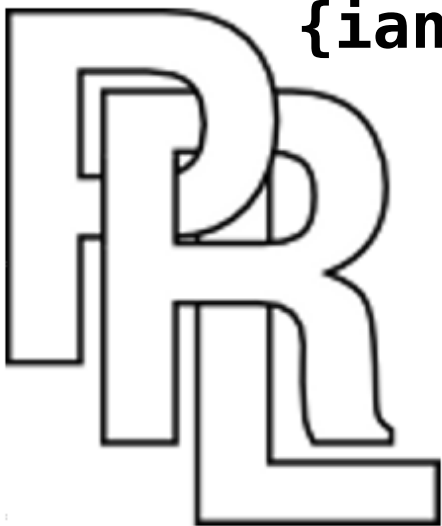


Designing Precise Pushdown Higher-Order Flow Analyses

J. Ian Johnson, David Van Horn and Olin Shivers
{ianj, dvanhorn, shivers}@ccs.neu.edu

Northeastern University
Boston, MA, USA



Flow analysis is indispensable

Flow analysis is indispensable

Every language should have one

Analysis is great

- Compiler optimizations

Analysis is great

- Compiler optimizations
- Type reconstruction

Analysis is great

- Compiler optimizations
- Type reconstruction
- Deadlock detection

Analysis is great

- Compiler optimizations
- Type reconstruction
- Deadlock detection
- Data race detection

Analysis is great

- Compiler optimizations
- Type reconstruction
- Deadlock detection
- Data race detection
- Code quality enforcement

Analysis is great

- Compiler optimizations
- Type reconstruction
- Deadlock detection
- Data race detection
- Code quality enforcement
- *Hundreds* more

Analysis is terrible

- Annoying false positives

Analysis is terrible

- Annoying false positives
- Takes too long

Analysis is terrible

- Annoying false positives
- Takes too long
- Hard to implement

Analysis is terrible

- Annoying false positives
- Takes too long
- Hard to implement
- Hard to show correct

Analysis is terrible

- Annoying false positives
- Takes too long
- Hard to implement
- Hard to show correct
- The good ones inherently first-order

Analysis is terrible

- Annoying false positives
- Takes too long
- Hard to implement
- Hard to show correct
- The good ones inherently first-order

We can fix this

Analysis is terrible

- Annoying false positives
- Takes too long
- Hard to implement
- Hard to show correct
- The good ones inherently first-order

We can fix this

First some history

Flow Graphs vs. Programs

Higher-order

First-order

(Hecht 77) CFG + lattice

Flow Graphs vs. Programs

Higher-order

(Jones & Muchnick 82) Program + set constraints

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

Flow Graphs vs. Programs

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

Flow Graphs vs. Programs

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

Flow Graphs vs. Programs

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

Flow Graphs vs. Programs

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

Flow Graphs vs. Programs

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

Graphs are a bad abstraction

Flow Graphs vs. Programs

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring



Graphs are a bad abstraction

Flow Graphs vs. Programs

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

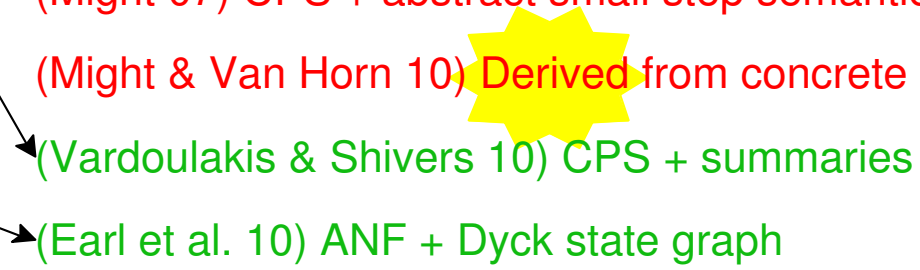
(Earl et al. 10) ANF + Dyck state graph

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring



Flow Graphs vs. Programs

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

?

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring



Analyses are derivable [Might & Van Horn 2010]

- Start: concrete machine semantics

Analyses are derivable [Might & Van Horn 2010]

- Start: concrete machine semantics
- Simple transforms: put semantics in right form

Analyses are derivable [Might & Van Horn 2010]

- Start: concrete machine semantics
- Simple transforms: put semantics in right form
- Analysis: pointwise-abstraction

Functions Without Stack

```
(define (sqr x) (* x x))
```

```
(sqrt (+ (sqr y) (sqr z)))
```

Functions Without Stack

```
(define (sqr x) (* x x))
```

```
(sqrt (+ (sqr y) (sqr z)))
```



Functions Without Stack

```
(define (sqr x) (* x x))
```

```
(sqrt (+ (sqr y) (sqr z)))
```

OCFA: Not even terminating!

A diagram consisting of two curved arrows forming a cycle. One arrow starts from the 'sqr' function call in the second line of code and points to the 'sqr' function definition in the first line. The other arrow starts from the 'sqr' function definition in the first line and points back to the 'sqr' function call in the second line, illustrating a recursive call loop.

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {( $\tilde{I}(pr)$ ,  $\tilde{I}(pr)$ )}
03  while W ≠ ∅
04    remove ( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ) from W
05    switch  $\tilde{\zeta}_2$ 
06      case  $\tilde{\zeta}_2$  of Entry, CApply, Inner-CEval
07        for each  $\tilde{\zeta}_3$  in succ( $\tilde{\zeta}_2$ ) Propagate( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_3$ )
08      case  $\tilde{\zeta}_2$  of Call
09        for each  $\tilde{\zeta}_3$  in succ( $\tilde{\zeta}_2$ )
10          Propagate( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_3$ )
11          insert ( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ,  $\tilde{\zeta}_3$ ) in Callers
12          for each ( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ ) in Summary Update( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ,  $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ )
13      case  $\tilde{\zeta}_2$  of Exit-CEval
14        if  $\tilde{\zeta}_1 = \tilde{I}(pr)$  then
15          Final( $\tilde{\zeta}_2$ )
16        else
17          insert ( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ) in Summary
18          for each ( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ ,  $\tilde{\zeta}_1$ ) in Callers Update( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ ,  $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ )
19          for each ( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ ,  $\tilde{\zeta}_1$ ) in TCallers Propagate( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_2$ )
20      case  $\tilde{\zeta}_2$  of Exit-TC
21        for each  $\tilde{\zeta}_3$  in succ( $\tilde{\zeta}_2$ )
22          Propagate( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_3$ )
23          insert ( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ,  $\tilde{\zeta}_3$ ) in TCallers
24          for each ( $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ ) in Summary Propagate( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_4$ )

Propagate( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ )  $\triangleq$ 
25  if ( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ) not in Seen then insert ( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ) in Seen and W

Update( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}_2$ ,  $\tilde{\zeta}_3$ ,  $\tilde{\zeta}_4$ )  $\triangleq$ 
26   $\tilde{\zeta}_1$  of the form ( $[(\lambda_{l_1}(u_1 k_1) call_1)]$ ,  $\hat{d}_1$ ,  $h_1$ )
27   $\tilde{\zeta}_2$  of the form ( $[(f e_2 (\lambda_{\gamma_2}(u_2) call_2))^{t_2}]$ ,  $tf_2$ ,  $h_2$ )
28   $\tilde{\zeta}_3$  of the form ( $[(\lambda_{l_3}(u_3 k_3) call_3)]$ ,  $\hat{d}_3$ ,  $h_2$ )
29   $\tilde{\zeta}_4$  of the form ( $[(k_4 e_4)^{\gamma_4}]$ ,  $tf_4$ ,  $h_4$ )
30   $\hat{d} \leftarrow \tilde{A}_u(e_4, \gamma_4, tf_4, h_4)$ 
31   $tf \leftarrow \begin{cases} tf_2[f \mapsto \{[(\lambda_{l_3}(u_3 k_3) call_3)]]\}] & S_{\gamma}(l_2, f) \\ tf_2 & H_{\gamma}(l_2, f) \vee Lam_{\gamma}(f) \end{cases}$ 
32   $\tilde{\zeta} \leftarrow ([(\lambda_{\gamma_2}(u_2) call_2)], \hat{d}, tf, h_4)$ 
33  Propagate( $\tilde{\zeta}_1$ ,  $\tilde{\zeta}$ )

