Relating Complexity and Precision in Control Flow Analysis

David Van Horn and Harry Mairson

Introduction

We investigate the *precision* of static, compile-time analysis, and the necessary analytic *tradeoff* with the computational *resources* that go into the analysis.

Outline

- What is Control Flow Analysis?
- What is kCFA?
- What is an exact analysis?
- Symmetric boolean logic gates
- PTIME-completeness of 0CFA
- NP-hardness of kCFA
- EXPTIME-completeness of nCFA
- A sketch of what's left in the paper

CFA Primer

1. For every application, which abstractions can be applied?
2. For every abstraction, to which arguments can it be applied?

Preliminaries: the Language

The language:

$$e ::= t^\ell \qquad \text{expressions (labeled terms)}$$

$$t ::= x \mid (e\ e) \mid (\lambda x.e) \qquad \text{terms (unlabeled expressions)}$$

For example:

$$((\lambda f.((f^1 f^2)^3 (\lambda y.y^4)^5)^6)^7 (\lambda x.x^8)^9)^{10}$$

Preliminaries: Contours

Contours are strings of @-labels of length $\leq k$.
Contour environments map variable names to contours.

$$\begin{array}{rcrcl} \delta & \in & \Delta & = & \mathbf{Lab}^{\leq k} \\ ce & \in & \mathbf{CEnv} & = & \mathbf{Var} \rightarrow \Delta \end{array}$$

Contours describe the context in which a term evaluates.
$(e_0([(\lambda x.e_1)]e_2)^{\ell_1})^{\ell_2} \Rightarrow$ "e_1 *evaluates in contour* $\ell_2\ell_1$."

Contour environments describe the context in which a variable was bound.
$(e_0([(\lambda x.e_1)]e_2)^{\ell_1})^{\ell_2} \Rightarrow x \mapsto \ell_2\ell_1 \Rightarrow$ "x *bound in contour* $\ell_2\ell_1$."

Preliminaries: the Analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps a label and contour to a set of abstract closures.

$$\widehat{\mathsf{C}} : \mathbf{Lab} \times \Delta \rightarrow \mathcal{P}(\mathbf{Term} \times \mathbf{CEnv})$$

$$\widehat{\mathsf{C}}(\ell, \delta) = \{\langle(\lambda y.\cdots), ce\rangle, \langle(\lambda z.\cdots), ce'\rangle\}$$

In contour δ, the term labeled ℓ evaluates to either the closure $\langle(\lambda y.\cdots), ce\rangle$, or $\langle(\lambda z.\cdots), ce'\rangle$.

Preliminaries: the Analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps a label and contour to a set of abstract closures.

$$\widehat{\mathsf{C}} : \mathbf{Lab} \times \Delta \to \mathcal{P}(\mathbf{Term} \times \mathbf{CEnv})$$

$$\widehat{\mathsf{C}}(\ell, \delta) = \{\langle \lambda y, ce \rangle, \langle \lambda z, ce' \rangle\}$$ shorthand

In contour δ, the term labeled ℓ evaluates to either the closure $\langle \lambda y, ce \rangle$, or $\langle \lambda z, ce' \rangle$.

Preliminaries: the Analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps a label and contour to a set of abstract closures.

$$\widehat{\mathsf{C}} : \mathbf{Lab} \times \Delta \rightarrow \mathcal{P}(\mathbf{Term} \times \mathbf{CEnv})$$

$$\widehat{\mathsf{C}}(x, \delta) = \{\langle \lambda y, ce \rangle, \langle \lambda z, ce' \rangle\}$$ overloading

In contour δ, x is bound to $\langle \lambda y, ce \rangle$, or $\langle \lambda z, ce' \rangle$...
... λx is applied to either $\langle \lambda y, ce \rangle$, or $\langle \lambda z, ce' \rangle$.

Decision problem

Control Flow Problem (kCFA): Given a closure and a label ℓ and contour δ, does that closure flow into the program point labeled ℓ under δ?

$$\langle \lambda x, ce \rangle \in \widehat{\mathsf{C}}(\ell, \delta)\text{?}$$

Acceptability

The analysis is acceptable for e, closed by ce, in contour δ:

$$\widehat{\mathsf{C}} \models_{\delta}^{ce} e$$

At the top-level (for a closed program), ce and δ are empty:

$$\widehat{\mathsf{C}} \models_{\epsilon}^{[\,]} e$$

What do δ and ce mean when non-empty?

