Pushdown Control-Flow Analysis of Higher Order Programs

Christopher Earl[1] Matthew Might[1] David Van Horn[2]

[1]University of Utah
{cwearl,might}@cs.utah.edu

[2]Northeastern University
dvanhorn@ccs.neu.edu

August 21, 2010

Who uses function (calls)?

Who uses function (calls)?

Pushdown control-flow analysis models function calls precisely.

Simple example of merging return-points

```
(let* ((id (lambda (x) x))
       (a (id 3))
       (b (id 4)))
    a)
```

The big picture

Classical control-flow analysis is not precise enough.

The big picture

Classical control-flow analysis is not precise enough.

Pushdown control-flow analysis has better precision.

The big picture

Classical control-flow analysis is not precise enough.

Pushdown control-flow analysis has better precision.

We generalize k-CFA to a pushdown control-flow analysis.

The big picture

Classical control-flow analysis is not precise enough.

Pushdown control-flow analysis has better precision.

We generalize k-CFA to a pushdown control-flow analysis.

Our approach has several advantages:
Direct-style
Polyvariant
Polynomial

Control-flow analysis $<$ pushdown control-flow analysis

Expressiveness of k-CFA = NFA

Control-flow analysis $<$ pushdown control-flow analysis

Expressiveness of k-CFA = NFA

Expressiveness of PDCFA = PDA

Our approach

Target language/stack behavior

$(\texttt{let}\ ((x\ e_1))\ e_2) \quad \Longrightarrow \quad$ Push frame $(x, e_2, \ldots)$ onto stack.

Target language/stack behavior

$$\texttt{(let}\ ((x\ e_1))\ e_2) \quad \Longrightarrow \quad \text{Push frame } (x, e_2, \ldots) \text{ onto stack.}$$

$$a \quad \Longrightarrow \quad \text{Pop top of stack.}$$

Target language/stack behavior

$$
\begin{array}{rcl}
(\texttt{let}\ ((x\ e_1))\ e_2) & \Longrightarrow & \text{Push frame } (x, e_2, \ldots) \text{ onto stack.} \\
a & \Longrightarrow & \text{Pop top of stack.} \\
(f\ a) & \Longrightarrow & \text{Stack no-op.}
\end{array}
$$

Concrete Semantics

A CESK machine.

Concrete Semantics

A CESK machine.

Configuration = State $\times$ Stack

Concrete Semantics

A CESK machine.

Configuration = State $\times$ Stack

State = Expression $\times$ Environment $\times$ Store

Abstract Semantics

Abstracted environment $\Longrightarrow$

Abstract Semantics

Abstracted environment $\implies$ environments = finite

Abstract Semantics

Abstracted environment $\implies$ environments = finite

Abstracted store $\implies$

Abstract Semantics

$$\text{Abstracted environment} \implies \text{environments = finite}$$

$$\text{Abstracted store} \implies \text{stores = finite}$$

Abstract Semantics

Abstracted environment $\Longrightarrow$ environments = finite

Abstracted store $\Longrightarrow$ stores = finite

Abstracted state $\Longrightarrow$

Abstract Semantics

Abstracted environment $\Longrightarrow$ environments = finite

Abstracted store $\Longrightarrow$ stores = finite

Abstracted state $\Longrightarrow$ states = finite

Size of the abstract configuration-space

Using the stack $\Longrightarrow$

Size of the abstract configuration-space

Using the stack $\Longrightarrow$ configuration-space = infinite

Size of the abstract configuration-space

Using the stack $\implies$ configuration-space = infinite

The configuration-space cannot be explicitly searched.

Size of the abstract state-space

State-space = finite
Always.

