Resolving and Exploiting the k-CFA Paradox

Matthew Might, Yannis Smaragdakis, and David Van Horn

Plan

- ⋆ What is k-CFA?
- ⋆ The **paradox** of k-CFA
- ⋆ A **resolution**: OO vs. functional k-CFA
- ⋆ An **exploitation**: m-CFA
- ⋆ Evaluation and conclusion

What is control flow analysis?

Answers the question: where does control transfer to at a call?

```
o.m(5);
```

Clearly a kind of points-to analysis.

What is control flow analysis?

Answers the question: where does control transfer to at a call?

```
n(o) { return o.m(5); }
```

What is control flow analysis?

Answers the question: where does control transfer to at a call?

```
n(o) { return o.m(5); }
...
if (P) n(q) else n(r);
```

What is control flow analysis?

Answers the question: where does control transfer to at a call?

```
(f 5)
```

What is control flow analysis?

Answers the question: where does control transfer to at a call?

```
(λ (f) (f 5))
```

What is control flow analysis?

Answers the question: where does control transfer to at a call?

```
((λ (f) (f 5))
  (if P q r))
```

What is control flow analysis?

Answers the question: where does control transfer to at a call?

But this is not easy to decide in a *higher-order* language.

Which languages are higher-order? Almost all of them: Scheme, ML, Haskell, JavaScript, Java, C, C++, Ruby, Python, etc.

A Simple use of Closures

```
; Number → (Number → Number)
(define (make-adder n)
  (λ (x)
    (+ x n)))

(define add5 (make-adder 5))
(define add3 (make-adder 3))

(add5 2) ;=> 7
(add3 7) ;=> 10
```

$$\texttt{(make-adder 5)} = \langle\, \texttt{(}\lambda\texttt{ (x) (+ x n))}, [\texttt{n} \mapsto \texttt{5}] \rangle.$$

A Simple use of Closures

```
class MkAddr {
  makeAdder(n) {
    return new Add {
       add(x) { return x+n; }}}}
```

Non-idiomatic. Anonymous inner class emulates λ*.*

A Simple *non*-use of Closures

```
class Add {
  n;
  Add(n) { this.n = n; }
  add(x) { return x+n; }
}
```

"Closures" emulated by local fields and constructor accepting value of "free" variables.

What is k-CFA?

Approximation in 0CFA

0CFA approximates closures by their code component.

$$
\begin{aligned}
\texttt{(make-adder 5)} &= \langle \texttt{(}\lambda\ \texttt{(x) (+ x n))}, [\texttt{n} \mapsto 5] \rangle \\
&\approx \texttt{(}\lambda\ \texttt{(x) (+ x n))} \\
\texttt{n} &\approx \{5, 3\} \\
\\
\texttt{(add5 2)} &\approx \{7, 5\}
\end{aligned}
$$

Context-insensitive points-to analysis.

What is k-CFA?

k-CFA uses *calling contexts* to increase analysis precision:

$$\texttt{(make-adder 5)}^1 \ \ldots\ \texttt{(make-adder 3)}^2$$

1-CFA:

- $\star$ $\langle$`(λ (x) (+ x n))`$, [\texttt{n} \mapsto 5]\rangle$.
- $\star$ $\langle$`(λ (x) (+ x n))`$, [\texttt{n} \mapsto 3]\rangle$.

$$\texttt{(add5 2)} \ \approx\ \{7\}$$

Call-site context-sensitive points-to analysis.

The Paradox

k-CFA is hard

It did not take long to discover that the basic analysis, for any $k > 0$, was intractably slow for large programs.

Shivers, Higher-order control-flow analysis in retrospect: Lessons learned, lessons abandoned (2004)

What makes k-CFA hard?

Closures.

```
((λ (f) (f 0) (f 1))
 (λ (x) (λ (z) (z x))))
```

1-CFA:

- ⋆ $\langle$`(λ (z) (z x))`$, [\mathtt{x} \mapsto \mathtt{0}]\rangle$.
- ⋆ $\langle$`(λ (z) (z x))`$, [\mathtt{x} \mapsto \mathtt{1}]\rangle$.

2 closures.