Polyvariance

During reduction, a function may copy its argument:

$$((\lambda f. \cdots (f e_1)^{\ell_1} \cdots (f e_2)^{\ell_2} \cdots)(\lambda x.e))$$

Contours and environments let us talk about each copy of e:

$$\widehat{\mathsf{C}} \models_{\ell_1}^{x \mapsto \ell_1} e \qquad \widehat{\mathsf{C}} \models_{\ell_2}^{x \mapsto \ell_2} e$$

The analysis is *polyvariant*. Contours and environments describe which *instance* (copy) of a term we are talking about.

Acceptability

The analysis is acceptable for e, closed by ce, in contour δ:

$$\widehat{\mathsf{C}} \models^{ce}_{\delta} e$$

The analysis is acceptable for the copy of e that occurs in context described by δ, closed by the environment ce which says what copy of a term each variable is bound to.

Finally, let's look at what is acceptable…

Acceptability

$$\widehat{\mathsf{C}} \models_{\delta}^{ce} x^{\ell} \quad \text{iff} \quad \widehat{\mathsf{C}}(x, ce(x)) \subseteq \widehat{\mathsf{C}}(\ell, \delta)$$

$$\widehat{\mathsf{C}} \models_{\delta}^{ce} (\lambda x.e)^{\ell} \quad \text{iff} \quad \langle(\lambda x.e), ce_0\rangle \in \widehat{\mathsf{C}}(\ell, \delta)$$

$$\text{where } ce_0 = ce|\mathbf{fv}(\lambda x.e_0)$$

$$\widehat{\mathsf{C}} \models_{\delta}^{ce} (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \quad \text{iff} \quad \widehat{\mathsf{C}} \models_{\delta}^{ce} t_1^{\ell_1} \land \widehat{\mathsf{C}} \models_{\delta}^{ce} t_2^{\ell_2} \land$$

$$\forall \langle(\lambda x.t_0^{\ell_0}), ce_0\rangle \in \widehat{\mathsf{C}}(\ell_1, \delta):$$

$$\widehat{\mathsf{C}} \models_{\delta_0}^{ce_0'} t_0^{\ell_0} \land$$

$$\widehat{\mathsf{C}}(\ell_2, \delta) \subseteq \widehat{\mathsf{C}}(x, \delta_0) \land$$

$$\widehat{\mathsf{C}}(\ell_0, \delta_0) \subseteq \widehat{\mathsf{C}}(\ell, \delta)$$

$$\text{where } \delta_0 = \lceil \delta, \ell \rceil_k \text{ and } ce_0' = ce_0[x \mapsto \delta_0]$$

Acceptability

$$\widehat{\mathsf{C}} \models^{ce}_{\delta} x^{\ell} \quad \text{iff} \quad \widehat{\mathsf{C}}(x, ce(x)) \subseteq \widehat{\mathsf{C}}(\ell, \delta)$$

$$\widehat{\mathsf{C}} \models^{ce}_{\delta} (\lambda x.e)^{\ell} \quad \text{iff} \quad \langle(\lambda x.e), ce_0\rangle \in \widehat{\mathsf{C}}(\ell, \delta)$$

where $ce_0 = ce|\mathbf{fv}(\lambda x.e_0)$

$$\widehat{\mathsf{C}} \models^{ce}_{\delta} (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \quad \text{iff} \quad \widehat{\mathsf{C}} \models^{ce}_{\delta} t_1^{\ell_1} \wedge \widehat{\mathsf{C}} \models^{ce}_{\delta} t_2^{\ell_2} \wedge$$

$$\forall \langle(\lambda x.t_0^{\ell_0}), ce_0\rangle \in \widehat{\mathsf{C}}(\ell_1, \delta) :$$

$$\widehat{\mathsf{C}} \models^{ce_0'}_{\delta_0} t_0^{\ell_0} \wedge$$

$$\widehat{\mathsf{C}}(\ell_2, \delta) \subseteq \widehat{\mathsf{C}}(x, \delta_0) \wedge$$

$$\widehat{\mathsf{C}}(\ell_0, \delta_0) \subseteq \widehat{\mathsf{C}}(\ell, \delta)$$

where $\delta_0 = \lceil \delta, \ell \rceil_k$ and $ce_0' = ce_0[x \mapsto \delta_0]$

Mr. Yuck: Ingesting formalisms may cause *rigor mortis*

Exact analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps label-contour pairs to sets of abstract closures.

$$\widehat{\mathsf{C}}(\ell, \delta) = \{\langle \lambda y, ce \rangle, \langle \lambda z, ce' \rangle\}$$

In contour δ, the term labeled ℓ evaluates to either the closure $\langle \lambda y, ce \rangle$, or $\langle \lambda z, ce' \rangle$.