Final( $\tilde{\zeta}$ )  $\triangleq$ 
34   $\tilde{\zeta}$  of the form ( $[(k e)^{\gamma}]$ ,  $tf$ ,  $h$ )
35  insert (halt,  $\tilde{A}_u(e, \gamma, tf, h)$ ,  $\emptyset$ ,  $h$ ) in Final

```

Figure 8: CFA2 workset algorithm

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {( $\tilde{I}(pr)$ ,  $\tilde{I}(pr)$ )}
03  while W ≠ ∅
04    remove ( $\zeta_1$ ,  $\zeta_2$ ) from W
05    switch  $\zeta_2$ 
06      case  $\zeta_2$  of Entry, CApply, Inner-CEval
07        for each  $\zeta_3$  in succ( $\zeta_2$ ) Propagate( $\zeta_1$ ,  $\zeta_3$ )
08      case  $\zeta_2$  of Call
09        for each  $\zeta_3$  in succ( $\zeta_2$ )
10          Propagate( $\zeta_3$ ,  $\zeta_3$ )
11          insert ( $\zeta_1$ ,  $\zeta_2$ ,  $\zeta_3$ ) in Callers
12          for each ( $\zeta_3$ ,  $\zeta_4$ ) in Summary Update( $\zeta_1$ ,  $\zeta_2$ ,  $\zeta_3$ ,  $\zeta_4$ )
13      case  $\zeta_2$  of Exit-CEval
14        if  $\zeta_1 = \tilde{I}(pr)$  then
15          Final( $\zeta_2$ )
16        else
17          insert ( $\zeta_1$ ,  $\zeta_2$ ) in Summary
18          for each ( $\zeta_3$ ,  $\zeta_4$ ,  $\zeta_1$ ) in Callers Update( $\zeta_3$ ,  $\zeta_4$ ,  $\zeta_1$ ,  $\zeta_2$ )
19          for each ( $\zeta_3$ ,  $\zeta_4$ ,  $\zeta_1$ ) in TCallers Propagate( $\zeta_3$ ,  $\zeta_2$ )
20      case  $\zeta_2$  of Exit-TC
21        for each  $\zeta_3$  in succ( $\zeta_2$ )
22          Propagate( $\zeta_3$ ,  $\zeta_3$ )
23          insert ( $\zeta_1$ ,  $\zeta_2$ ,  $\zeta_3$ ) in TCallers
24        for each ( $\zeta_3$ ,  $\zeta_4$ ) in Summary Propagate( $\zeta_1$ ,  $\zeta_4$ )

Propagate( $\zeta_1$ ,  $\zeta_2$ )  $\triangleq$ 
25  if ( $\zeta_1$ ,  $\zeta_2$ ) not in Seen then insert ( $\zeta_1$ ,  $\zeta_2$ ) in Seen and W

Update( $\zeta_1$ ,  $\zeta_2$ ,  $\zeta_3$ ,  $\zeta_4$ )  $\triangleq$ 
26   $\zeta_1$  of the form  $([(\lambda_{l_1}(u_1 k_1) call_1)] , \hat{d}_1, h_1)$ 
27   $\zeta_2$  of the form  $([(f e_2 (\lambda_{\gamma_2}(u_2) call_2))^{t_2}] , tf_2, h_2)$ 
28   $\zeta_3$  of the form  $([(\lambda_{l_3}(u_3 k_3) call_3)] , \hat{d}_3, h_2)$ 
29   $\zeta_4$  of the form  $([(k_4 e_4)^{\gamma_4}] , tf_4, h_4)$ 
30   $\hat{d} \leftarrow \hat{A}_u(e_4, \gamma_4, tf_4, h_4)$ 
31   $tf \leftarrow \begin{cases} tf_2[f \mapsto \{[(\lambda_{l_3}(u_3 k_3) call_3)]]\}] & S_{\gamma}(l_2, f) \\ tf_2 & H_{\gamma}(l_2, f) \vee Lam_{\gamma}(f) \end{cases}$ 
32   $\zeta \leftarrow ((\lambda_{\gamma_2}(u_2) call_2)] , \hat{d}, tf, h_4)$ 
33  Propagate( $\zeta_1$ ,  $\zeta$ )

Final( $\zeta$ )  $\triangleq$ 
34   $\zeta$  of the form  $([(k e)^{\gamma}] , tf, h)$ 
35  insert (halt,  $\hat{A}_u(e, \gamma, tf, h), \emptyset, h)$  in Final

```

Figure 8: CFA2 workset algorithm

$$\begin{aligned}
\mathcal{F}'(M) &= f, \text{ where} \\
M &= (Q, \Gamma, \delta, q_0) \\
f(G, G_e, \Delta G, \Delta H) &= (G', G'_e, \Delta G', \Delta H' - H), \text{ where} \\
(S, \Gamma, E, s_0) &= G \\
(S, H) &= G_e \\
(\Delta S, \Delta E) &= \Delta G \\
(\Delta E_0, \Delta H_0) &= \bigcup_{s \in \Delta S} sprout_M(s) \\
(\Delta E_1, \Delta H_1) &= \bigcup_{(s, \gamma_+, s') \in \Delta E} addPush_M(G, G_e)(s, \gamma_+, s') \\
(\Delta E_2, \Delta H_2) &= \bigcup_{(s, \gamma_-, s') \in \Delta E} addPop_M(G, G_e)(s, \gamma_-, s') \\
(\Delta E_3, \Delta H_3) &= \bigcup_{(s, e, s') \in \Delta E} addEmpty_M(G, G_e)(s, s') \\
(\Delta E_4, \Delta H_4) &= \bigcup_{(s, s') \in \Delta H} addEmpty_M(G, G_e)(s, s') \\
S' &= S \cup \Delta S \\
E' &= E \cup \Delta E \\
H' &= H \cup \Delta H \\
\Delta E' &= \Delta E_0 \cup \Delta E_1 \cup \Delta E_2 \cup \Delta E_3 \cup \Delta E_4 \\
\Delta S' &= \{s' : (s, g, s') \in \Delta E'\} \\
\Delta H' &= \Delta H_0 \cup \Delta H_1 \cup \Delta H_2 \cup \Delta H_3 \cup \Delta H_4 \\
G' &= (S \cup \Delta S, \Gamma, E', q_0) \\
G'_e &= (S', H') \\
\Delta G' &= (\Delta S' - S', \Delta E' - E').
\end{aligned}$$

