

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 k -CFA?

What is k -CFA?

CFA is the sound approximation of evaluation at compile-time.

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

$$\left(\left(\lambda (f) (f u)^1 (f v)^2 \right) \left(\lambda (x) \left(\lambda (y) x \right) \right) \right)$$

1-CFA:

- ★ $\langle (\lambda (y) x), [x \mapsto 1] \rangle$ and $[1 \mapsto u]$.
- ★ $\langle (\lambda (y) x), [x \mapsto 2] \rangle$ and $[2 \mapsto v]$.

Concrete semantics: k -CFA

$((f\ e)^\ell, \beta, \sigma, a) \Rightarrow (e', \beta', \sigma', a')$, where

$$\mathcal{E}(f, \beta, \sigma) = \langle (\lambda (x) e'), \beta'' \rangle \quad a' = \ell \cdot a$$

$$\mathcal{E}(e, \beta, \sigma) = d \quad \beta' = \beta''[x \mapsto a']$$

$$\sigma' = \sigma[a' \mapsto d]$$

And:

$$\mathcal{E}(x, \beta, \sigma) = \sigma(\beta(x))$$

$$\mathcal{E}((\lambda (x) e), \beta, \sigma) = \langle (\lambda (x) e), \beta \rangle$$

Abstract semantics: k -CFA

$((f\ e)^\ell, \hat{\beta}, \hat{\sigma}, \hat{a}) \rightsquigarrow (e', \hat{\beta}', \hat{\sigma}', \hat{a}')$, where

$$\begin{aligned}\hat{\mathcal{E}}(f, \hat{\beta}, \hat{\sigma}) &\ni \langle (\lambda\ (x)\ e'), \hat{\beta}'' \rangle & \hat{a}' &= \lfloor \ell \cdot \hat{a} \rfloor_k \\ \hat{\mathcal{E}}(e, \hat{\beta}, \hat{\sigma}) &= \hat{d} & \hat{\beta}' &= \hat{\beta}''[x \mapsto \hat{a}'] \\ & & \hat{\sigma}' &= \hat{\sigma} \sqcup [\hat{a}' \mapsto \hat{d}]\end{aligned}$$

And:

$$\begin{aligned}\hat{\mathcal{E}}(x, \hat{\beta}, \hat{\sigma}) &= \hat{\sigma}(\hat{\beta}(x)) \\ \hat{\mathcal{E}}((\lambda\ (x)\ e), \hat{\beta}, \hat{\sigma}) &= \{ \langle (\lambda\ (x)\ e), \hat{\beta} \rangle \}\end{aligned}$$

The Paradox of k -CFA

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.

$((\lambda (f) (f u)^1 (f v)^2) (\lambda (x) (\lambda (y) x)))$

1-CFA:

- ★ $\langle (\lambda (y) x), [x \mapsto 1] \rangle$ and $[1 \mapsto u]$.
- ★ $\langle (\lambda (y) x), [x \mapsto 2] \rangle$ and $[2 \mapsto v]$.

What makes k -CFA hard?

$$\begin{aligned}
 & ((\lambda (f_1) (f_1 0)^1 (f_1 1)^2) \\
 & (\lambda (x_1) \\
 & \dots \\
 & ((\lambda (f_n) (f_n 0)^{2n-1} (f_n 1)^{2n}) \\
 & (\lambda (x_n) \\
 & (\lambda (z) (z x_1 \dots x_n))) \dots))
 \end{aligned}$$

1-CFA:

- ★ $[x_1 \mapsto 0, \dots, x_n \mapsto 0]$
- ★ $[x_1 \mapsto 1, \dots, x_n \mapsto 0]$
- ★ \dots
- ★ $[x_1 \mapsto 0, \dots, x_n \mapsto 1]$
- ★ $[x_1 \mapsto 1, \dots, x_n \mapsto 1]$

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

k-CFA is easy

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

Implemented in Datalog (Bravenboer and Smaragdakis, '09).

PTIME \subsetneq EXPTIME

OO vs. functional k -CFA

A Resolution:

FP in OO and back again

$A \rightarrow B$

\approx

```
interface Fun<A,B> { B apply(A a); }
```

FP in OO and back again

$((\lambda (f) (f u) (f v)) (\lambda (x) (\lambda (y) x)))$

\approx

```
new Fun() { apply(f) { f.apply(u); f.apply(v); }}  
.apply(new Fun() { apply(x) { new Fun() { apply(y) { x; }}}});
```

\approx

```
class Lam1 { apply(f) { f.apply(u); f.apply(v); }}  
class Lam2 { apply(x) { new Lam3(x); }}  
class Lam3 { Lam3(x){this.x = x}; apply(y) { x; }}  
new Lam1().apply(new Lam2());
```

\approx

$((\lambda (f) (f u) (f v)) (\lambda (x) (\text{let } x' = x \text{ in } (\lambda (y) x'))))$

\approx

$((\lambda (f) (f u) (f v)) (\lambda (x) ((\lambda (x') (\lambda (y) x')) x)))$

An Exploitation: *m*-CFA

The idea: Flat closures

Change the representation of closures:

$$Clo = Lam \times (Var \rightarrow Addr)$$

$$Clo' = Lam \times Addr$$

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

Concrete semantics: m -CFA

$((f\ e)^\ell, a, \sigma) \Rightarrow (e', a', \sigma')$, where

$$\mathcal{E}(f, a, \sigma) = \langle (\lambda (x) e'), a'' \rangle \quad a' = \ell \cdot a$$

$$\mathcal{E}(e, a, \sigma) = d \quad d'_j = \sigma(a'', y_j)$$

$$\{y_1, \dots, y_n\} = fv(e') \setminus \{x\} \quad \sigma' = \sigma[a' \mapsto (d, d'_1, \dots, d'_n)]$$

And:

$$\mathcal{E}(x, a, \sigma) = \sigma(a, x)$$

$$\mathcal{E}((\lambda (x) e), a, \sigma) = \langle (\lambda (x) e), a \rangle$$

Abstract semantics: m -CFA

$((f e)^\ell, \hat{a}, \hat{\sigma}) \Rightarrow (e', \hat{a}', \hat{\sigma}')$, where

$$\hat{\mathcal{E}}(f, \hat{a}, \hat{\sigma}) \ni \langle (\lambda (x) e'), \hat{a}'' \rangle \quad \hat{a}' = \lfloor \ell \cdot \hat{a} \rfloor_m$$

$$\hat{\mathcal{E}}(e, \hat{a}, \hat{\sigma}) = \hat{d} \quad \hat{d}'_j = \hat{\sigma}(\hat{a}'', y_j)$$

$$\{y_1, \dots, y_n\} = fv(e') \setminus \{x\} \quad \hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}' \mapsto (\hat{d}, \hat{d}'_1, \dots, \hat{d}'_n)]$$

And:

$$\hat{\mathcal{E}}(x, \hat{a}, \hat{\sigma}) = \hat{\sigma}(\hat{a}, x)$$

$$\hat{\mathcal{E}}((\lambda (x) e), \hat{a}, \hat{\sigma}) = \{ \langle (\lambda (x) e), \hat{a} \rangle \}$$

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

Doggie bag

- ★ k -CFA of λ -programs is hard because of closures.
- ★ k -CFA of OO-programs is easy because of flat closures.
- ★ m -CFA: similar precision, less cost.
- ★ m -CFA is always in PTIME.



The End

Thank you.

Preprint, implementation, benchmarks:

<http://www.ucombinator.org/projects/mcfa/>