Exact analysis

An *exact* analysis is a table $\widehat{\mathsf{C}}$ that maps label-contour pairs to *an* abstract closure.

$$\widehat{\mathsf{C}}(\ell, \delta) = \{\langle \lambda y, ce \rangle\}$$

In contour δ, the term labeled ℓ evaluates to either the closure $\langle \lambda y, ce \rangle$.

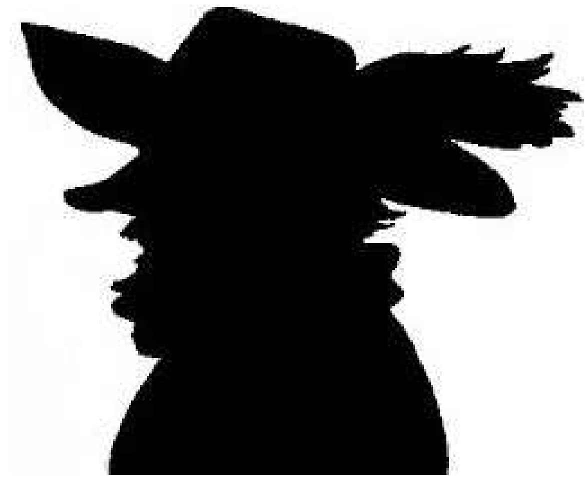
Exact analysis

An *exact* analysis is a table $\widehat{\mathsf{C}}$ that maps label-contour pairs to *an* abstract closure.

$$\widehat{\mathsf{C}}(\ell, \delta) = \{\langle \lambda y, ce \rangle\}$$

In contour δ, the term labeled ℓ evaluates to either the closure $\langle \lambda y, ce \rangle$.

Pick any nose you want! (You only have one)



Evaluator

Evaluate the term t, which is closed under environment ce.

$$\mathcal{E}[\![t^{\ell}]\!]^{ce}_{\delta}$$

Write the result into location (ℓ, δ) of the table.

Evaluator (exact)

$$
\begin{array}{lcl}
\mathcal{E}[\![x^\ell]\!]^{ce}_\delta & = & \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(x, ce(x)) \\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!]^{ce}_\delta & = & \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \langle \lambda x.e_0, ce_0 \rangle \\
 & & \quad \text{where } ce_0 = ce|\mathbf{fv}(\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!]^{ce}_\delta & = & \mathcal{E}[\![t_1^{\ell_1}]\!]^{ce}_\delta ; \mathcal{E}[\![t_2^{\ell_2}]\!]^{ce}_\delta ; \\
 & & \text{let } \langle \lambda x.t_0^{\ell_0}, ce_0 \rangle = \widehat{\mathsf{C}}(\ell_1,\delta) \text{ in} \\
 & & \quad \widehat{\mathsf{C}}(x,\delta,\ell) \leftarrow \widehat{\mathsf{C}}(\ell_2,\delta); \\
 & & \quad \mathcal{E}[\![t_0^{\ell_0}]\!]^{ce_0[x\mapsto\delta,\ell]}_{\delta,\ell} ; \\
 & & \quad \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(\ell_0,\delta,\ell)
\end{array}
$$

If e has an exact kCFA analysis, then $\mathcal{E}[\![e]\!]^{[\,]}_\epsilon$ constructs it.

Evaluator (exact)

$$
\begin{aligned}
\mathcal{E}[\![x^\ell]\!]^{ce}_\delta &= \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(x, ce(x))\\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!]^{ce}_\delta &= \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \langle \lambda x.e_0, ce_0\rangle\\
&\qquad \text{where } ce_0 = ce|\mathbf{fv}(\lambda x.e_0)\\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!]^{ce}_\delta &= \mathcal{E}[\![t_1^{\ell_1}]\!]^{ce}_\delta ; \mathcal{E}[\![t_2^{\ell_2}]\!]^{ce}_\delta ;\\
&\quad \text{let } \langle \lambda x.t_0^{\ell_0}, ce_0\rangle = \widehat{\mathsf{C}}(\ell_1,\delta) \text{ in}\\
&\qquad \widehat{\mathsf{C}}(x,\delta,\ell) \leftarrow \widehat{\mathsf{C}}(\ell_2,\delta);\\
&\qquad \mathcal{E}[\![t_0^{\ell_0}]\!]^{ce_0[x\mapsto\delta,\ell]}_{\delta,\ell};\\
&\qquad \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(\ell_0,\delta,\ell)
\end{aligned}
$$

Mr. Natural: Exact analysis is normalization.