Figure 3. The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {(\tilde{I}(pr), \tilde{I}(pr))}
03  while W ≠ ∅
04    remove (\zeta_1, \zeta_2) from W
05    switch \zeta_2
06      case \zeta_2 of Entry, CApply, Inner-CEval
07        for each \zeta_3 in succ(\zeta_2) Propagate(\zeta_1, \zeta_3)
08      case \zeta_2 of Call
09        for each \zeta_3 in succ(\zeta_2)
10          Propagate(\zeta_3, \zeta_3)
11          insert (\zeta_1, \zeta_2, \zeta_3) in Callers
12          for each (\zeta_3, \zeta_4) in Summary Update(\zeta_1, \zeta_2, \zeta_3, \zeta_4)
13      case \zeta_2 of Exit-CEval
14        if \zeta_1 = \tilde{I}(pr) then
15          Final(\zeta_2)
16        else
17          insert (\zeta_1, \zeta_2) in Summary
18          for each (\zeta_3, \zeta_4, \zeta_1) in Callers Update(\zeta_3, \zeta_4, \zeta_1, \zeta_2)
19          for each (\zeta_3, \zeta_4, \zeta_1) in TCallers Propagate(\zeta_3, \zeta_2)
20      case \zeta_2 of Exit-TC
21        for each \zeta_3 in succ(\zeta_2)
22          Propagate(\zeta_3, \zeta_3)
23          insert (\zeta_1, \zeta_2, \zeta_3) in TCallers
24          for each (\zeta_3, \zeta_4) in Summary Propagate(\zeta_1, \zeta_4)
25
26  Propagate(\zeta_1, \zeta_2) \triangleq
27  if (\zeta_1, \zeta_2) not in Seen then insert (\zeta_1, \zeta_2) in Seen and W
28
29  Update(\zeta_1, \zeta_2, \zeta_3, \zeta_4) \triangleq
30  \zeta_1 of the form ([(\lambda_{l_1}(u_1 k_1) call_1)], \hat{d}_1, h_1)
31  \zeta_2 of the form ([(\lambda_{\gamma_2}(u_2) call_2)]^{t_2}], tf_2, h_2)
32  \zeta_3 of the form ([(\lambda_{l_3}(u_3 k_3) call_3)], \hat{d}_3, h_2)
33  \zeta_4 of the form ([(\lambda_{\gamma_4}(u_4) call_4)], tf_4, h_4)
34  \hat{d} \leftarrow \tilde{\mathcal{A}}_u(e_4, \gamma_4, tf_4, h_4)
35  tf \leftarrow \begin{cases} tf_2[f \mapsto \{[(\lambda_{l_3}(u_3 k_3) call_3)]]\}] & S_\gamma(l_2, f) \\ tf_2 & H_\gamma(l_2, f) \vee Lam_\gamma(f) \end{cases}
36  \zeta \leftarrow ([(\lambda_{\gamma_2}(u_2) call_2)], \hat{d}, tf, h_4)
37  Propagate(\zeta_1, \zeta)
38
39  Final(\zeta) \triangleq
40  \zeta of the form ([(\lambda_{\gamma_2}(u_2) call_2)], tf, h)
41  insert (halt, \tilde{\mathcal{A}}_u(e, \gamma, tf, h), \emptyset, h) in Final

```

Figure 8: CFA2 workset algorithm

$$\begin{aligned}
 & sprout_{(Q, \Gamma, \delta)}(s) = (\Delta E, \Delta H), \text{ where} \\
 \Delta E &= \left\{ s \xrightarrow{e} q : s \xrightarrow{e} q \in \delta \right\} \cup \left\{ s \xrightarrow{\gamma_+} q : s \xrightarrow{\gamma_+} q \in \delta \right\} \\
 \Delta H &= \left\{ s \mapsto q : s \xrightarrow{e} q \in \delta \right\}. \\
 (\Delta S, \Delta E) &= \Delta G \\
 (\Delta E_0, \Delta H_0) &= \bigcup_{s \in \Delta S} sprout_M(s) \\
 (\Delta E_1, \Delta H_1) &= \bigcup_{(s, \gamma_+, s') \in \Delta E} addPush_M(G, G_e)(s, \gamma_+, s') \\
 (\Delta E_2, \Delta H_2) &= \bigcup_{(s, \gamma_-, s') \in \Delta E} addPop_M(G, G_e)(s, \gamma_-, s') \\
 (\Delta E_3, \Delta H_3) &= \bigcup_{(s, e, s') \in \Delta E} addEmpty_M(G, G_e)(s, s') \\
 (\Delta E_4, \Delta H_4) &= \bigcup_{(s, s') \in \Delta H} addEmpty_M(G, G_e)(s, s') \\
 S' &= S \cup \Delta S \\
 E' &= E \cup \Delta E \\
 H' &= H \cup \Delta H \\
 \Delta E' &= \Delta E_0 \cup \Delta E_1 \cup \Delta E_2 \cup \Delta E_3 \cup \Delta E_4 \\
 \Delta S' &= \{s' : (s, g, s') \in \Delta E'\} \\
 \Delta H' &= \Delta H_0 \cup \Delta H_1 \cup \Delta H_2 \cup \Delta H_3 \cup \Delta H_4 \\
 G' &= (S \cup \Delta S, \Gamma, E', g_0) \\
 G'_e &= (S', H') \\
 \Delta G' &= (\Delta S' - S', \Delta E' - E').
 \end{aligned}$$

Figure 3. The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {{I(pr), I(pr)}}
03  while W ≠ ∅
04    remove (ξ1, ξ2) from W
05    switch ξ2
06      case ξ2 of Entry, CApply, Inner-CEval
07        for each ξ3 in succ(ξ2) Propagate(ξ1, ξ3)
08      case ξ2 of Call
09        for each ξ3 in succ(ξ2)
10          Propagate(ξ3, ξ3)
11          insert (ξ1, ξ2, ξ3) in Callers
12          for each (ξ3, ξ4) in Summary Update(ξ1, ξ2, ξ3, ξ4)
13      case ξ2 of Exit-CEval
14        if ξ1 = I(pr) then
15          Final(ξ2)
16        else
17          insert (ξ1, ξ2) in Summary
18          for each (ξ3, ξ4, ξ1) in Callers Update(ξ3, ξ4, ξ1, ξ2)
19          for each (ξ3, ξ4, ξ1) in TCallers Propagate(ξ3, ξ2)
20      case ξ2 of Exit-TC
21        for each ξ3 in succ(ξ2)
22          Propagate(ξ3, ξ3)
23          insert (ξ1, ξ2, ξ3) in TCallers
24          for each (ξ3, ξ4) in Summary Propagate(ξ1, ξ4)
25
26  Propagate(ξ1, ξ2) ≜
27    if (ξ1, ξ2) not in Seen then insert (ξ1, ξ2) in Seen and W
28
29  Update(ξ1, ξ2, ξ3, ξ4) ≜
30    ξ1 of the form ([[λ1(u1 k1) call1]], d̂1, h1)
31    ξ2 of the form ([[f e2 (λ2(u2) call2)t2], tf2, h2)
32    ξ3 of the form ([[λ3(u3 k3) call3]], d̂3, h2)
33    ξ4 of the form ([[k4 e4]t4], tf4, h4)
34    d̂ ← Au(e4, γ4, tf4, h4)
35    tf ← { tf2[f ↦ [[λ3(u3 k3) call3]]] Sγ(l2, f)
36           { tf2 Hγ(l2, f) ∨ Lamγ(f)
37
38    ξ ← ([[λ2(u2) call2]], d̂, tf, h4)
39    Propagate(ξ1, ξ)
40
41  Final(ξ) ≜
42    ξ of the form ([[k e]γ], tf, h)
43    insert (halt, Au(e, γ, tf, h), ∅, h) in Final

```

Figure 8: CFA2 workset algorithm

$\text{sprout}_{(Q, \Gamma, \delta)}(s) = (\Delta E, \Delta H), \text{ where}$ $\Delta E = \left\{ s \xrightarrow{\epsilon} q : s \xrightarrow{\epsilon} q \in \delta \right\} \cup \left\{ s \xrightarrow{\gamma^+} q : s \xrightarrow{\gamma^+} q \in \delta \right\}$ $\Delta H = \left\{ s \mapsto q : s \xrightarrow{\epsilon} q \in \delta \right\}.$
$\text{addPush}_{(Q, \Gamma, \delta)}(G, G_e)(s \xrightarrow{\gamma^+} q) = (\Delta E, \Delta H), \text{ where}$ $\Delta E = \left\{ q' \xrightarrow{\gamma^-} q'' : q' \in \overline{G_e}[q] \text{ and } q' \xrightarrow{\gamma^-} q'' \in \delta \right\}$ $\Delta H = \left\{ s \mapsto q'' : q' \in \overline{G_e}[q] \text{ and } q' \xrightarrow{\gamma^-} q'' \in \delta \right\}.$

$$(\Delta E_3, \Delta H_3) = \bigcup_{(s, e, s') \in \Delta E} \text{addEmpty}_M(G, G_e)(s, s')$$

$$(\Delta E_4, \Delta H_4) = \bigcup_{(s, s') \in \Delta H} \text{addEmpty}_M(G, G_e)(s, s')$$

$$S' = S \cup \Delta S$$

$$E' = E \cup \Delta E$$

$$H' = H \cup \Delta H$$

$$\Delta E' = \Delta E_0 \cup \Delta E_1 \cup \Delta E_2 \cup \Delta E_3 \cup \Delta E_4$$

$$\Delta S' = \{s' : (s, g, s') \in \Delta E'\}$$

$$\Delta H' = \Delta H_0 \cup \Delta H_1 \cup \Delta H_2 \cup \Delta H_3 \cup \Delta H_4$$

