Resolving and Exploiting the $\kappa$-CFA Paradox

Matthew Might, Yannis Smaragdakis, and David Van Horn
Plan

★ What is program analysis (aka abstract interpretation)?
★ 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 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
- ...

Resolving and Exploiting the k-CFA Paradox, University of Oregon CIS Colloquium, April, 2010 – p.3/3
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?

\[ ((\lambda (f) (f\ x)) \ (\text{if } P (\lambda (y) \ldots) (\lambda (z) \ldots))) \]
What is control flow analysis?

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

\[ \circ \cdot m(x); \]
What is control flow analysis?

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

```plaintext
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.
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);
}
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);
}
(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 $\kappa$-CFA?
Approximation in 0CFA

0CFA approximates closures by their code component.

\[
\begin{align*}
(\text{deriv } \sin) & = \langle (\lambda \ (x) \ldots f \ldots f \ldots), [f \mapsto \sin] \rangle \\
& \approx (\lambda \ (x) \ldots f \ldots f \ldots) \\
f & \approx \{\sin, \text{sqr}\}
\end{align*}
\]

\[
(\cos \ pi) \approx \{-0.999, 6.2831, \ldots\}
\]
What is $k$-CFA?

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

$$(\text{deriv } \text{sin})^1 \ldots (\text{deriv } \text{sqr})^2$$

1-CFA:

$\star \langle (\lambda (x) \ldots f \ldots f \ldots), [f \mapsto 1]\rangle \text{ and } [1 \mapsto \text{sin}].$

$\star \langle (\lambda (x) \ldots f \ldots f \ldots), [f \mapsto 2]\rangle \text{ and } [2 \mapsto \text{sqr}].$

$$(\cos \pi) \approx \{-0.999\}$$
Concrete semantics: $\kappa$-CFA

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

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

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

$a' = \ell \cdot a$

$\beta' = \beta''[x \mapsto a']$

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

And:

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

$\mathcal{E}( (\lambda \ (x) \ e), \beta, \sigma) = \langle (\lambda \ (x) \ e), \beta \rangle$
Abstract semantics: $\kappa$-CFA

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

$\hat{E}(f, \hat{\beta}, \hat{\sigma}) \ni \langle (\lambda (x) \ e') , \hat{\beta}'' \rangle$

$\hat{E}(e, \hat{\beta}, \hat{\sigma}) = \hat{d}$

$\hat{a}' = [\ell \cdot \hat{a}]_k$

$\hat{\beta}' = \hat{\beta}''[x \mapsto \hat{a}']$

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

And:

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

$\hat{E}( (\lambda (x) \ e) , \hat{\beta}, \hat{\sigma}) = \{ \langle (\lambda (x) \ e) , \hat{\beta} \rangle \}$
The Paradox of $\kappa$-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.

What makes $\kappa$-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?

\[
( (\lambda (f_1) (f_1 0)^1 (f_1 1)^2) \\
(\lambda (x_1)) \\
... \\
( (\lambda (f_n) (f_n 0)^{2n-1} (f_n 1)^{2n}) \\
(\lambda (x_n)) \\
(\lambda (z) (z x_1 ... x_n)) ) ) ) ... )
\]

1-CFA:

\[
* [x_1 \mapsto 0, \ldots, x_n \mapsto 0] \\
* [x_1 \mapsto 1, \ldots, x_n \mapsto 0] \\
* \ldots \\
* [x_1 \mapsto 0, \ldots, x_n \mapsto 1] \\
* [x_1 \mapsto 1, \ldots, x_n \mapsto 1]
\]

$k$-CFA is complete for EXPTIME (Van Horn and Mairson, ’08).
\( \kappa \)-CFA is easy

\( \kappa \)-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).

$\text{PTIME} \subset \text{EXPTIME}$
A Resolution:
OO vs. functional $\kappa$-CFA
FP in OO and back again

\[ A \to B \]

\[ \equiv \]

interface Fun\langle A, B \rangle \{ B \; \text{apply}(A \; a); \}

Resolving and Exploiting the k-CFA Paradox, University of Oregon CIS Colloquium, April, 2010 – p.25/3
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) \ (let \ x' = x \ in \ (\lambda (y) \ x')))))

\[
\equiv
\]
((\lambda (f) \ (f \ u) \ (f \ v)) \ (\lambda (x) \ ((\lambda (x') \ (\lambda (y) \ x')) \ x)))

Resolving and Exploiting the k-CFA Paradox, University of Oregon CIS Colloquium, April, 2010 – p.26/3
An Exploitation: $m$-CFA
The idea: Flat closures

Change the representation of closures:

\[ \text{Clo} = \text{Lam} \times (\text{Var} \rightarrow \text{Addr}) \]
\[ \text{Clo}' = \text{Lam} \times \text{Addr} \]

At call sites, copy values of closure into the store.
Concrete semantics: \( m\text{-CFA} \)

\[
(f \ e) ^{\ell} , a, \sigma \Rightarrow (e', a', \sigma'), \text{ where}
\]
\[
E(f, a, \sigma) = \langle (\lambda \ (x) \ e') , a'' \rangle \quad a' = \ell \cdot a
\]
\[
E(e, a, \sigma) = d \quad d'_j = \sigma(a'', y_j)
\]
\[
\{y_1, \ldots, y_n\} = fv(e') \setminus \{x\} \quad \sigma' = \sigma[a' \mapsto (d, d'_1, \ldots, d'_n)]
\]

And:

\[
E(x, a, \sigma) = \sigma(a, x)
\]
\[
E((\lambda \ (x) \ e), a, \sigma) = \langle (\lambda \ (x) \ e), a \rangle
\]
Abstract semantics: $m$-CFA

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

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

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

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

And:

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

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

Resolving and Exploiting the k-CFA Paradox, University of Oregon CIS Colloquium, April, 2010 – p.30/3
Evaluation and conclusion
## Speed: Worst-case benchmark

<table>
<thead>
<tr>
<th>Terms</th>
<th>$k = 1$</th>
<th>$m = 1$</th>
<th>poly.,$k=1$</th>
<th>$k=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>123</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>231</td>
<td>46 s</td>
<td>$\epsilon$</td>
<td>2 s</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>447</td>
<td>$\infty$</td>
<td>3 s</td>
<td>5 s</td>
<td>2 s</td>
</tr>
<tr>
<td>879</td>
<td>$\infty$</td>
<td>48 s</td>
<td>1 m 8 s</td>
<td>15 s</td>
</tr>
<tr>
<td>1743</td>
<td>$\infty$</td>
<td>51 m</td>
<td>$\infty$</td>
<td>3 m 48 s</td>
</tr>
</tbody>
</table>
## Speed and precision

<table>
<thead>
<tr>
<th>Prog/Terms</th>
<th>$k = 1$</th>
<th>$m = 1$</th>
<th>poly.,$k=1$</th>
<th>$k=0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>eta / 49</td>
<td>$\epsilon$ 7</td>
<td>$\epsilon$ 7</td>
<td>$\epsilon$ 3</td>
<td>$\epsilon$ 3</td>
</tr>
<tr>
<td>map / 157</td>
<td>$\epsilon$ 8</td>
<td>$\epsilon$ 8</td>
<td>$\epsilon$ 8</td>
<td>$\epsilon$ 6</td>
</tr>
<tr>
<td>sat / 223</td>
<td>$\infty$ -</td>
<td>$\epsilon$ 12</td>
<td>1s 12</td>
<td>$\epsilon$ 12</td>
</tr>
<tr>
<td>regex / 1015</td>
<td>4s 25</td>
<td>3s 25</td>
<td>14s 25</td>
<td>2s 25</td>
</tr>
<tr>
<td>scm2java / 2318</td>
<td>5s 86</td>
<td>3s 86</td>
<td>3s 79</td>
<td>4s 79</td>
</tr>
<tr>
<td>interp / 4289</td>
<td>5s 123</td>
<td>4s 123</td>
<td>9s 123</td>
<td>5s 123</td>
</tr>
<tr>
<td>scm2c / 6219</td>
<td>179s 136</td>
<td>143s 136</td>
<td>157s 131</td>
<td>55s 131</td>
</tr>
</tbody>
</table>

Column 1: analysis time, Column 2: inlinings enabled
Doggie bag

★ $k$-CFA of $\lambda$-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/