If e has an exact kCFA analysis, then $\mathcal{E}[\![e]\!]^{[\,]}_\epsilon$ constructs it.

Evaluator (inexact)

$$
\begin{aligned}
\mathcal{E}[\![x^\ell]\!]^{ce}_\delta &= \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(x, ce(x)) \\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!]^{ce}_\delta &= \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \langle \lambda x.e_0, ce_0\rangle \\
&\quad \text{where } ce_0 = ce|\mathbf{fv}(\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!]^{ce}_\delta &= \mathcal{E}[\![t_1^{\ell_1}]\!]^{ce}_\delta ; \mathcal{E}[\![t_2^{\ell_2}]\!]^{ce}_\delta ; \\
&\quad \forall \langle \lambda x.t_0^{\ell_0}, ce_0\rangle \in \widehat{\mathsf{C}}(\ell_1,\delta) : \\
&\qquad \widehat{\mathsf{C}}(x, \lceil \delta,\ell\rceil_k) \leftarrow \widehat{\mathsf{C}}(\ell_2,\delta); \\
&\qquad \mathcal{E}[\![t_0^{\ell_0}]\!]^{ce_0[x\mapsto\lceil\delta,\ell\rceil_k]}_{\lceil\delta,\ell\rceil_k} ; \\
&\qquad \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(\ell_0, \lceil\delta,\ell\rceil_k)
\end{aligned}
$$

The kCFA analysis of e is constructed by iterating $\mathcal{E}[\![e]\!]^{[\,]}_\epsilon$ until $\widehat{\mathsf{C}}$ reaches a fixed point.

Evaluator (inexact, $k = 0$)

$$\begin{array}{lcl}
\mathcal{E}[\![x^\ell]\!]^{ce}_\delta & = & \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(x, ce(x)) \\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!]^{ce}_\delta & = & \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \langle \lambda x.e_0, ce_0\rangle \\
 & & \quad \text{where } ce_0 = ce|\mathbf{fv}(\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!]^{ce}_\delta & = & \mathcal{E}[\![t_1^{\ell_1}]\!]^{ce}_\delta ; \mathcal{E}[\![t_2^{\ell_2}]\!]^{ce}_\delta ; \\
 & & \forall \langle \lambda x.t_0^{\ell_0}, ce_0\rangle \in \widehat{\mathsf{C}}(\ell_1,\delta) : \\
 & & \quad \widehat{\mathsf{C}}(x, \lceil \delta, \ell\rceil_0) \leftarrow \widehat{\mathsf{C}}(\ell_2,\delta); \\
 & & \quad \mathcal{E}[\![t_0^{\ell_0}]\!]^{ce_0[x\mapsto \lceil\delta,\ell\rceil_0]}_{\lceil\delta,\ell\rceil_0} ; \\
 & & \quad \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(\ell_0, \lceil \delta,\ell\rceil_0)
\end{array}$$

But $\lceil \delta, \ell\rceil_0 = \epsilon$, so all contours are empty and all environments map all variables to ϵ. We can safely discard both.

Evaluator (inexact, $k = 0$)

$$
\begin{array}{lcl}
\mathcal{E}[\![x^{\ell}]\!] & = & \widehat{\mathsf{C}}(\ell) \leftarrow \widehat{\mathsf{C}}(x) \\
\mathcal{E}[\![(\lambda x.e_0)^{\ell}]\!] & = & \widehat{\mathsf{C}}(\ell) \leftarrow (\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^{\ell}]\!] & = & \mathcal{E}[\![t_1^{\ell_1}]\!]; \mathcal{E}[\![t_2^{\ell_2}]\!]; \\
 & & \forall (\lambda x.t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \\
 & & \quad \widehat{\mathsf{C}}(x) \leftarrow \widehat{\mathsf{C}}(\ell_2, \delta); \\
 & & \quad \mathcal{E}[\![t_0^{\ell_0}]\!]; \\
 & & \quad \widehat{\mathsf{C}}(\ell) \leftarrow \widehat{\mathsf{C}}(\ell_0)
\end{array}
$$

But $\lceil \delta, \ell \rceil_0 = \epsilon$, so all contours are empty and all environments map all variables to ϵ. We can safely discard both.