$$G' = (S \cup \Delta S, \Gamma, E', g_0)$$

$$G'_e = (S', H')$$

$$\Delta G' = (\Delta S' - S', \Delta E' - E').$$

Figure 3. The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {(\tilde{I}(pr), \tilde{I}(pr))}
03  while W ≠ ∅
04    remove (\zeta_1, \zeta_2) from W
05    switch \zeta_2
06      case \zeta_2 of Entry, CApply, Inner-CEval
07        for each \zeta_3 in succ(\zeta_2) Propagate(\zeta_1, \zeta_3)
08      case \zeta_2 of Call
09        for each \zeta_3 in succ(\zeta_2)
10          Propagate(\zeta_3, \zeta_3)
11          insert (\zeta_1, \zeta_2, \zeta_3) in Callers
12          for each (\zeta_3, \zeta_4) in Summary Update(\zeta_1, \zeta_2, \zeta_3, \zeta_4)
13      case \zeta_2 of Exit-CEval
14        if \zeta_1 = \tilde{I}(pr) then
15          Final(\zeta_2)
16        else
17          insert (\zeta_1, \zeta_2) in Summary
18          for each (\zeta_3, \zeta_4, \zeta_1) in Callers Update(\zeta_3, \zeta_4, \zeta_1, \zeta_2)
19          for each (\zeta_3, \zeta_4, \zeta_1) in TCallers Propagate(\zeta_3, \zeta_2)
20      case \zeta_2 of Exit-TC
21        for each \zeta_3 in succ(\zeta_2)
22          Propagate(\zeta_3, \zeta_3)
23          insert (\zeta_1, \zeta_2, \zeta_3) in TCallers
24          for each (\zeta_3, \zeta_4) in Summary Propagate(\zeta_1, \zeta_4)
25
26  Propagate(\zeta_1, \zeta_2) ≜
27  if (\zeta_1, \zeta_2) not in Seen then insert (\zeta_1, \zeta_2) in Seen and W
28
29  Update(\zeta_1, \zeta_2, \zeta_3, \zeta_4) ≜
30  \zeta_1 of the form ([(\lambda_{l_1}(u_1 k_1) call_1)], \hat{d}_1, h_1)
31  \zeta_2 of the form ([(\lambda_{l_2}(u_2 k_2) call_2)]^{l_2}], tf_2, h_2)
32  \zeta_3 of the form ([(\lambda_{l_3}(u_3 k_3) call_3)], \hat{d}_3, h_3)
33  \zeta_4 of the form ([(\lambda_{l_4}(u_4 k_4) call_4)], \hat{d}_4, h_4)
34  \hat{d} ← \tilde{A}_u(e_4, \gamma_4, tf_4, h_4)
35  tf ← \begin{cases} tf_2[f \mapsto [(\lambda_{l_3}(u_3 k_3) call_3)]] & S_\gamma(l_2, f) \\ tf_2 & H_\gamma(l_2, f) \vee Lam_\gamma(f) \end{cases}
36  \zeta ← [(\lambda_{\gamma_2}(u_2) call_2)], \hat{d}, tf, h_4)
37  Propagate(\zeta_1, \zeta)
38
39  Final(\zeta) ≜
40  \zeta of the form ([(\lambda_{\gamma_2}(u_2) call_2)], tf, h)
41  insert (halt, \tilde{A}_u(e, \gamma, tf, h), \emptyset, h) in Final

```

Figure 8: CFA2 workset algorithm

$sprout_{(Q, \Gamma, \delta)}(s) = (\Delta E, \Delta H), \text{ where}$ $\Delta E = \{s \xrightarrow{\epsilon} q : s \xrightarrow{\epsilon} q \in \delta\} \cup \{s \xrightarrow{\gamma^+} q : s \xrightarrow{\gamma^+} q \in \delta\}$ $\Delta H = \{s \mapsto q : s \xrightarrow{\epsilon} q \in \delta\}.$
$addPush_{(Q, \Gamma, \delta)}(G, G_e)(s \xrightarrow{\gamma^+} q) = (\Delta E, \Delta H), \text{ where}$ $\Delta E = \{q' \xrightarrow{\gamma^-} q'' : q' \in \overline{G}_e[q] \text{ and } q' \xrightarrow{\gamma^-} q'' \in \delta\}$ $\Delta H = \{s \mapsto q'' : q' \in \overline{G}_e[q] \text{ and } q' \xrightarrow{\gamma^-} q'' \in \delta\}.$
$addEmpty_{(Q, \Gamma, \delta)}(G, G_e)(s'' \mapsto s''') = (\Delta E, \Delta H), \text{ where}$ $\Delta E = \{s''' \xrightarrow{\gamma^-} q : s' \in \overline{G}_e[s''] \text{ and } s''' \in \overline{G}_e[s'''] \text{ and } s \xrightarrow{\gamma^+} s' \in G\}$ $\Delta H = \{s \mapsto q : s' \in \overline{G}_e[s''] \text{ and } s''' \in \overline{G}_e[s'''] \text{ and } s \xrightarrow{\gamma^+} s' \in G\}$ $\cup \{s' \mapsto s''' : s' \in \overline{G}_e[s'']\}$ $\cup \{s'' \mapsto s''' : s''' \in \overline{G}_e[s''']\}$ $\cup \{s' \mapsto s''' : s' \in \overline{G}_e[s''] \text{ and } s''' \in \overline{G}_e[s''']\}.$
$G' = (S \cup \Delta S, \Gamma, E', q_0)$ $G'_e = (S', H')$ $\Delta G' = (\Delta S' - S', \Delta E' - E').$

Figure 3. The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {(I(pr), I(pr))}
03  while W ≠ ∅
04    remove (ξ1, ξ2) from W
05    switch ξ2
06      case ξ2 of Entry, CApply, Inner-CEval
07        for each ξ3 in succ(ξ2) Propagate(ξ1, ξ3)
08      case ξ2 of Call
09        for each ξ3 in succ(ξ2)
10          Propagate(ξ3, ξ3)
11          insert (ξ1, ξ2, ξ3) in Callers
12          for each (ξ3, ξ4) in Summary Update(ξ1, ξ2, ξ3, ξ4)
13      case ξ2 of Exit-CEval
14        if ξ1 = I(pr) then
15          Final(ξ2)
16        else
17          insert (ξ1, ξ2) in Summary
18          for each (ξ3, ξ4, ξ1) in Callers Update(ξ3, ξ4, ξ1, ξ2)
19          for each (ξ3, ξ4, ξ1) in TCallers Propagate(ξ3, ξ2)
20      case ξ2 of Exit-TC
21        for each ξ3 in succ(ξ2)
22          Propagate(ξ3, ξ3)
23          insert (ξ1, ξ2, ξ3) in TCallers
24          for each (ξ3, ξ4) in Summary Propagate(ξ1, ξ4)
25  Propagate(ξ1, ξ2) ≜
    if (ξ1, ξ2) not in Seen then insert (ξ1, ξ2) in Seen and W
26  Update(ξ1, ξ2, ξ3, ξ4) ≜
    ξ1 of the form ([λl1(u1 k1) call1]), d1, h1)
    ξ2 of the form ([f e2 (λγ2(u2) call2)l2], tf2, h2)
    ξ3 of the form ([λl3(u3 k3) call3]), d3, h2)
    ξ4 of the form ([k4 e4]l4], tf4, h4)
    d ← Au(e4, γ4, tf4, h4)
    tf ← { tf2[f ↦ [[λl3(u3 k3) call3]]] Sγ(l2, f)
          { tf2 Hγ(l2, f) ∨ Lamγ(f)
    ζ ← ([λγ2(u2) call2]), d, tf, h4)
    Propagate(ξ1, ζ)
27  Final(ξ) ≜
    ξ of the form [(k e)γ], tf, h)
    insert (halt, Au(e, γ, tf, h), ∅, h) in Final

```

Figure 8: CFA2 workset algorithm

$$\begin{aligned}
 & \hat{f}((\hat{P}, \hat{E}), \hat{H}, \hat{\sigma}) = ((\hat{P}', \hat{E}'), \hat{H}', \hat{\sigma}'), \text{ where} \\
 \hat{T}_+ &= \left\{ (\hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{\hat{\phi}_+} (\hat{\psi}', \hat{\sigma}') \right\} \\
 \hat{T}_e &= \left\{ (\hat{\psi} \xrightarrow{e} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{e} (\hat{\psi}', \hat{\sigma}') \right\} \\
 \hat{T}_- &= \left\{ (\hat{\psi}'' \xrightarrow{\hat{\phi}_-} \hat{\psi}''', \hat{\sigma}') : \hat{\psi}'' \xrightarrow{\hat{\phi}_-} (\hat{\psi}''', \hat{\sigma}') \text{ and} \right. \\
 & \quad \left. \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{E} \text{ and} \right. \\
 & \quad \left. \hat{\psi}' \mapsto \hat{\psi}'' \in \hat{H} \right\} \\
 \hat{T}' &= \hat{T}_+ \cup \hat{T}_e \cup \hat{T}_- \\
 \hat{E}' &= \left\{ \hat{e} : (\hat{e}, \cdot) \in \hat{T}' \right\} \\
 \hat{\sigma}'' &= \bigsqcup \left\{ \hat{\sigma}' : (\cdot, \hat{\sigma}') \in \hat{T}' \right\} \\
 \hat{H}_e &= \left\{ \hat{\psi} \mapsto \hat{\psi}' : \hat{\psi} \mapsto \hat{\psi}' \in \hat{H} \text{ and } \hat{\psi}' \mapsto \hat{\psi}'' \in \hat{H} \right\} \\
 \hat{H}_{+-} &= \left\{ \hat{\psi} \mapsto \hat{\psi}''' : \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{E} \text{ and } \hat{\psi}' \mapsto \hat{\psi}'' \in \hat{H} \right. \\
 & \quad \left. \text{and } \hat{\psi}'' \xrightarrow{\hat{\phi}_-} \hat{\psi}''' \in \hat{E} \right\} \\
 \hat{H}' &= \hat{H}_e \cup \hat{H}_{+-} \\
 \hat{P}' &= \hat{P} \cup \left\{ \hat{\psi}' : \hat{\psi} \xrightarrow{g} \hat{\psi}' \right\}. \\
 G' &= (S \cup \Delta S, \Gamma, E', g_0) \\
 G'_e &= (S', H') \\
 \Delta G' &= (\Delta S' - S', \Delta E' - E').
 \end{aligned}$$

