Deciding kCFA is complete for EXPTIME

David Van Horn and Harry Mairson

Overview

For any $k > 0$, we prove that the control flow decision problem is complete for deterministic exponential time (**EXPTIME**).

This theorem:

- ⋆ gives an exact characterization of the computational complexity of the kCFA hierarchy
- ⋆ validates empirical observations that such control flow analysis is intractable

Plan

- ⋆ Proving lower bounds — *programming with analysis*
  - — What is kCFA?
  - — Linearity and precision
  - — Non-linearity and an exponential iterator
- ⋆ Simulating exponential Turing machines with kCFA
- ⋆ Conclusions

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

- ⋆ given the description of a Turing machine and its input,

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

- ⋆ given the description of a Turing machine and its input,
- ⋆ produce an instance of the kCFA problem,

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

- ⋆ given the description of a Turing machine and its input,
- ⋆ produce an instance of the kCFA problem,
- ⋆ whose analysis faithfully simulates the TM on the input

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

- ⋆ given the description of a Turing machine and its input,
- ⋆ produce an instance of the kCFA problem,
- ⋆ whose analysis faithfully simulates the TM on the input
- ⋆ for an exponential number of steps.

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

- ⋆ given the description of a Turing machine and its input,
- ⋆ produce an instance of the kCFA problem,
- ⋆ whose analysis faithfully simulates the TM on the input
- ⋆ for an exponential number of steps.

A compiler!

Proving lower bounds

A lower bound establishes the minimum computational requirements it takes to solve a class of problems.

kCFA is *provably intractable* (**EXPTIME**-hard)

The *proof* goes by construction:

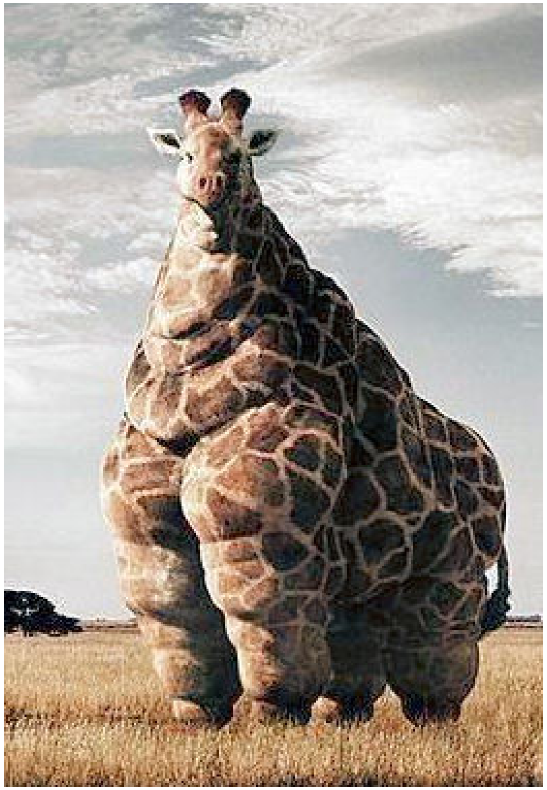
- ⋆ given the description of a Turing machine and its input,
- ⋆ produce an instance of the kCFA problem,
- ⋆ whose analysis faithfully simulates the TM on the input
- ⋆ for an exponential number of steps.

A (weird) compiler!

Strange animal

A compiler:

- $\star$ Source language: exponential TMs with input
- $\star$ Target language: the λ-calculus
- $\star$ Interpreter: kCFA (as TM simulator)



$\therefore$ kCFA is complete for **EXPTIME**.

More strange animals

Other compilers (ICFP'07):

- $\star$ Source language: Boolean formulas
- $\star$ Target language: the λ-calculus
- $\star$ Interpreter: kCFA (as SAT solver)

$\therefore$ kCFA is **NP**-hard.

More strange animals

Other compilers (ICFP'07):

- $\star$ Source language: circuit with inputs
- $\star$ Target language: the linear λ-calculus
- $\star$ Interpreter: 0CFA (as λ evaluator)

$\therefore$ 0CFA is complete for **PTIME**.