What makes k-CFA hard?

$$(\lambda\ (\mathtt{z})\ (\mathtt{z}\ \mathtt{x}_0 \ldots \mathtt{x}_n))$$

1-CFA:

- $[\mathtt{x}_1 \mapsto 0, \ldots, \mathtt{x}_n \mapsto 0]$
- $[\mathtt{x}_1 \mapsto 1, \ldots, \mathtt{x}_n \mapsto 0]$
- $\ldots$
- $[\mathtt{x}_1 \mapsto 0, \ldots, \mathtt{x}_n \mapsto 1]$
- $[\mathtt{x}_1 \mapsto 1, \ldots, \mathtt{x}_n \mapsto 1]$

$O(2^n)$ closures.

k-CFA is complete for EXPTIME (Van Horn and Mairson, '08).

k-CFA is easy

k-CFA of object-oriented programs is in PTIME.

E.g., written in Datalog (Bravenboer and Smaragdakis, ’09).

k-CFA is easy

k-CFA of object-oriented programs is in PTIME.

E.g., written in Datalog (Bravenboer and Smaragdakis, '09).

$$\text{PTIME} \subsetneq \text{EXPTIME}$$

The Resolution

What makes (OO) k-CFA easy?

No closures.

Explicit closures: 1 variable

```
class Add {
  n;
  Add(n) { this.n = n; }
  add(x) { return x+n; }
}

new Add(5);

new Add(2);

new Add(7);
```

Creates 3 "closures" over 1 variable.

Explicit closures: c variables

```
class Add {
  n0; ··· nc;
  Add(n0,...,nc) { this.ni = ni; }
  add(x) { return x···n0···nc; }
}

new Add(p1,...,pc);

new Add(q1,...,qc);

new Add(r1,...,rc);
```

Creates 3 “closures” over c variables.

Only consider 3 “closures”, not 3^c!

Analytic message

"Explicit" closing collapses the value set.

Resolving the Paradox

k-CFA of:

- $\star$ an FP language with closures is EXPTIME.
- $\star$ an OO language without closures is PTIME.

Resolving the Paradox

k-CFA of:

- ⋆ an FP language with closures is EXPTIME.
- ⋆ an OO language without closures is PTIME.
- ⋆ an OO language *with* closures is EXPTIME.
- ⋆ an FP language *without* closures is PTIME.

Resolving the Paradox

k-CFA of:

- $\star$ an FP language with closures is EXPTIME.
- $\star$ an OO language without closures is PTIME.
- $\star$ an OO language *with* closures is EXPTIME.
- $\star$ an FP language *without* closures is PTIME.

But the k-CFA is *the same*.

The Exploitation

The idea: Flat closures

Change the representation of closures:

$$Clo = \mathsf{Lam} \times (\mathsf{Var} \to Addr)$$
$$Clo' = \mathsf{Lam} \times Addr$$

At call sites, copy values of closure into the store.

Essentially lifting λ-lifting into semantics.

Evaluation and conclusion

Speed: Worst-case benchmark

Terms	$k = 1$	$m = 1$	poly.,k=1	k=0
69	ϵ	ϵ	ϵ	ϵ
123	ϵ	ϵ	ϵ	ϵ
231	46 s	ϵ	2 s	ϵ
447	∞	3 s	5 s	2 s
879	∞	48 s	1 m 8 s	15 s
1743	∞	51 m	∞	3 m 48 s

Speed and precision

Prog/Terms	$k = 1$		$m = 1$		poly.,k=1		k=0	
eta / 49	ϵ	7	ϵ	7	ϵ	3	ϵ	3
map / 157	ϵ	8	ϵ	8	ϵ	8	ϵ	6
sat / 223	∞	-	ϵ	12	1s	12	ϵ	12
regex / 1015	4s	25	3s	25	14s	25	2s	25
scm2java / 2318	5s	86	3s	86	3s	79	4s	79
interp / 4289	5s	123	4s	123	9s	123	5s	123
scm2c / 6219	179s	136	143s	136	157s	131	55s	131

Column 1: analysis time, Column 2: inlinings enabled

Benchmark message

m-CFA v. k-CFA: as precise, but faster.

Doggie bag

- k-CFA of functional programs is hard because of closures.
- k-CFA of OO-programs is easy because of no closures.
- m-CFA: similar precision, faster.
- m-CFA is always in PTIME.

The End

Thank you.

Implementation and benchmarks:

`http://www.ucombinator.org/projects/mcfa/`