$$

$$\sigma \in \mathit{Store} = \mathit{Addr} \to \wp(\mathit{Value} \times \mathit{Env})$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathtt{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathtt{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathit{alloc}(\varsigma)$$

$$\rho \in \mathrm{Env} = \mathrm{Var} \to \mathrm{Addr}$$

$$\sigma \in \mathrm{Store} = \mathrm{Addr} \to \wp(\mathrm{Value} \times \mathrm{Env})$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathtt{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathtt{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathtt{alloc}(\varsigma)$$

$$\rho \in Env = Var \to Addr$$

$$\sigma \in Store = Addr \to \wp(Value \times Env)$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, ar(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, fn(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, fn(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = alloc(\varsigma)$$

$$\rho \in Env = Var \rightarrow Addr$$

$$\sigma \in Store = Addr \rightarrow \wp(Value \times Env)$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, ar(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, fn(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, fn(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = alloc(\varsigma)$$

$$\rho \in Env = Var \to Addr$$

$$\sigma \in Store = Addr \to \wp(Value \times Env)$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, ar(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, ar(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, fn(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, fn(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = alloc(\varsigma)$$

$$\rho \in Env = Var \rightarrow Addr$$

$$\sigma \in Store = Addr \rightarrow \wp(Value \times Env)$$

$$\langle x, \rho, \sigma, \kappa \rangle \mapsto \langle v, \rho', \sigma, \kappa \rangle$$

$$\text{if } (v, \rho') \in \sigma(\rho(x))$$

$$\langle (e_0\ e_1), \rho, \sigma, \kappa \rangle \mapsto \langle e_0, \rho, \sigma, \mathtt{ar}(e_1, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{ar}(e, \rho) : \kappa \rangle \mapsto \langle e, \rho, \sigma, \mathtt{fn}(v, \rho) : \kappa \rangle$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$a = \mathtt{alloc}(\varsigma)$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathrm{fn}(\lambda x . e , \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathrm{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

where $\rho'' = \rho'[x \mapsto a]$

$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho'):\kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathrm{fn}(\lambda x.e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

where $\rho'' = \rho'[x \mapsto a]$

$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$

$$\langle v, \rho, \sigma, \mathrm{fn}(\lambda x . e, \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

where $\rho'' = \rho'[x \mapsto a]$

$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e , \rho'):\kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

where $\rho'' = \rho'[x \mapsto a]$

$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho'):\kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x . e , \rho') : \kappa \rangle \mapsto \langle e, \rho'', \sigma', \kappa \rangle$$

where $\rho'' = \rho'[x \mapsto a]$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathrm{fn}(\lambda x . e, \rho') : \kappa, M \rangle \mapsto \langle e, \rho'', \sigma', \mathrm{rt}(ctx) : \kappa, M \rangle$$

$$\text{or } \langle v', \rho, \kappa, M \rangle \text{ if } v' \in M(ctx)$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$ctx = (e, \rho'', \sigma')$$

$$\langle v, \rho, \sigma, \mathrm{fn}(\lambda x.e, \rho'):\kappa, M \rangle \mapsto \langle e, \rho'', \sigma', \mathrm{rt}(ctx):\kappa, M \rangle$$

$$\text{or } \langle v', \rho, \kappa, M \rangle \text{ if } v' \in M(ctx)$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$ctx = (e, \rho'', \sigma')$$

$$\langle v, \rho, \sigma, \mathrm{rt}(ctx):\kappa, M \rangle \mapsto \langle v, \rho, \sigma, \kappa, M' \rangle$$

$$\text{where } M' = M \sqcup [ctx \mapsto \{(v, \rho)\}]$$

$$\langle v, \rho, \sigma, \mathtt{fn}(\lambda x.e, \rho'):\kappa, M \rangle \mapsto \langle e, \rho'', \sigma', \mathtt{rt}(ctx):\kappa, M \rangle$$