Evaluator (exact, $k = 0$)

$$
\begin{array}{lcl}
\mathcal{E}[\![x^\ell]\!] & = & \widehat{\mathsf{C}}(\ell) \leftarrow \widehat{\mathsf{C}}(x) \\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!] & = & \widehat{\mathsf{C}}(\ell) \leftarrow (\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!] & = & \mathcal{E}[\![t_1^{\ell_1}]\!]; \mathcal{E}[\![t_2^{\ell_2}]\!]; \\
& & \text{let } (\lambda x.t_0^{\ell_0}) = \widehat{\mathsf{C}}(\ell_1) \text{ in} \\
& & \quad \widehat{\mathsf{C}}(x) \leftarrow \widehat{\mathsf{C}}(\ell_2, \delta); \\
& & \quad \mathcal{E}[\![t_0^{\ell_0}]\!]; \\
& & \quad \widehat{\mathsf{C}}(\ell) \leftarrow \widehat{\mathsf{C}}(\ell_0)
\end{array}
$$

This is an evaluator, but for what language?

Evaluator (exact, $k = 0$)

$$
\begin{array}{lcl}
\mathcal{E}[\![x^\ell]\!] & = & \widehat{\mathsf{C}}(\ell) \leftarrow \widehat{\mathsf{C}}(x) \\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!] & = & \widehat{\mathsf{C}}(\ell) \leftarrow (\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!] & = & \mathcal{E}[\![t_1^{\ell_1}]\!]; \mathcal{E}[\![t_2^{\ell_2}]\!]; \\
& & \text{let } (\lambda x.t_0^{\ell_0}) = \widehat{\mathsf{C}}(\ell_1) \text{ in} \\
& & \quad \widehat{\mathsf{C}}(x) \leftarrow \widehat{\mathsf{C}}(\ell_2, \delta); \\
& & \quad \mathcal{E}[\![t_0^{\ell_0}]\!]; \\
& & \quad \widehat{\mathsf{C}}(\ell) \leftarrow \widehat{\mathsf{C}}(\ell_0)
\end{array}
$$

This is an evaluator, but for what language?

The linear λ-calculus.

Symmetric logic gates

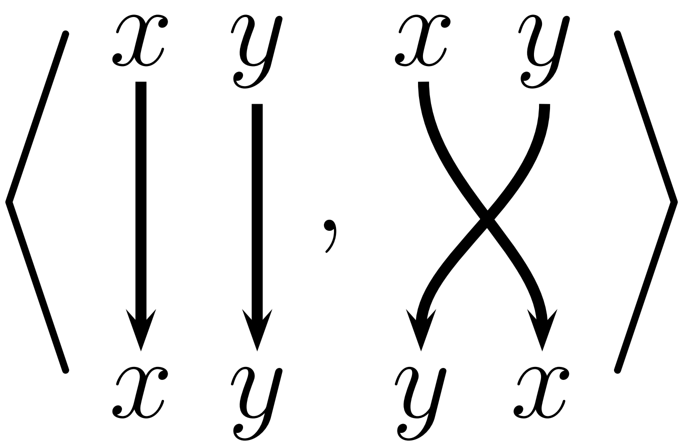
```
- fun TT(x:'a,y:'a)= (x,y);
val TT= fn : 'a * 'a -> 'a * 'a
- fun FF(x:'a,y:'a)= (y,x);
val FF= fn : 'a * 'a -> 'a * 'a
```