Finite model of pushdown control-flow analysis

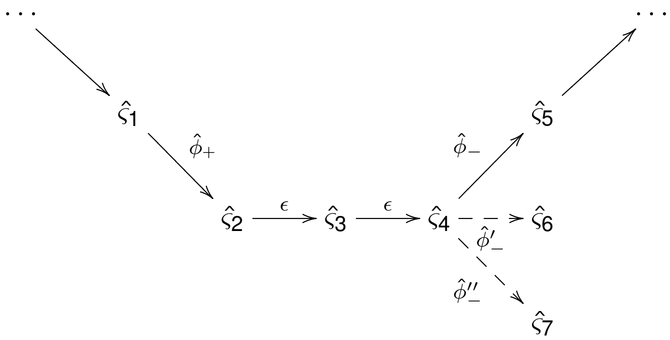
$$
\begin{tikzcd}
\cdots \arrow[dr] & & & & & & \cdots \\
& \hat{\varsigma}_1 \arrow[dr, "\hat{\phi}_+"] & & & & \hat{\varsigma}_5 \arrow[ur] & \\
& & \hat{\varsigma}_2 \arrow[r, "\epsilon"] & \hat{\varsigma}_3 \arrow[r, "\epsilon"] & \hat{\varsigma}_4 \arrow[ur, "\hat{\phi}_-"] & &
\end{tikzcd}
$$

Finite model of pushdown control-flow analysis

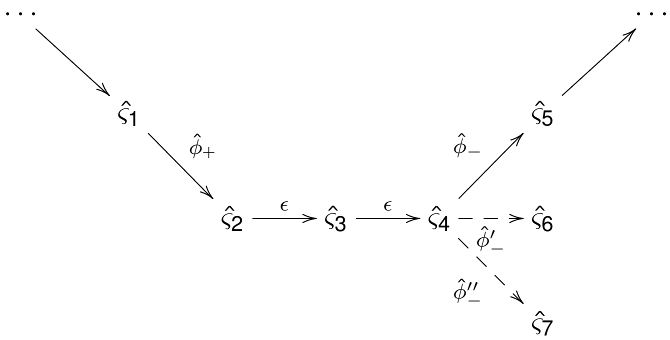
$$\begin{tikzcd}
\cdots \arrow[dr] & & & & & & \cdots \\
& \hat{\varsigma}_1 \arrow[dr, "\hat{\phi}_+"] & & & & \hat{\varsigma}_5 \arrow[ur] & \\
& & \hat{\varsigma}_2 \arrow[r, "\epsilon"] & \hat{\varsigma}_3 \arrow[r, "\epsilon"] & \hat{\varsigma}_4 \arrow[ur, "\hat{\phi}_-"] & &
\end{tikzcd}$$

This representation is a PDA.

While finite, this naive PDA is inefficient:

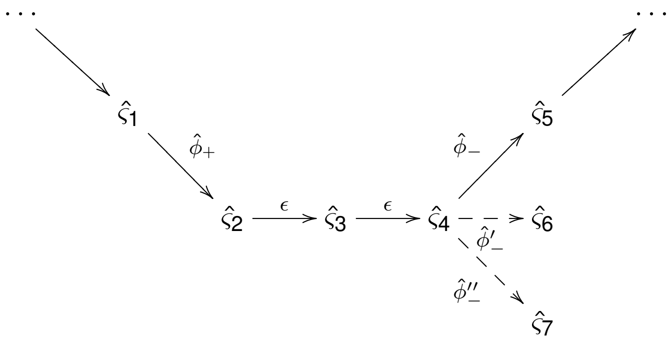


While finite, this naive PDA is inefficient:



(Provably) unreachable configurations/states are included.

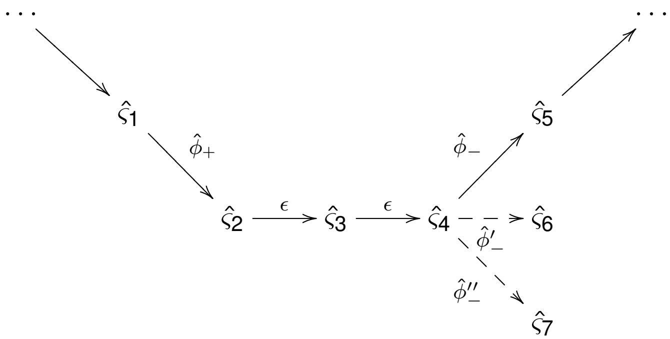
While finite, this naive PDA is inefficient:



(Provably) unreachable configurations/states are included.