or $\langle v', \rho, \kappa, M \rangle$ if $v' \in M(ctx)$

where $\rho'' = \rho'[x \mapsto a]$

$\{(v, \rho)\}]$

$ctx = (e, \rho'', \sigma')$

(2) store return points

$$\langle v, \rho, \sigma, \mathtt{rt}(ctx):\kappa, M \rangle \mapsto \langle v, \rho, \sigma, \kappa, M' \rangle$$

where $M' = M \sqcup [ctx \mapsto \{(v, \rho)\}]$

$$\langle v, \rho, \sigma, fn(\lambda x.e, \rho') : \kappa, M, \Xi \rangle \mapsto \langle e, \rho'', \sigma', rt(ctx), M, \Xi' \rangle$$

$$\text{or } \langle v', \rho, \kappa, M, \Xi' \rangle \text{ if } v' \in M(ctx)$$

$$\text{where } \rho'' = \rho'[x \mapsto a]$$

$$\sigma' = \sigma \sqcup [a \mapsto \{(v, \rho)\}]$$

$$ctx = (e, \rho'', \sigma')$$

$$\Xi' = \Xi \sqcup [ctx \mapsto \{\kappa\}]$$

$$\langle v, \rho, \sigma, rt(ctx), M, \Xi \rangle \mapsto \langle v, \rho, \sigma, \kappa, M', \Xi \rangle$$

$$\text{if } \kappa \in \Xi(ctx)$$

$$\text{where } M' = M \sqcup [ctx \mapsto \{(v, \rho)\}]$$

How does this look?

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_0

Memo

Store in rt: N/A

Contexts

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_1

f_0	id
y_0	1

Memo

Store in rt: σ_1

Contexts

$\langle$(f y) ρ_1 $\sigma_1\rangle$ (let* (... [n1 •] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_2

f_0	id
y_0	1
x_0	1

Memo

Store in rt: σ_2

Contexts

⟨(f y) ρ_1 σ_1⟩ (let* (... [n1 •] ...) ...)

⟨x ρ_1 σ_2⟩ (let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_2

f_0	id
y_0	1
x_0	1

Memo

⟨x ρ_1 σ_2⟩	1

Store in rt: σ_2

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_2

f_0	id
y_0	1
x_0	1

Memo

⟨x ρ_1 σ_2⟩	1
⟨(f y) ρ_1 σ_1⟩	1

Store in rt: σ_1

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)]])
  (+ n1 n2))
```

Store: σ_3

f_0	id
y_0	1
x_0	1
$n1_0$	1

Memo

$\langle$x ρ_1 $\sigma_2\rangle$	1
$\langle$(f y) ρ_1 $\sigma_1\rangle$	1

Store in rt: N/A

Contexts

$\langle$(f y) ρ_1 $\sigma_1\rangle$	(let* (... [n1 •] ...) ...)
$\langle$x ρ_1 $\sigma_2\rangle$	(let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_4

f_0, f_1	id
y_0	1
x_0	1
$n1_0$	1
y_1	2

Memo

⟨x ρ_1 σ_2⟩	1
⟨(f y) ρ_1 σ_1⟩	1

Store in rt: σ_4

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)
⟨(f y) ρ_4 σ_4⟩	(let* (... [n2 •]) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_5

f_0, f_1	id
y_0	1
x_0	1
$n1_0$	1
y_1	2
x_1	2

Memo

⟨x ρ_1 σ_2⟩	1
⟨(f y) ρ_1 σ_1⟩	1

Store in rt: σ_5

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)
⟨(f y) ρ_4 σ_4⟩	(let* (... [n2 •]) ...)
⟨x ρ_5 σ_5⟩	(let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_5

f_0, f_1	id
y_0	1
x_0	1
$n1_0$	1
y_1	2
x_1	2

Memo

⟨x ρ_1 σ_2⟩	1
⟨(f y) ρ_1 σ_1⟩	1
⟨x ρ_5 σ_5⟩	2

Store in rt: σ_5

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)
⟨(f y) ρ_4 σ_4⟩	(let* (... [n2 •]) ...)
⟨x ρ_5 σ_5⟩	(let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_5

f_0, f_1	id
y_0	1
x_0	1
$n1_0$	1
y_1	2
x_1	2

Memo

⟨x ρ_1 σ_2⟩	1
⟨(f y) ρ_1 σ_1⟩	1
⟨x ρ_5 σ_5⟩	2
⟨(f y) ρ_4 σ_4⟩	2

Store in rt: σ_4

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)
⟨(f y) ρ_4 σ_4⟩	(let* (... [n2 •]) ...)
⟨x ρ_5 σ_5⟩	(let* (... [app (λ (f y) •)] ...) ...)