Booleans built out of constants TT, FF

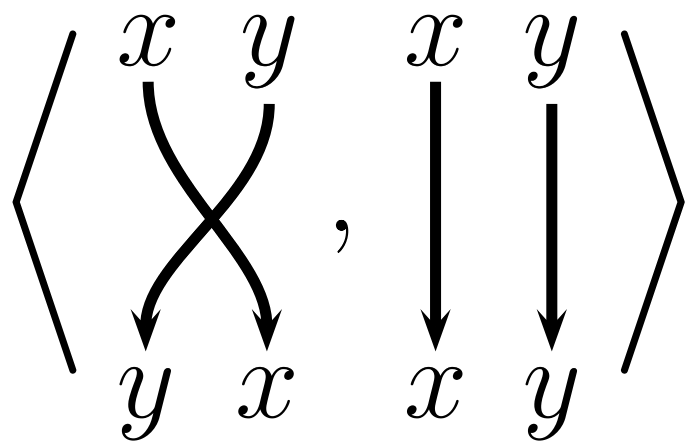
```
- val True= (TT: ('a * 'a -> 'a * 'a),  FF: ('a * 'a -> 'a * 'a));
val True = (fn,fn) : ('a * 'a -> 'a * 'a) * ('a * 'a -> 'a * 'a)

- val False= (FF: ('a * 'a -> 'a * 'a), TT: ('a * 'a -> 'a * 'a));
val False = (fn,fn) : ('a * 'a -> 'a * 'a) * ('a * 'a -> 'a * 'a)
```

True=



False=



Symmetric garbage is self-annihilating

$\texttt{And (p,p') (q,q')} \equiv \texttt{(p}\wedge\texttt{q, p'}\vee\texttt{q')} \equiv \texttt{(p}\wedge\texttt{q, }\neg\texttt{(p'}\wedge\texttt{q'))}$

```
- fun And (p,p') (q,q')=
    let val ((u,v),(u',v')) = (p (q,FF), p' (TT,q'))
    in (u,Compose (Compose (u',v),Compose (v',FF))) end;
val And = fn
  : ('a * ('b * 'b -> 'b * 'b) -> 'c * ('d -> 'e))
    * (('f * 'f -> 'f * 'f) * 'g  -> ('e -> 'h) * ('i * 'i -> 'd))
    -> 'a * 'g -> 'c * ('i * 'i -> 'h)
```

When p=TT (identity),

```
(u,v)   = (q,FF)
(u',v') = (q',TT)
```

When p=FF (twist),

```
(u,v)   = (FF,q)
(u',v') = (TT,q')
```

So, {v,v'}={TT,FF}, and

```
Compose (v,Compose(v',FF)) = TT
Compose (Compose (u',v),Compose (v',FF)) = u'
```

Symmetric logic gates

```
- fun Copy (p,p')= (p (TT,FF), p' (FF,TT));
val Copy = fn
  : (('a * 'a -> 'a * 'a) * ('b * 'b -> 'b * 'b) -> 'c)
    * (('d * 'd -> 'd * 'd) * ('e * 'e -> 'e * 'e) -> 'f)
    -> 'c * 'f
```

`[p= TT]: Copy (p,p') = ((TT,FF), (TT,FF))` [second component reversed]

`[p= FF]: Copy (p,p') = ((FF,TT), (FF,TT))` [first component reversed]

```
- fun Not (x,y)= (y,x);
val Not = fn : 'a * 'b -> 'b * 'a
```

`Or` is symmetric to `And`

PTIME Hardness of 0CFA

Code the boolean circuit as a linear term. Either `(u,u')` is bound to `(TT, FF)` or `(FF, TT)` (but not both, by linearity).

```
let val (u,u')= φ(v⃗) in
let val ((x,y),(x',y'))= (u (f,g), u' (f',g'))
    ((x a, y b),(x' a', y' b')) end end;
```

We know u is either `TT` (identity) or `FF` (twist)... so `(u (f,g))` is either `(f,g)` or `(g,f)`... therefore `(x,y)` is bound to either `(f,g)` or `(g,f)`...

so `f = x` iff $\phi(\vec{v})$= `True` (`f = y` iff $\phi(\vec{v})$= `False`)

so `f` is applied to a iff $\phi(\vec{v})$= `True`.

0CFA is PTIME-hard

PTIME Inclusion of 0CFA

Well known, eg. PPA Nielson et al. (1999):

- 0CFA computes a binary relation over a *fixed structure* (the graph description of a program).
- The computation of the relation is *monotone*: begins empty and is added to incrementally.
- A *fixed point* must be reached by this incremental computation (structure is finite).
- The binary relation can be at most *polynomial in size*, and each increment is *computed in polynomial time*.

0CFA is PTIME-complete

Closures

Because CFA makes approximations, many closures can flow to a single program point and contour. In 1CFA, for example,

$$(\lambda w.wx_1x_2\ldots x_n)$$

Has n free variables, with 2^n possible associated environments mapping these variables to program points (contours of length 1).