Figure 3. The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {(I(pr), I(pr))}
03  while W ≠ ∅
04    remove (ξ1, ξ2) from W
05    switch ξ2
06      case ξ2 of Entry, CApply, Inner-CEval
07        for each ξ3 in succ(ξ2) Propagate(ξ1, ξ3)
08      case ξ2 of Call
09        for each ξ3 in succ(ξ2)
10          Propagate(ξ3, ξ3)
11          insert (ξ1, ξ2, ξ3) in Callers
12          for each (ξ3, ξ4) in Summary Update(ξ1, ξ2, ξ3, ξ4)
13      case ξ2 of Exit-CEval
14        if ξ1 = I(pr) then
15          Final(ξ2)
16        else
17          insert (ξ1, ξ2) in Summary
18          for each (ξ3, ξ4, ξ1) in Callers Update(ξ3, ξ4, ξ1, ξ2)
19          for each (ξ3, ξ4, ξ1) in TCallers Propagate(ξ3, ξ2)
20      case ξ2 of Exit-TC
21        for each ξ3 in succ(ξ2)
22          Propagate(ξ3, ξ3)
23          insert (ξ1, ξ2, ξ3) in TCallers
24          for each (ξ3, ξ4) in Summary Propagate(ξ1, ξ4)
25  Propagate(ξ1, ξ2) ≜
    if (ξ1, ξ2) not in Seen then insert (ξ1, ξ2) in Seen and W
26  Update(ξ1, ξ2, ξ3, ξ4) ≜
    ξ1 of the form ([λl1(u1 k1) call1]), d1, h1)
    ξ2 of the form ([f e2 (λγ2(u2) call2)l2], tf2, h2)
    ξ3 of the form ([λl3(u3 k3) call3]), d3, h2)
    ξ4 of the form ([k4 e4]l4], tf4, h4)
    d ← Au(e4, γ4, tf4, h4)
    tf ← { tf2[f ↦ [[λl3(u3 k3) call3]]] Sγ(l2, f)
          { tf2 Hγ(l2, f) ∨ Lamγ(f)
    ξ ← ([λγ2(u2) call2]), d, tf, h4)
    Propagate(ξ1, ξ)
27  Final(ξ) ≜
    ξ of the form ([k e]γ], tf, h)
    insert (halt, Au(e, γ, tf, h), ∅, h) in Final

```

Figure 8: CFA2 workset algorithm

$$\begin{aligned}
 & \hat{f}((\hat{P}, \hat{E}), \hat{H}, \hat{\sigma}) = ((\hat{P}', \hat{E}'), \hat{H}', \hat{\sigma}'), \text{ where} \\
 \hat{T}_+ &= \left\{ (\hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{\hat{\phi}_+} (\hat{\psi}', \hat{\sigma}') \right\} \\
 \hat{T}_e &= \left\{ (\hat{\psi} \xrightarrow{e} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{e} (\hat{\psi}', \hat{\sigma}') \right\} \\
 \hat{T}_- &= \left\{ (\hat{\psi}'' \xrightarrow{\hat{\phi}_-} \hat{\psi}''', \hat{\sigma}') : \hat{\psi}'' \xrightarrow{\hat{\phi}_-} (\hat{\psi}''', \hat{\sigma}') \text{ and} \right. \\
 & \quad \left. \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{E} \text{ and} \right. \\
 & \quad \left. \hat{\psi}' \mapsto \hat{\psi}'' \in \hat{H} \right\} \\
 \hat{T}' &= \hat{T}_+ \cup \hat{T}_e \cup \hat{T}_- \\
 \hat{E}' &= \left\{ \hat{e} : (\hat{e}, \cdot) \in \hat{T}' \right\} \\
 \hat{\sigma}'' &= \bigsqcup \left\{ \hat{\sigma}' : (\cdot, \hat{\sigma}') \in \hat{T}' \right\} \\
 \hat{H}_e &= \left\{ \hat{\psi} \mapsto \hat{\psi}' : \hat{\psi} \mapsto \hat{\psi}' \in \hat{H} \text{ and } \hat{\psi}' \mapsto \hat{\psi}'' \in \hat{H} \right\} \\
 \hat{H}_{+-} &= \left\{ \hat{\psi} \mapsto \hat{\psi}''' : \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{E} \text{ and } \hat{\psi}' \mapsto \hat{\psi}'' \in \hat{H} \right. \\
 & \quad \left. \text{and } \hat{\psi}'' \xrightarrow{\hat{\phi}_-} \hat{\psi}''' \in \hat{E} \right\} \\
 \hat{H}' &= \hat{H}_e \cup \hat{H}_{+-} \\
 \hat{P}' &= \hat{P} \cup \left\{ \hat{\psi}' : \hat{\psi} \xrightarrow{g} \hat{\psi}' \right\}.
 \end{aligned}$$

$$\begin{aligned}
 & \text{addPop}_{(Q, \Gamma, \delta)}(G, G_e)(s'' \xrightarrow{\gamma^-} q) = (\Delta E, \Delta H), \text{ where} \\
 \Delta E &= \emptyset \text{ and } \Delta H = \left\{ s \mapsto q : s' \in \overline{G}_e[s''] \text{ and } s \xrightarrow{\gamma^+} s' \in G \right\}
 \end{aligned}$$

Figure 3. The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01 Summary, Callers
02 Seen, W ←
03 while W ≠
04   remove
05   switch
06   case of Apply, Inner-CEval
07     for each (ξ2) Propagate(ξ1, ξ3)
08   case of
09     for each (ξ2)
10     Propagate(ξ1, ξ3)
11     insert (ξ1, ξ2, ξ3) in Callers
12     for each (ξ3, ξ4)
13   case of Exit-CEval
14     if ξ1 = I(pr) then
15       Final(ξ2)
16     else
17       insert (ξ1, ξ2) in Summary
18       for each (ξ3, ξ4, ξ1) in Callers
19       for each (ξ3, ξ4, ξ1) in TCallers
20   case ξ2 of Exit-TC
21     for each ξ1 in succ(ξ2)
22       Propagate(ξ1, ξ3)
23     insert (ξ1, ξ2, ξ3) in TCallers
24     for each (ξ3, ξ4) in Summary
25
26 Propagate(ξ1, ξ2) ≜
27   if (ξ1, ξ2) in Seen then
28     Final(ξ2)
29   else
30     Update(ξ1, ξ2, ξ1) ≜
31     ξ1 of the form (λ1 (u1 k1) h1)
32     ξ2 of the form (λ2 (u2 k2) h2)
33     ξ3 of the form (λ3 (u3 k3) h3)
34     ξ4 of the form (λ4 (u4 k4) h4)
35     d ← Au(e4, γ4, h4)
36     tf ← [tf2 | f ↦ [(u1 k1) (u2 k2) (u3 k3) (u4 k4) d]
37           (l2, f)
38           (l1, f) ∨ Lamγ(f)]
39     ζ ← [(u1 call1) (u2 call2) (u3 call3) (u4 call4) d]
40     Propagate(ξ1, ξ2)
41     Final(ξ2)
42     insert (half, (e1, tf, ζ)) in Summary

```

Figure 8: CFA2 workset algorithm

$f((\hat{P}, \hat{E}), \hat{H}, \hat{\sigma}) = ((\hat{P}', \hat{E}'), \hat{H}', \hat{\sigma}')$, where

$\hat{T}_+ = \left\{ (\hat{\psi} \xrightarrow{\hat{\sigma}_+} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{\hat{\sigma}_+} (\hat{\psi}', \hat{\sigma}') \right\}$

$\hat{T}_- = \left\{ (\hat{\psi}'' \xrightarrow{\hat{\sigma}_-} \hat{\psi}', \hat{\sigma}') : \hat{\psi}'' \xrightarrow{\hat{\sigma}_-} (\hat{\psi}', \hat{\sigma}') \text{ and } \hat{\psi}'' \xrightarrow{\hat{\sigma}_+} \hat{\psi}' \in \hat{E} \text{ and } \hat{\psi}'' \xrightarrow{\hat{\sigma}_-} \hat{\psi}' \in \hat{H} \right\}$

$\hat{T}' = \hat{T}_+ \cup \hat{T}_- \cup \hat{T}_c$

$\hat{E}' = \left\{ \hat{e} : (\hat{e}, \cdot) \in \hat{T}' \right\}$

$\hat{H}' = \left\{ \hat{h} : (\hat{h}, \cdot) \in \hat{T}' \right\}$

$\hat{\sigma}' = \left\{ \hat{\psi} \xrightarrow{\hat{\sigma}_+} \hat{\psi}' : \hat{\psi} \xrightarrow{\hat{\sigma}_+} \hat{\psi}' \in \hat{E} \text{ and } \hat{\psi}' \xrightarrow{\hat{\sigma}_-} \hat{\psi}'' \in \hat{H} \right\}$

$\hat{\sigma}' = \left\{ \hat{\psi} \xrightarrow{\hat{\sigma}_-} \hat{\psi}' : \hat{\psi} \xrightarrow{\hat{\sigma}_-} \hat{\psi}' \in \hat{H} \text{ and } \hat{\psi}' \xrightarrow{\hat{\sigma}_+} \hat{\psi}'' \in \hat{E} \right\}$

$\hat{\sigma}' = \left\{ \hat{\psi}' \xrightarrow{\hat{\sigma}_+} \hat{\psi}'' \right\}$

$\text{AddPop}_{(Q, \Gamma, \delta)}(G, G_c)(s'' \xrightarrow{\gamma^-} q) = (\Delta E, \Delta H)$, where

$\Delta E = \emptyset$ and $\Delta H = \left\{ s \mapsto q : s' \in \overline{G}_c[s''] \text{ and } s \xrightarrow{\gamma^+} s' \in G \right\}$

Figure 3. The fixed point of the function $\mathcal{F}^*(M)$ contains the Dyck state graph of the rooted pushdown system M .