```
(let* ([id (λ (x) x)]
       [app (λ (f y) (f y))]
       [n1 (app id 1)]
       [n2 (app id 2)])
  (+ n1 n2))
```

Store: σ_6

f_0, f_1	id
y_0	1
x_0	1
$n1_0$	1
y_1	2
x_1	2
$n2_0$	2

Store in rt: N/A

Memo

⟨x ρ_1 σ_2⟩	1
⟨(f y) ρ_1 σ_1⟩	1
⟨x ρ_5 σ_5⟩	2
⟨(f y) ρ_4 σ_4⟩	2

Contexts

⟨(f y) ρ_1 σ_1⟩	(let* (... [n1 •] ...) ...)
⟨x ρ_1 σ_2⟩	(let* (... [app (λ (f y) •)] ...) ...)
⟨(f y) ρ_4 σ_4⟩	(let* (... [n2 •]) ...)
⟨x ρ_5 σ_5⟩	(let* (... [app (λ (f y) •)] ...) ...)

$$\langle e, \rho, \sigma, \kappa, M, \Xi \rangle$$

$$\begin{aligned}
\rho &\in \text{Env} &=&\quad \text{Var} \to \text{Addr} \\
\sigma &\in \text{Store} &=&\quad \text{Addr} \to \wp(\text{Value} \times \text{Env}) \\
M &\in \text{Memo} &=&\quad \text{Expr} \times \text{Env} \times \text{Store} \to \wp(\text{Value}) \\
\Xi &\in \text{KTable} &=&\quad \text{Expr} \times \text{Env} \times \text{Store} \to \wp(\text{Kont})
\end{aligned}$$

$$\kappa ::= [\,] \mid \mathbf{rt}(e, \rho, \sigma) \mid \varphi : \kappa$$

$$\varphi ::= \mathbf{ar}(e, \rho) \mid \mathbf{fn}(v, \rho)$$

Two things:

Pushdown analysis is easy

You should model your analyses concretely

```
E[(reset F[(shift k e)])]
            ↦
E[e{k:=(λ (x) F[x])}]
```

```
E[(reset F[(shift k e)])]
          ↦
E[e{k:=(λ (x) F[x])}]
```

F doesn't contain any resets

Deriving Pushdown Analyses

- Transform: memoize functions / continuations
- Transform: store return points for ENTIRE states
- Analysis: bound store

To Conclude

- Design: Model abstract mechanisms concretely

To Conclude

- Design: Model abstract mechanisms concretely
- Pushdown: Memo and local continuation tables

To Conclude

- Design: Model abstract mechanisms concretely
- Pushdown: Memo and local continuation tables
- Works for control operators / GC (not shown)

To Conclude

- Design: Model abstract mechanisms concretely
- Pushdown: Memo and local continuation tables
- Works for control operators / GC (not shown)

https://github.com/ianj/pushdown-shift-reset

Thank you

Garbage collection

Read root addresses of κ through Ξ

$$\mathcal{T}(\mathtt{rt}(\mathtt{e},\ \rho,\ \sigma)) = \bigcup\{\mathcal{T}(\kappa) : \kappa \in \Xi(\mathtt{e},\ \rho,\ \sigma)\}$$