Legal path from initial configuration/state $\implies$

While finite, this naive PDA is inefficient:



(Provably) unreachable configurations/states are included.

Legal path from initial configuration/state $\implies$ reachable

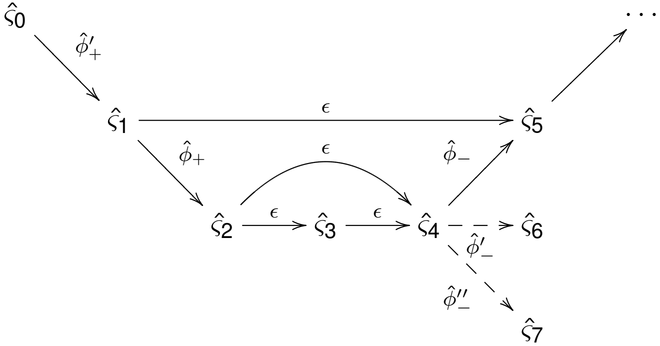
Shortcut edges: finding the top of the stack

$$
\begin{tikzcd}
\cdots \arrow[dr] & & & & & & \cdots \\
 & \hat{\varsigma}_1 \arrow[dr, "\hat{\phi}_+"] & & & & \hat{\varsigma}_5 \arrow[ur] & \\
 & & \hat{\varsigma}_2 \arrow[r, "\epsilon"] \arrow[rr, bend left, "\epsilon"] & \hat{\varsigma}_3 \arrow[r, "\epsilon"] & \hat{\varsigma}_4 \arrow[ur, "\hat{\phi}_-"] \arrow[r, dashed, "\hat{\phi}'_-"'] \arrow[dr, dashed, "\hat{\phi}''_-"'] & \hat{\varsigma}_6 & \\
 & & & & & \hat{\varsigma}_7 &
\end{tikzcd}
$$

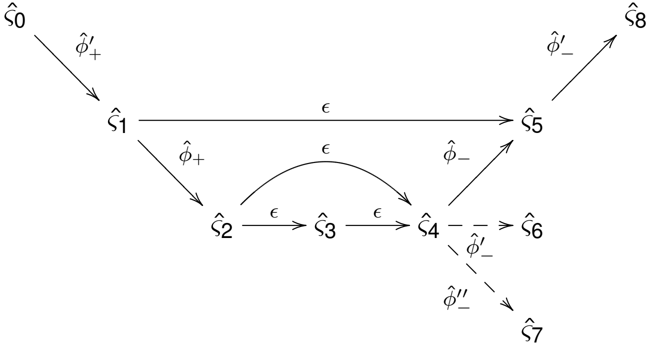
Shortcut edges: finding the top of the stack

$$
\begin{tikzcd}
\cdots \arrow[dr] & & & & & & \cdots \\
& \hat{\varsigma}_1 \arrow[rrrr, "\epsilon"] \arrow[dr, "\hat{\phi}_+"] & & & & \hat{\varsigma}_5 \arrow[ur] & \\
& & \hat{\varsigma}_2 \arrow[r, "\epsilon"] \arrow[rr, bend left, "\epsilon"] & \hat{\varsigma}_3 \arrow[r, "\epsilon"] & \hat{\varsigma}_4 \arrow[ur, "\hat{\phi}_-"] \arrow[r, dashed, "\hat{\phi}'_-"'] \arrow[dr, dashed, "\hat{\phi}''_-"'] & \hat{\varsigma}_6 & \\
& & & & & \hat{\varsigma}_7 &
\end{tikzcd}
$$