1CFA as SAT solver

$$(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False}))$$
$$(\lambda x_1.$$
$$\quad(\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False}))$$
$$\quad(\lambda x_2.$$
$$\qquad(\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False}))$$
$$\qquad(\lambda x_3.$$
$$\qquad\qquad\cdots$$
$$\qquad\qquad(\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False}))$$
$$\qquad\qquad(\lambda x_n.$$
$$\qquad\qquad\quad C[(\lambda v.\phi\ v)(\lambda w.w x_1 x_2 \cdots x_n)]) \cdots))))$$

1CFA as SAT solver

$(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False}))$

$(\lambda x_1.$

$(\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False}))$

$(\lambda x_2.$

$(\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False}))$

$(\lambda x_3.$

$\ldots$

$(\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False}))$

$(\lambda x_n.$

$C[(\lambda v.\phi\ v)(\lambda w.w x_1 x_2 \cdots x_n)]) \cdots))))$

Approximation allows us to bind each x_i to either of the closed λ-terms for True and False.

1CFA as SAT solver

$$
\begin{array}{l}
(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False})) \\
(\lambda x_1. \\
\quad (\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False})) \\
\quad (\lambda x_2. \\
\quad\quad (\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False})) \\
\quad\quad (\lambda x_3. \\
\quad\quad\quad\quad \ldots \\
\quad\quad\quad\quad (\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False})) \\
\quad\quad\quad\quad (\lambda x_n. \\
\quad\quad\quad\quad\quad C[(\lambda v.\phi\ v)(\lambda w.w x_1 x_2 \cdots x_n)]) \cdots))))
\end{array}
$$

Applying a Boolean function necessitates computation of all 2^n bindings to compute the flow out of the application.

1CFA as SAT solver

$$
\begin{array}{l}
(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False}))\\
(\lambda x_1.\\
\quad (\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False}))\\
\quad (\lambda x_2.\\
\qquad (\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False}))\\
\qquad (\lambda x_3.\\
\qquad\qquad \ldots\\
\qquad\qquad (\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False}))\\
\qquad\qquad (\lambda x_n.\\
\qquad\qquad\quad C[(\lambda v.\phi\ v)(\lambda w.w x_1 x_2 \cdots x_n)]) \cdots))))
\end{array}
$$

`True` flows out of the apply iff the Boolean function is satisfied by some truth valuation.

1CFA as SAT solver

$(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False}))$

$(\lambda x_1.$

$(\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False}))$

$(\lambda x_2.$

$(\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False}))$

$(\lambda x_3.$

$\cdots$

$(\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False}))$

$(\lambda x_n.$

$C[(\lambda v.\phi\ v)(\lambda w.w x_1 x_2 \cdots x_n)]) \cdots))))$

Approximation of closures as non-deterministic computation!

The Widget

$C =$

```
let val (u,u')= [ ] in
let val ((x,y),(x',y'))= (u (f,g), u' (f',g')) in
    ((x a, y b),(x' a', y' b')) end end;
```

In $C[(\lambda v.\phi\ v)(\lambda w.w x_1 x_2 \cdots x_n)]$:

`f` is applied to `a` iff ϕ is satisfiable.

1CFA is NP-hard

The Widget

$C =$

```
let val (u,u')= [ ] in
let val ((x,y),(x',y'))= (u (f,g), u' (f',g')) in
    ((x a, y b),(x' a', y' b')) end end;
```

In $C[(\lambda v.\phi\ v)(\lambda w.wx_1x_2\cdots x_n)]$:

`f` is applied to a iff ϕ is satisfiable.

1CFA is NP-hard

$(k > 1)$CFA is just as hard. The construction just needs to be "padded" to undo the added precision of longer contours.

kCFA is NP-hard

Naïve exponential algorithm for kCFA

- The $\widehat{\mathsf{C}}$ table is finite and has n^{k+1} entries.
- Each entry contains a set of closures.
- The environment maps p free variables to any one of n^k contours.
- There are n possible λx terms and n^{kp} environments, so each entry contains at most n^{1+kp} closures.
- Approximate evaluation is monotoic, and there are at most $n^{1+(k+1)p}$ updates to $\widehat{\mathsf{C}}$
- $p \leq n$ so kCFA in EXPTIME

kCFA in NP?