More strange animals

Other compilers (SAS'08):

- $\star$ Source language: circuit with inputs
- $\star$ Target language: the linear λ-calculus
- $\star$ Interpreter: Simple closure analysis (as λ evaluator)

$\therefore$ Simple closure analysis is complete for **PTIME**.

More strange animals

Other compilers (Mairson, JFP'04):

- ⋆ Source language: circuit with inputs
- ⋆ Target language: the linear λ-calculus
- ⋆ Interpreter: type inference (as λ evaluator)

$\therefore$ Simple type inference is complete for **PTIME**.

More strange animals

Other examples (Neergaard and Mairson, ICFP'04):

- $\star$ Source language: elementary TMs with input
- $\star$ Target language: the λ calculus
- $\star$ Interpreter: rank-k $\wedge$-type inference (as λ evaluator)

$\therefore$ Rank-k $\wedge$-type inference is complete for $\mathbf{DTIME}(\mathbf{K}(k,n))$.

More strange animals

Other examples (Mairson, POPL'89):

- $\star$ Source language: exponential TMs with input
- $\star$ Target language: ML
- $\star$ Interpreter: type inference (as ML evaluator)

$\therefore$ ML type inference is complete for **EXPTIME**.

A complexity zoo of static analysis

0CFA $\equiv$ Simple closure analysis $\equiv$ Sub-0CFA $\equiv$ Simple type inference $\equiv$ Linear λ-calculus $\equiv$ MLL...

$\subset$

kCFA $\equiv$ ML type inference...

$\subset$

Rank-k intersection type inference...

$\subset$

Exact CFA $\equiv$ Simply typed λ-calculus...

$\subset$

∞CFA $\equiv$ The λ-calculus...

"Program analysis is still far from being able to precisely relate ingredients of different approaches to one another."

(Nielson et al. 1999)

Plan

- ⋆ Proving lower bounds — *programming with analysis*
  - — What is kCFA?
  - — Linearity and precision
  - — Non-linearity and an exponential iterator
- ⋆ Simulating exponential Turing machines with kCFA
- ⋆ Conclusions

Flow analysis

Flow analysis is concerned with the sound approximation of run-time values at compile time.

Analysis answers decision problems such as:

does expression e *possibly evaluate to value* v*?*

- ⋆ The most approximate analysis always answers *yes*.
  — no resources to compute, but useless
- ⋆ The most precise analysis answers *yes* iff e evaluates to v.
  — useful, but unbounded resources to compute

For tractability, there is a necessary sacrifice of information in static analysis. (A blessing and a curse.)

kCFA

Intuition— the more information we compute about contexts, the more precisely we can answer flow questions.
But this takes work.

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)

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 (strings of application labels) let us talk about e in each of the distinct calling contexts.

Cache-based evaluator

Exp	$e ::= t^{\ell}$	expressions (or labeled terms)
Term	$t ::= x \mid e\ e \mid \lambda x.e$	terms (or unlabeled expressions)

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 cache C.

C$(\ell, \delta) = v$ means t^{ℓ} evaluates to v in context δ.

∞CFA

A cache-based evaluator:

$$\mathsf{C} \in \mathbf{Cache} = (\mathbf{Lab} + \mathbf{Var}) \times \mathbf{Lab}^{\star} \rightarrow (\mathbf{Term} \times \mathbf{Env})$$

$$\begin{array}{rcl} \mathcal{E}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]^{ce}_{\delta} & = & \mathcal{E}[\![t^{\ell_1}]\!]^{ce}_{\delta}; \mathcal{E}[\![t^{\ell_2}]\!]^{ce}_{\delta}; \\ & & \text{let } \langle \lambda x.t^{\ell_0}, ce' \rangle = \mathsf{C}(\ell_1, \delta) \text{ in} \\ & & \quad \mathsf{C}(x, \delta\ell) \leftarrow \mathsf{C}(\ell_2, \delta); \\ & & \quad \mathcal{E}[\![t^{\ell_0}]\!]^{ce'[x \mapsto \delta\ell]}_{\delta\ell} \\ & & \quad \mathsf{C}(\ell, \delta) \leftarrow \mathsf{C}(\ell_0, \delta\ell) \end{array}$$

kCFA

An *abstraction* of the cache-based evaluator:

$$\widehat{\mathsf{C}} \in \widehat{\mathbf{Cache}} = (\mathbf{Lab} + \mathbf{Var}) \times \mathbf{Lab}^{\leq k} \to \mathcal{P}(\mathbf{Term} \times \mathbf{Env})$$

$$\begin{aligned}
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]^{ce}_{\delta} \quad = \quad & \mathcal{A}[\![t^{\ell_1}]\!]^{ce}_{\delta}; \mathcal{A}[\![t^{\ell_2}]\!]^{ce}_{\delta}; \\
& \textsf{foreach } \langle \lambda x.t^{\ell_0}, ce' \rangle \in \widehat{\mathsf{C}}(\ell_1, \delta) : \\
& \quad \widehat{\mathsf{C}}(x, \lceil \delta\ell \rceil_k) \leftarrow \widehat{\mathsf{C}}(\ell_2, \delta); \\
& \quad \mathcal{A}[\![t^{\ell_0}]\!]^{ce'[x \mapsto \lceil \delta\ell \rceil_k]}_{\lceil \delta\ell \rceil_k}; \\
& \quad \widehat{\mathsf{C}}(\ell, \delta) \leftarrow \widehat{\mathsf{C}}(\ell_0, \lceil \delta\ell \rceil_k)
\end{aligned}$$

Plan

- ⋆ Proving lower bounds — *programming with analysis*
  - — What is kCFA?
  - — Linearity and precision
  - — Non-linearity and an exponential iterator
- ⋆ Simulating exponential Turing machines with kCFA
- ⋆ Conclusions

Linearity and evaluation

Since in a *linear* λ-term,

- $\star$ each abstraction can be applied to at most one argument
- $\star$ each variable can be bound to at most one value

Analysis of a linear term coincides exactly with its evaluation.

Boolean logic

Coding Boolean logic in linear λ-calculus (ICFP'07):

$$\mathtt{TT} \equiv \lambda p.\text{let } \langle x, y\rangle = p \text{ in } \langle x, y\rangle \qquad \mathtt{True} \equiv \langle \mathtt{TT}, \mathtt{FF}\rangle$$

$$\mathtt{FF} \equiv \lambda p.\text{let } \langle x, y\rangle = p \text{ in } \langle y, x\rangle \qquad \mathtt{False} \equiv \langle \mathtt{FF}, \mathtt{TT}\rangle$$

$$\mathtt{Copy} \equiv \lambda b.\text{let } \langle u, v\rangle = b \text{ in } \langle u\langle \mathtt{TT}, \mathtt{FF}\rangle, v\langle \mathtt{FF}, \mathtt{TT}\rangle\rangle$$

$$\begin{aligned}\mathtt{Implies} \equiv\ & \lambda b_1.\lambda b_2. \\ & \text{let } \langle u_1, v_1\rangle = b_1 \text{ in} \\ & \text{let } \langle u_2, v_2\rangle = b_2 \text{ in} \\ & \text{let } \langle p_1, p_2\rangle = u_1\langle u_2, \mathtt{TT}\rangle \text{ in} \\ & \text{let } \langle q_1, q_2\rangle = v_1\langle \mathtt{FF}, v_2\rangle \text{ in} \\ & \quad \langle p_1, q_1 \circ p_2 \circ q_2 \circ \mathtt{FF}\rangle\end{aligned}$$

Plan

- Proving lower bounds — *programming with analysis*
  - What is kCFA?
  - Linearity and precision
  - Non-linearity and an exponential iterator
- Simulating exponential Turing machines with kCFA
- Conclusions

Approximation as power tool

Hardness of kCFA relies on two insights:

1. Program points are approximated by an exponential number of closures.
2. *Inexactness* of analysis engenders *reevaluation* which provides *computational power*.

Abstract closures

Many closures can flow to a single program point:

$$(\lambda w.w x_1 x_2 \ldots x_n)$$

- ⋆ n free variables
- ⋆ an *exponential* number of possible associated environments mapping these variables to program points (contours of length 1 in 1CFA).