Pushdown Higher-Order Flow Analysis

```

01  Summary, Callers, TCallers, Final ← ∅
02  Seen, W ← {(I(pr), I(pr))}
03  while W ≠ ∅
04    remove (ξ1, ξ2) from W
05    switch ξ2
06      case ξ2 of Entry, CApply, Inner-CEval
07        for each ξ3 in succ(ξ2) Propagate(ξ1, ξ3)
08      case ξ2 of Call
09        for each ξ3 in succ(ξ2)
10          Propagate(ξ3, ξ3)
11          insert (ξ1, ξ2, ξ3) in Callers
12          for each (ξ3, ξ4) in Summary Update(ξ1, ξ2, ξ3, ξ4)
13      case ξ2 of Exit-CEval
14        if ξ1 = I(pr) then
15          Final(ξ2)
16        else
17          insert (ξ1, ξ2) in Summary
18          for each (ξ3, ξ4, ξ1) in Callers Update(ξ3, ξ4, ξ1, ξ2)
19          for each (ξ3, ξ4, ξ1) in TCallers Propagate(ξ3, ξ2)
20      case ξ2 of Exit-TC
21        for each ξ3 in succ(ξ2)
22          Propagate(ξ3, ξ3)
23          insert (ξ1, ξ2, ξ3) in TCallers
24          for each (ξ3, ξ4) in Summary Propagate(ξ1, ξ4)
25  Propagate(ξ1, ξ2) ≜
    if (ξ1, ξ2) not in Seen then insert (ξ1, ξ2) in Seen and W
26  Update(ξ1, ξ2, ξ3, ξ4) ≜
    ξ1 of the form ([λl1(u1 k1) call1]), d̂1, h1)
    ξ2 of the form ([f e2 (λγ2(u2) call2)l2], tf2, h2)
    ξ3 of the form ([λl3(u3 k3) call3]), d̂3, h2)
    ξ4 of the form ([k4 e4]l4], tf4, h4)
    d̂ ← Au(e4, γ4, tf4, h4)
    tf ← { tf2[f ↦ [[λl3(u3 k3) call3]]] Sγ(l2, f)
          { tf2 Hγ(l2, f) ∨ Lamγ(f)
    ξ ← ([λγ2(u2) call2], d̂, tf, h4)
    Propagate(ξ1, ξ)
27  Final(ξ) ≜
    ξ of the form ([k e]γ], tf, h)
    insert (halt, Au(e, γ, tf, h), ∅, h) in Final

```

Figure 8: CFA2 workset algorithm

$$\begin{aligned}
 & \hat{f}((\hat{P}, \hat{E}), \hat{H}, \hat{\sigma}) = ((\hat{P}', \hat{E}'), \hat{H}', \hat{\sigma}'), \text{ where} \\
 \hat{T}_+ &= \left\{ (\hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{\hat{\phi}_+} (\hat{\psi}', \hat{\sigma}') \right\} \\
 \hat{T}_e &= \left\{ (\hat{\psi} \xrightarrow{e} \hat{\psi}', \hat{\sigma}') : \hat{\psi} \xrightarrow{e} (\hat{\psi}', \hat{\sigma}') \right\} \\
 \hat{T}_- &= \left\{ (\hat{\psi}'' \xrightarrow{\hat{\phi}_-} \hat{\psi}''', \hat{\sigma}') : \hat{\psi}'' \xrightarrow{\hat{\phi}_-} (\hat{\psi}''', \hat{\sigma}') \text{ and} \right. \\
 & \quad \left. \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{E} \text{ and} \right. \\
 & \quad \left. \hat{\psi}' \xrightarrow{\hat{\phi}_-} \hat{\psi}'' \in \hat{H} \right\} \\
 \hat{T}' &= \hat{T}_+ \cup \hat{T}_e \cup \hat{T}_- \\
 \hat{E}' &= \left\{ \hat{e} : (\hat{e}, \cdot) \in \hat{T}' \right\} \\
 \hat{\sigma}'' &= \bigsqcup \left\{ \hat{\sigma}' : (\cdot, \hat{\sigma}') \in \hat{T}' \right\} \\
 \hat{H}_e &= \left\{ \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' : \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{H} \text{ and } \hat{\psi}' \xrightarrow{\hat{\phi}_-} \hat{\psi}'' \in \hat{H} \right\} \\
 \hat{H}_{+-} &= \left\{ \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}'' : \hat{\psi} \xrightarrow{\hat{\phi}_+} \hat{\psi}' \in \hat{E} \text{ and } \hat{\psi}' \xrightarrow{\hat{\phi}_-} \hat{\psi}'' \in \hat{H} \right. \\
 & \quad \left. \text{and } \hat{\psi}'' \xrightarrow{\hat{\phi}_-} \hat{\psi}''' \in \hat{E} \right\} \\
 \hat{H}' &= \hat{H}_e \cup \hat{H}_{+-} \\
 \hat{P}' &= \hat{P} \cup \left\{ \hat{\psi}' : \hat{\psi} \xrightarrow{g} \hat{\psi}' \right\}.
 \end{aligned}$$

$$\begin{aligned}
 & \text{addPop}_{(Q, \Gamma, \delta)}(G, G_e)(s'' \xrightarrow{\gamma_-} q) = (\Delta E, \Delta H), \text{ where} \\
 \Delta E &= \emptyset \text{ and } \Delta H = \left\{ s \mapsto q : s' \in \overleftarrow{G}_e[s''] \text{ and } s \xrightarrow{\gamma_+} s' \in G \right\}
 \end{aligned}$$

Figure 3: The fixed point of the function $\mathcal{F}'(M)$ contains the Dyck state graph of the rooted pushdown system M .

Analysis is ~~terrible~~ salvagable

- Annoying false positives



- ~~• Takes too long~~



- Hard to implement



- Hard to show correct



- ~~• The good ones inherently first order~~



Deriving Pushdown Analyses

- Start: Concrete machine semantics

Deriving Pushdown Analyses

- Start: Concrete machine semantics
- Simple transform: memoize functions

Deriving Pushdown Analyses

- Start: Concrete machine semantics
- Simple transform: memoize functions
- Simple transform: store functions' calling contexts

Deriving Pushdown Analyses

- Start: Concrete machine semantics
- Simple transform: memoize functions
- Simple transform: store functions' calling contexts
- Simple transform: heap-allocate data

Deriving Pushdown Analyses

- Start: Concrete machine semantics
- Simple transform: memoize functions
- Simple transform: store functions' calling contexts
- Simple transform: heap-allocate data
- Analysis: pointwise-abstraction

Deriving Pushdown Analyses

- Start: Concrete machine semantics
- Simple transform: memoize functions
- Simple transform: store functions' calling contexts
- Simple transform: heap-allocate data
- Analysis: pointwise-abstraction

Gives semantic account of summaries

Analysis is ~~terrible~~ salvagable

- Annoying false positives



- ~~• Takes too long~~



- ~~• Hard to implement~~



- ~~• Hard to show correct~~



- ~~• The good ones inherently first order~~



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

Environment:

id ...
app ...

Memo

Contexts

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

Environment:

```
id ...
f id
y 1
```

Memo

Contexts

```
<app id 1> (let* (... [n1 •] ...) ...)
```

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

Environment:

x 1

Memo

Contexts

⟨app id 1⟩ (let* (... [n1 •] ...) ...)