$$
\begin{aligned}
\mathcal{E}[\![x^\ell]\!]^{ce}_\delta &= \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(x, ce(x)) \\
\mathcal{E}[\![(\lambda x.e_0)^\ell]\!]^{ce}_\delta &= \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \langle \lambda x.e_0, ce_0\rangle \\
&\qquad \text{where } ce_0 = ce|\mathbf{fv}(\lambda x.e_0) \\
\mathcal{E}[\![(t_1^{\ell_1} t_2^{\ell_2})^\ell]\!]^{ce}_\delta &= \mathcal{E}[\![t_1^{\ell_1}]\!]^{ce}_\delta ; \mathcal{E}[\![t_2^{\ell_2}]\!]^{ce}_\delta ; \\
&\quad \forall \langle \lambda x.t_0^{\ell_0}, ce_0\rangle \in \widehat{\mathsf{C}}(\ell_1,\delta) : \\
&\qquad \widehat{\mathsf{C}}(x, \lceil \delta,\ell \rceil_k) \leftarrow \widehat{\mathsf{C}}(\ell_2,\delta); \\
&\qquad \mathcal{E}[\![t_0^{\ell_0}]\!]^{ce_0[x \mapsto \lceil \delta,\ell \rceil_k]}_{\lceil \delta,\ell \rceil_k}; \\
&\qquad \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(\ell_0, \lceil \delta,\ell \rceil_k)
\end{aligned}
$$

Can we guess our way through the computation to answer the kCFA decision problem?

Exact analysis for non-linear terms

0CFA is exact for linear terms...

...When is kCFA exact for non-linear terms?

Suppose ϕ is a linear term coding the transition function of a Turing machine and I is the (linear) initial machine configuration.

$$((\overline{2}\phi)I) \equiv (((\lambda s.(\lambda z.(s^1(s^2 z))))\phi)I)$$

1CFA analyzes each application of ϕ distinctly in contour 1 and 2, and therefore is *exact*...

...So analysis simulates 2 steps of the TM.

Exact analysis for non-linear terms

0CFA is exact for linear terms...

...When is kCFA exact for non-linear terms?

Scaling up, consider:

$$((\overline{2}(\overline{2}\phi))I) \equiv (((\lambda s.(\lambda z.(s^3(s^4 z))))$$
$$((\lambda s.(\lambda z.(s^1(s^2 z))))\phi))I)$$

2CFA analyzes each application of ϕ distinctly in contour 31, 32, 41 and 42, and therefore is *exact*...

...So analysis simulates 4 steps of the TM.

Exact analysis for non-linear terms

0CFA is exact for linear terms. . .

. . . When is kCFA exact for non-linear terms?

In general, nCFA is *exact* for:

$$((\overline{2}^n \phi) I)$$

. . . So analysis simulates 2^n steps of the TM.

Exact analysis for non-linear terms

0CFA is exact for linear terms. . .

. . . When is kCFA exact for non-linear terms?

In general, nCFA is *exact* for:

$$((\overline{2}^n \phi) I)$$

. . . So analysis simulates 2^n steps of the TM.

nCFA is EXPTIME-complete

The Doggie Bag

What I want you take home:

- Linearity subverts approximation in static analysis.
- It doesn't matter how big your k is (as long as it's constant), but how you use your closures.
- Either you run the program or you do something stupid.

What else?

Preprint available:

Relating Complexity and Precision in Control Flow Analysis

`http://www.cs.brandeis.edu/~dvanhorn/pubs/vanhorn-mairson-07.pdf`

- A graph formulation of kCFA
- Allows for (direct-style) kCFA of $\lambda + \texttt{call/cc}$, λ_μ, symmetric λ, (any calculus w/ a CPS semantics).
- Draws connection between kCFA and linear logic.
- 0CFA of simply-typed, η-expanded programs is LOGSPACE-complete.

The End

Drawings by Robert Crumb, used without permission.

Citations

References

Harry G. Mairson. From Hilbert spaces to Dilbert spaces: Context semantics made simple. In *FST TCS '02: Proceedings of the 22nd Conference Kanpur on Foundations of Software Technology and Theoretical Computer Science*, pages 2–17, London, UK, 2002. Springer-Verlag. ISBN 3-540-00225-1.

Peter Møller Neergaard and Harry G. Mairson. Types, potency, and idempotency: why nonlinearity and amnesia make a type system work. In *ICFP '04: Proceedings of the ninth ACM SIGPLAN international conference on Functional programming*, pages 138–149, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-905-5. doi: http://doi.acm.org/10.1145/1016850.1016871.

Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999. ISBN 3540654100.