Shortcut edges: finding the top of the stack



Shortcut edges: finding the top of the stack



Shortcut edges: finding the top of the stack

$$
\begin{tikzcd}
\hat{\varsigma}_0 \arrow[rrrrrr, "\epsilon"] \arrow[dr, "\hat{\phi}'_+"] & & & & & & \hat{\varsigma}_8 \\
& \hat{\varsigma}_1 \arrow[rrrr, "\epsilon"] \arrow[dr, "\hat{\phi}_+"] & & & & \hat{\varsigma}_5 \arrow[ur, "\hat{\phi}'_-"] & \\
& & \hat{\varsigma}_2 \arrow[r, "\epsilon"] \arrow[rr, bend left, "\epsilon"] & \hat{\varsigma}_3 \arrow[r, "\epsilon"] & \hat{\varsigma}_4 \arrow[ur, "\hat{\phi}_-"] \arrow[r, dashed, "\hat{\phi}'_-"'] \arrow[dr, dashed, "\hat{\phi}''_-"'] & \hat{\varsigma}_6 & \\
& & & & & \hat{\varsigma}_7 &
\end{tikzcd}
$$

Dyck state graphs: a lean PDA representation

$$
\begin{tikzcd}
\hat{\varsigma}_0 \arrow[rrrrrr, "\epsilon"] \arrow[dr, "\hat{\phi}'_+"] & & & & & & \hat{\varsigma}_8 \\
& \hat{\varsigma}_1 \arrow[rrrr, "\epsilon"] \arrow[dr, "\hat{\phi}_+"] & & & & \hat{\varsigma}_5 \arrow[ur, "\hat{\phi}'_-"] & \\
& & \hat{\varsigma}_2 \arrow[r, "\epsilon"] \arrow[rr, bend left, "\epsilon"] & \hat{\varsigma}_3 \arrow[r, "\epsilon"] & \hat{\varsigma}_4 \arrow[ur, "\hat{\phi}_-"] & &
\end{tikzcd}
$$

Only reachable states and configurations are included.

Our contributions

Direct-style

Polyvariant

Polynomial

Direct-style:

Direct-style: by the language (A-Normal Form)

Direct-style: by the language (A-Normal Form)

Polyvariant:

Direct-style: by the language (A-Normal Form)

Polyvariant: the abstract semantics can use a parameter, k, identical to the k in k-CFA

Polynomial: monovariance and store-widening

Standard (infinite) pushdown control-flow analysis:

Configuration = Expression $\times$ Environment $\times$ Store $\times$ Stack

Frame = Variable $\times$ Expression $\times$ Environment

Polynomial: monovariance and store-widening

Dyck state graphs:

State = Expression $\times$ Environment $\times$ Store

Frame = Variable $\times$ Expression $\times$ Environment

Polynomial: monovariance and store-widening

Monovariant Dyck state graphs:

State = Expression $\times$ Store

Frame = Variable $\times$ Expression

Polynomial: monovariance and store-widening

Monovariant Dyck state graphs with store-widening:

State = Expression (with a global store)

Frame = Variable $\times$ Expression

Recap

Pushdown control-flow analysis precisely models the stack.

Recap

Pushdown control-flow analysis precisely models the stack.

Our formulation only explores reachable configurations/states.

Recap

Pushdown control-flow analysis precisely models the stack.

Our formulation only explores reachable configurations/states.

Our formulation works for direct-style programs.

Recap

Pushdown control-flow analysis precisely models the stack.

Our formulation only explores reachable configurations/states.

Our formulation works for direct-style programs.

Our formulation allows for either:

Recap

Pushdown control-flow analysis precisely models the stack.

Our formulation only explores reachable configurations/states.

Our formulation works for direct-style programs.

Our formulation allows for either:

Polyvariance

Recap

Pushdown control-flow analysis precisely models the stack.

Our formulation only explores reachable configurations/states.

Our formulation works for direct-style programs.

Our formulation allows for either:

Polyvariance

Polynomial running-time

Questions?

$O(n^6)$