⟨id 1⟩ (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))
```

Environment:

Memo

⟨id 1⟩ 1

Contexts

⟨app id 1⟩ (let* (... [n1 •] ...) ...)

⟨id 1⟩ (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))

```

Environment:

Memo

```

⟨id 1⟩      1
⟨app id 1⟩  1

```

Contexts

```

⟨app id 1⟩ (let* (... [n1 •] ...) ...)
⟨id 1⟩    (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))

```

Environment:

```

id   ...
app  ...
n1   1

```

Memo

```

⟨id 1⟩      1
⟨app id 1⟩  1

```

Contexts

```

⟨app id 1⟩ (let* (... [n1 •] ...) ...)
⟨id 1⟩     (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))

```

Environment:

```

id ...
f  id
y  2

```

Memo

```

⟨id 1⟩      1
⟨app id 1⟩  1

```

Contexts

```

⟨app id 1⟩ (let* (... [n1 •] ...) ...)
⟨id 1⟩     (let* (... [app (λ (f y) •)] ...) ...)
⟨app id 2⟩ (let* (... [n2 •]) ...)

```



```

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

```

Environment:

x 2

Memo

<id 1> 1

<app id 1> 1

Contexts

<app id 1> (let* (... [n1 •] ...) ...)

<id 1> (let* (... [app (λ (f y) •)] ...) ...)

<app id 2> (let* (... [n2 •]) ...)

<id 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))

```

Environment:

Memo

```

⟨id 1⟩      1
⟨app id 1⟩  1
⟨id 2⟩      2

```

Contexts

```

⟨app id 1⟩ (let* (... [n1 •] ...) ...)
⟨id 1⟩     (let* (... [app (λ (f y) •)] ...) ...)
⟨app id 2⟩ (let* (... [n2 •] ...) ...)
⟨id 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))

```

Environment:

Memo

```

<id 1>      1
<app id 1>  1
<id 2>      2
<app id 2>  2

```

Contexts

```

<app id 1> (let* (... [n1 •] ...) ...)
<id 1>     (let* (... [app (λ (f y) •)] ...) ...)
<app id 2> (let* (... [n2 •]) ...)
<id 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))

```

Environment:

```

id    ...
app   ...
n1    1
n2    2

```

Memo

```

⟨id 1⟩      1
⟨app id 1⟩  1
⟨id 2⟩      2
⟨app id 2⟩  2

```

Contexts

```

⟨app id 1⟩ (let* (... [n1 •] ...) ...)
⟨id 1⟩     (let* (... [app (λ (f y) •)] ...) ...)
⟨app id 2⟩ (let* (... [n2 •] ...) ...)
⟨id 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_0

Memo

Start Store: σ_0

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	id
y	1

Memo

Start Store: σ_1

Contexts

$\langle \text{app } \sigma_1 \rangle \left(\left(\text{let}^* \left(\dots [n1 \bullet] \dots \right) \dots \right) \sigma_0 \right)$

```

(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	id
y	1
x	1

Memo

Start Store: σ_2

Contexts

$\langle \text{app } \sigma_1 \rangle$ $((\text{let}^* (\dots [n1 \bullet] \dots) \dots) \sigma_0)$

$\langle \text{id } \sigma_2 \rangle$ $((\text{let}^* (\dots [\text{app } (\lambda (f y) \bullet)] \dots) \dots) \sigma_1)$

```

(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 id

y 1

x 1

Memo

$\langle id \ \sigma_2 \rangle$ (1 σ_2)

Start Store: σ_2

Contexts

$\langle app \ \sigma_1 \ \rangle$ ((let* (... [n1 •] ...) ...) σ_0)

$\langle id \ \sigma_2 \ \rangle$ ((let* (... [app (λ (f y) •)] ...) ...) σ_1)


```

(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 id

y 1

x 1

Memo

$\langle \text{id } \sigma_2 \rangle$ (1 σ_2)

$\langle \text{app } \sigma_1 \rangle$ (1 σ_2)

Start Store: σ_1

Contexts

$\langle \text{app } \sigma_1 \rangle$ ((let* (... [n1 •] ...) ...) σ_0)

$\langle \text{id } \sigma_2 \rangle$ ((let* (... [app (λ (f y) •)] ...) ...) σ_1)

```

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

```

Store: σ_3

<i>f</i>	<i>id</i>
<i>y</i>	1
<i>x</i>	1
<i>n1</i>	1

Memo

$\langle \text{id } \sigma_2 \rangle$ (**1** σ_2)

$\langle \text{app } \sigma_1 \rangle$ (**1** σ_2)

Start Store: σ_0

Contexts

$\langle \text{app } \sigma_1 \rangle$ ((let* (... [n1 •] ...) ...) σ_0)

$\langle \text{id } \sigma_2 \rangle$ ((let* (... [app (λ (f y) •)] ...) ...) σ_1)

```

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

```

Memo

```

⟨id σ2⟩ (1 σ2)
⟨app σ1⟩ (1 σ2)

```

Contexts

```

⟨app σ1⟩ ((let* (... [n1 •] ...) ...) σ0)
⟨id σ2⟩ ((let* (... [app (λ (f y) •)] ...) ...) σ1)
⟨app σ4⟩ ((let* (... [n2 •]) ...) σ0)

```

Store: σ₄

<i>f</i>	<i>id</i>
<i>y</i>	(1 2)
<i>x</i>	1
<i>n1</i>	1

Start Store: σ₄

```

(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	id
y	$(1\ 2)$
x	$(1\ 2)$
$n1$	1

Memo

```

⟨id  $\sigma_2$ ⟩   (1  $\sigma_2$ )
⟨app  $\sigma_1$ ⟩ (1  $\sigma_2$ )

```

Start Store: σ_5

Contexts

```

⟨app  $\sigma_1$ ⟩ ((let* (... [n1 •] ...) ...)  $\sigma_0$ )
⟨id  $\sigma_2$ ⟩  ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_1$ )
⟨app  $\sigma_4$ ⟩ ((let* (... [n2 •]) ...)  $\sigma_0$ )
⟨id  $\sigma_5$ ⟩ ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_4$ )

```

```

(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   id
y   (1 2)
x   (1 2)
n1  1

```

Memo

```

⟨id  $\sigma_2$ ⟩   (1  $\sigma_2$ )
⟨app  $\sigma_1$ ⟩ (1  $\sigma_2$ )
⟨id  $\sigma_5$ ⟩  (2  $\sigma_5$ )

```

Start Store: σ_5

Contexts

```

⟨app  $\sigma_1$ ⟩ ((let* (... [n1 •] ...) ...)  $\sigma_0$ )
⟨id  $\sigma_2$ ⟩  ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_1$ )
⟨app  $\sigma_4$ ⟩ ((let* (... [n2 •]) ...)  $\sigma_0$ )
⟨id  $\sigma_5$ ⟩ ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_4$ )

```

```

(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   id
y   (1 2)
x   (1 2)
n1  1

```

Memo

```

⟨id  $\sigma_2$ ⟩   (1  $\sigma_2$ )
⟨app  $\sigma_1$ ⟩ (1  $\sigma_2$ )
⟨id  $\sigma_5$ ⟩   (2  $\sigma_5$ )
⟨app  $\sigma_4$ ⟩ (2  $\sigma_5$ )

```

Start Store: σ_4

Contexts

```

⟨app  $\sigma_1$ ⟩ ((let* (... [n1 •] ...) ...)  $\sigma_0$ )
⟨id  $\sigma_2$ ⟩  ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_1$ )
⟨app  $\sigma_4$ ⟩ ((let* (... [n2 •]) ...)  $\sigma_0$ )
⟨id  $\sigma_5$ ⟩  ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_4$ )

```

```

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

```

Memo

```

⟨id σ2⟩   (1 σ2)
⟨app σ1⟩  (1 σ2)
⟨id σ5⟩   (2 σ5)
⟨app σ4⟩  (2 σ5)

```

Contexts

```

⟨app σ1⟩ ((let* (... [n1 •] ...) ...) σ0)
⟨id σ2⟩  ((let* (... [app (λ (f y) •)] ...) ...) σ1)
⟨app σ4⟩ ((let* (... [n2 •]) ...) σ0)
⟨id σ5⟩  ((let* (... [app (λ (f y) •)] ...) ...) σ4)

```

Store: σ₆

<i>f</i>	<i>id</i>
<i>y</i>	(1 2)
<i>x</i>	(1 2)
<i>n1</i>	1
<i>n2</i>	(1 2)

Start Store: σ₀

Heaps are imprecise

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

Memo

```
<id σ2> (1 σ2)
<app σ1> (1 σ2)
<id σ5> (2 σ5)
<app σ4> (2 σ5)
```

Contexts

```
<app σ1> ((let* (... [n1 •] ...) ...) σ0)
<id σ2> ((let* (... [app (λ (f y) •)] ...) ...) σ1)
<app σ4> ((let* (... [n2 •] ...) ...) σ0)
<id σ5> ((let* (... [app (λ (f y) •)] ...) ...) σ4)
```

Store: σ₆

<i>f</i>	<i>id</i>
<i>y</i>	(1 2)
<i>x</i>	(1 2)
<i>n1</i>	1
<i>n2</i>	(1 2)

Start Store: σ₀

Fixed[Vardoulakis & Shivers 10]

Local storage (stack frames)

```
(define (foldr f b lst)
  (letrec
    ([loop
     (λ (lst)
      (cond [(empty? lst) b]
            [else (cons (f (car lst))
                        (loop (cdr lst)))]))]
    (loop lst)))
```

```
(define (foldr f b lst)
  (letrec
    ([loop
     (λ (lst)
      (cond [(empty? lst) b]
            [else (cons (f (car lst))
                        (loop (cdr lst)))]))]
    (loop lst)))
```

```
(define (foldr f b lst)
  (letrec
    ([loop
     (λ (lst)
      (cond [(empty? lst) b]
            [else (cons (f (car lst))
                        (loop (cdr lst)))]))]
    (loop lst)))
```

Fixed

Instrument your escape analysis

Escape Analysis

Which addresses outlive their creator's context?

