Resolving and Exploiting the k-CFA Paradox

Matthew Might, Yannis Smaragdakis, and David Van Horn

Plan

- $\star$ What is program analysis (aka abstract interpretation)?
- $\star$ What is k-CFA?
- $\star$ The paradox of k-CFA
- $\star$ A resolution: OO vs. functional k-CFA
- $\star$ An exploitation: m-CFA
- $\star$ Evaluation and conclusion

What is abstract interpretation?

What is abstract interpretation?

AI is the sound computable approximation of evaluation.

Useful for:

- ⋆ Safety
- ⋆ Error detection
- ⋆ Optimization
- ⋆ Transformation
- ⋆ Termination
- ⋆ ...

Kinds:

- ⋆ Numerical analysis
- ⋆ Polyhedral analysis
- ⋆ Congruence analysis
- ⋆ Strictness analysis
- ⋆ Control-flow analysis
- ⋆ ...

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure call?

$$(f\ x)$$

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure call?

$$(\lambda\ (f)\ \ (f\ x)\ )$$

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure call?

$$(\,(\lambda\ (f)\ \ (f\ x)\,)$$

$$(\texttt{if}\ P\ \ (\lambda\ (y)\ \ \ldots)\ \ (\lambda\ (z)\ \ \ldots)\,)\,)$$

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure call?

```
o.m(x);
```

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure call?

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

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure call?

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

...

n(P ? q : r);
```

What is control flow analysis?

Control flow analysis answers the question: where does control transfer to at a given procedure 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, Ruby, Python, etc.

Fun. programming in Java

```
class Calculus {

  interface Fun<X,Y> { Y apply(X x); }

  Fun<Fun<Double,Double>,Fun<Double,Double>> deriv =
    new Fun<Fun<Double,Double>,Fun<Double,Double>>() {
      double d = 0.001;
      public Fun<Double,Double> apply(final Fun<Double,Double> f) {
        return new Fun<Double,Double>() {
          public Double apply(Double x) {
            return (f.apply(x + d) - f.apply(x - d)) / (d * 2);}};}};

  Fun<Double,Double> sin = new Fun<Double,Double>() {
    public Double apply(Double x) { return Math.sin(x); }
  };

  Fun<Double,Double> cos = deriv.apply(sin);
}
```

Fun. programming in Java

```
class Calculus {

  interface Fun<X,Y> { Y apply(X x); }

  deriv =
    new Fun<Fun<Double,Double>,Fun<Double,Double>>() {
      double d = 0.001;
      apply(f) {
        return new Fun<Double,Double>() {
          apply(x) {
            return (f.apply(x + d) - f.apply(x - d)) / (d * 2);}};}};

  sin = new Fun<Double,Double>() {
    apply(x) { return Math.sin(x); }
  };

  cos = deriv.apply(sin);
}
```

Fun. programming in Scheme

```
(define (deriv f) ; (Number ⟶ Number) ⟶ (Number ⟶ Number)
  (let ((d 0.001))
    (λ (x) ; Number ⟶ Number
      (/ (- (f (+ x d))
            (f (- x d)))
         (* d 2)))))

(define cos (deriv sin))
(define dbl (deriv sqr))
```

What is k-CFA?

Approximation in 0CFA

0CFA approximates closures by their code component.

$$
\begin{array}{rcl}
\texttt{(deriv sin)} & = & \langle\,\texttt{(}\lambda\ \texttt{(x)}\ \texttt{...f..f...)}, [\texttt{f} \mapsto \texttt{sin}]\rangle \\
 & \approx & \texttt{(}\lambda\ \texttt{(x)}\ \texttt{...f..f...)} \\
\texttt{f} & \approx & \{\texttt{sin}, \texttt{sqr}\} \\
 & & \\
\texttt{(cos pi)} & \approx & \{-0.999, 6.2831, \ldots\}
\end{array}
$$

What is k-CFA?

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

$$\texttt{(deriv sin)}^1 \;\texttt{...}\; \texttt{(deriv sqr)}^2$$

1-CFA:

- $\star$ $\langle$`(λ (x)...f..f...)`$, [\texttt{f} \mapsto 1]\rangle$ and $[1 \mapsto \texttt{sin}]$.
- $\star$ $\langle$`(λ (x)...f..f...)`$, [\texttt{f} \mapsto 2]\rangle$ and $[2 \mapsto \texttt{sqr}]$.

$$\texttt{(cos pi)} \;\approx\; \{-0.999\}$$

Concrete semantics: k-CFA

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

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

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

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

And:

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

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

Abstract semantics: k-CFA

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

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

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

$$\hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}' \mapsto \hat{d}]$$

And:

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

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

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:

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

What makes k-CFA hard?

$$
\begin{array}{l}
((\lambda\ (f_1)\ \ (f_1\ 0)^{1}\ (f_1\ 1)^{2}) \\
\quad (\lambda\ (x_1) \\
\quad \ldots \\
\quad\quad ((\lambda\ (f_n)\ \ (f_n\ 0)^{2n-1}\ (f_n\ 1)^{2n}) \\
\quad\quad\quad (\lambda\ (x_n) \\
\quad\quad\quad\quad (\lambda\ (z)\ \ (z\ x_1 \ldots x_n)))) \cdots))
\end{array}
$$

1-CFA:

- $\star$ $[x_1 \mapsto 0, \ldots, x_n \mapsto 0]$
- $\star$ $[x_1 \mapsto 1, \ldots, x_n \mapsto 0]$
- $\star$ $\ldots$
- $\star$ $[x_1 \mapsto 0, \ldots, x_n \mapsto 1]$
- $\star$ $[x_1 \mapsto 1, \ldots, x_n \mapsto 1]$

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).

k-CFA is easy

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

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

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

A Resolution:
OO vs. functional k-CFA

FP in OO and back again

$$A \to B$$

$$\equiv$$

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

FP in OO and back again

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

$$\equiv$$

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

$$\equiv$$

```
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());
```

$$\equiv$$

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

$$\equiv$$

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

An Exploitation: m-CFA

The idea: Flat closures

Change the representation of closures:

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

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

Concrete semantics: m-CFA

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

$$\begin{aligned}
\mathcal{E}(f, a, \sigma) &= \langle \texttt{(}\lambda\ \texttt{(}x\texttt{)}\ e'\texttt{)}, a''\rangle & a' &= \ell \cdot a \\
\mathcal{E}(e, a, \sigma) &= d & d'_j &= \sigma(a'', y_j) \\
\{y_1, \ldots, y_n\} &= fv(e') \setminus \{x\} & \sigma' &= \sigma[a' \mapsto (d, d'_1, \ldots, d'_n)]
\end{aligned}$$

And:

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

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

Abstract semantics: m-CFA

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

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

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

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

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/`