Escape Analysis

Which addresses outlive their creator's context?

We introduce three components:

- Escaped addresses \mathcal{E}
- Addresses owned by current function \mathcal{O}
- The random access stack Ξ

Escape Analysis

Which addresses outlive their creator's context?

We introduce three components:

- Escaped addresses \mathcal{E}
- Addresses owned by current function \mathcal{O}
- The random access stack Ξ

Lookup is now defined with

$$\mathcal{L}(a, \sigma, \Xi, \mathcal{E}) = a \in \mathcal{E} \rightarrow \sigma(a), \Xi(a)$$

$$\langle (x, \rho), \sigma, \Xi, \mathbf{K} \rangle \mapsto \langle v, \sigma, \Xi, \mathbf{K} \rangle$$

where $v \in \mathcal{L}(\rho(x), \sigma, \Xi, \mathcal{E})$

$$\langle (x, \rho), \sigma, \Xi, \mathbf{k} \rangle \mapsto \langle v, \sigma, \Xi, \mathbf{k} \rangle$$

where $v \in \mathcal{L}(\rho(x), \sigma, \Xi, \mathcal{E})$

$$\langle v, \sigma, \Xi, \mathbf{fn}((\lambda x.e, \rho), \mathbf{k}) \rangle \mapsto \langle c, \sigma_{next}, \Xi[a \mapsto v], []^c \rangle$$

where $c = (e, \rho[x \mapsto a])$

$$\sigma_{next} = \sigma[a \mapsto v]$$

$$\mathcal{E}' = \mathcal{E} \cap \mathcal{R}(\mathcal{A}(c), \sigma_{next})$$

$$\langle (x, \rho), \sigma, \Xi, \mathbf{k} \rangle \mapsto \langle v, \sigma, \Xi, \mathbf{k} \rangle$$

where $v \in \mathcal{L}(\rho(x), \sigma, \Xi, \mathcal{E})$

$$\langle v, \sigma, \Xi, \text{fn}((\lambda x.e, \rho), \mathbf{k}) \rangle \mapsto \langle c, \sigma_{next}, \Xi[a \mapsto v], []^c \rangle$$

where $c = (e, \rho[x \mapsto a])$

$$\sigma_{next} = \sigma[a \mapsto v]$$

$$\mathcal{E}' = \mathcal{E} \cap \mathcal{R}(\mathcal{A}(c), \sigma_{next})$$

$$\mathcal{O}' = \{a\}$$

$$\langle (x, \rho), \sigma, \Xi, \kappa \rangle \mapsto \langle v, \sigma, \Xi, \kappa \rangle$$

where $v \in \mathcal{L}(\rho(x), \sigma, \Xi, \mathcal{E})$

$$\langle v, \sigma, \Xi, \text{fn}((\lambda x.e, \rho), \kappa) \rangle \mapsto \langle c, \sigma_{next}, \Xi[a \mapsto v], []^c \rangle$$

where $c = (e, \rho[x \mapsto a])$

$$\sigma_{next} = \sigma[a \mapsto v]$$

$$\mathcal{E}' = \mathcal{E} \cap \mathcal{R}(\mathcal{A}(c), \sigma_{next})$$

$$\mathcal{O}' = \{a\}$$

$$L' = L \sqcup [(c, \sigma_{next}) \mapsto (\kappa, \sigma_{cur}, \Xi, \mathcal{E}, \mathcal{O})]$$

$$\langle v, \sigma, \Xi, []^c \rangle \mapsto \langle v, \sigma, \Xi', \kappa \rangle$$

if $(\kappa, \sigma_{next}, \Xi', \mathcal{E}', \emptyset) \in L(c, \sigma_{cur})$

$$M' = M[(c, \sigma_{cur}) \mapsto (v, \sigma)]$$

$$\mathcal{E}'' = \mathcal{E}' \cup ((\mathcal{E} \cup \emptyset) \cap \mathcal{R}(\mathcal{A}(v), \sigma))$$

Analysis is ~~terrible~~ salvagable

~~• Annoying false positives~~



• Takes too long



~~• Hard to implement~~



~~• Hard to show correct~~



~~• The good ones inherently first order~~



We're not done yet

Too context-sensitive

```
(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 id
y (1 2)
x (1 2)
n1 1
n2 (1 2)
```

Start Store: σ_0

Memo

```
<id  $\sigma_2$ > (1  $\sigma_2$ )
<app  $\sigma_1$ > (1  $\sigma_2$ )
<id  $\sigma_5$ > (2  $\sigma_5$ )
<app  $\sigma_4$ > (2  $\sigma_5$ )
```

Contexts

```
<app  $\sigma_1$ > ((let* (... [n1 •] ...) ...)  $\sigma_0$ )
<id  $\sigma_2$ > ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_1$ )
<app  $\sigma_4$ > ((let* (... [n2 •] ...) ...)  $\sigma_0$ )
<id  $\sigma_5$ > ((let* (... [app (λ (f y) •)] ...) ...)  $\sigma_4$ )
```


First-Order Influence

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

Today's talk

?

First-Order Influence

First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

(Oh et al. 12) CFG + lattice + bypassing

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

Today's talk

?

Higher-order

(Jones & Muchnick 82) Program + set constraints

(Shivers 91) CPS + set constraints

(Might 07) CPS + abstract small step semantics

(Might & Van Horn 10) Derived from concrete

(Vardoulakis & Shivers 10) CPS + summaries

(Earl et al. 10) ANF + Dyck state graph

Today's talk

?

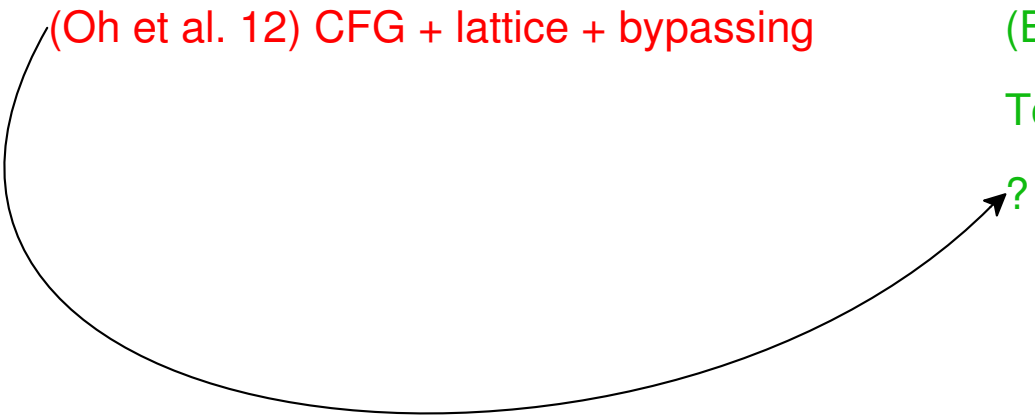
First-order

(Hecht 77) CFG + lattice

(Sharir & Pnueli 81) CFG + lattice + summaries

(Reps 95) PDS + idempotent semiring

(Oh et al. 12) CFG + lattice + bypassing



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
- Precise analyses: Stack allocation \implies precision

To Conclude

- Design: Model abstract mechanisms concretely
- Pushdown: Memo and local continuation tables
- Precise analyses: Stack allocation \implies precision
- (Not shown) works for control operators / GC

To Conclude

- Design: Model abstract mechanisms concretely
- Pushdown: Memo and local continuation tables
- Precise analyses: Stack allocation \implies precision
- (Not shown) works for control operators / GC
- Future work: sparse analysis via time travel

To Conclude

- Design: Model abstract mechanisms concretely
- Pushdown: Memo and local continuation tables
- Precise analyses: Stack allocation \implies precision
- (Not shown) works for control operators / GC
- Future work: sparse analysis via time travel

Thank you