Toy calculation, with insights

Consider the following *non-linear* example

$$(\lambda f.(f\ \texttt{True})(f\ \texttt{False}))$$
$$(\lambda x.$$
$$(\lambda p.p(\lambda u.p(\lambda v.(\texttt{Implies}\ u\ v))))(\lambda w.wx))$$

Toy calculation, with insights

Consider the following *non-linear* example

$$(\lambda f.(f\ \texttt{True})(f\ \texttt{False}))$$
$$(\lambda x.$$
$$(\lambda p.p(\lambda u.p(\lambda v.(\texttt{Implies}\ u\ v))))(\lambda w.wx))$$

Q: What does $\texttt{Implies}\ u\ v$ evaluate to ?

Toy calculation, with insights

Consider the following *non-linear* example

$$(\lambda f.(f\ \texttt{True})(f\ \texttt{False}))$$
$$(\lambda x.$$
$$(\lambda p.p(\lambda u.p(\lambda v.(\texttt{Implies}\ u\ v))))(\lambda w.wx))$$

Q: What does $\texttt{Implies}\ u\ v$ evaluate to?

A: $\texttt{True}$: it is equivalent to $\texttt{Implies}\ x\ x$, a tautology.

Toy calculation, with insights

Consider the following *non-linear* example

$$(\lambda f.(f\ \texttt{True})(f\ \texttt{False}))$$
$$(\lambda x.$$
$$(\lambda p.p(\lambda u.p(\lambda v.(\texttt{Implies}\ u\ v))))(\lambda w.wx))$$

Q: What does $\texttt{Implies}\ u\ v$ evaluate to ?
A: $\texttt{True}$: it is equivalent to $\texttt{Implies}\ x\ x$, a tautology.

Q: What flows out of $\texttt{Implies}\ u\ v$?

Toy calculation, with insights

Consider the following *non-linear* example

$$(\lambda f.(f\ \texttt{True})(f\ \texttt{False}))$$
$$(\lambda x.$$
$$\quad (\lambda p.p(\lambda u.p(\lambda v.(\texttt{Implies}\ u\ v))))(\lambda w.wx))$$

Q: What does $\texttt{Implies}\ u\ v$ evaluate to ?
A: $\texttt{True}$: it is equivalent to $\texttt{Implies}\ x\ x$, a tautology.

Q: What flows out of $\texttt{Implies}\ u\ v$?
A: **both** $\texttt{True}$ and $\texttt{False}$: *Not true evaluation!*

Toy calculation, with insights

Consider the following *non-linear* example

$$(\lambda f.(f\ \texttt{True})(f\ \texttt{False}))$$
$$(\lambda x.$$
$$(\lambda p.p(\lambda u.p(\lambda v.(\texttt{Implies}\ u\ v))))(\lambda w.wx))$$

Q: What does $\texttt{Implies}\ u\ v$ evaluate to?
A: `True`: it is equivalent to $\texttt{Implies}\ x\ x$, a tautology.

Q: What flows out of $\texttt{Implies}\ u\ v$?
A: **both** `True` and `False`: *Not true evaluation!*

- We are *computing with the approximation* (spurious flows).

Plan

- ⋆ Proving lower bounds — *programming with analysis*
  - — What is kCFA?
  - — Linearity and precision
  - — Non-linearity and an exponential iterator
- ⋆ Simulating exponential Turing machines with kCFA
- ⋆ Conclusions

Jigsaw puzzles, Machines

The idea:

- ⋆ Break machine ID into an exponential number of pieces
- ⋆ Do piecemeal transitions on pairs of puzzle pieces

$$\langle T, S, H, C, b \rangle$$

“At time T, machine is in state S, the head is at cell H, and cell C holds symbol b”

Jigsaw puzzles, Machines

$\langle T, S, H, C, b\rangle$: *"At time T, machine is in state S, the head is at cell H, and cell C holds symbol b"*

1) Compute:

$$\delta\langle T, S, H, H, b\rangle\langle T, S', H', C', b'\rangle =$$
$$\langle T+1, \delta_Q(S, b), \delta_{LR}(S, H, b), H, \delta_\Sigma(S, b)\rangle$$

Jigsaw puzzles, Machines

$\langle T, S, H, C, b\rangle$: *"At time T, machine is in state S, the head is at cell H, and cell C holds symbol b"*

1) Compute:

$$\delta\langle T, S, H, H, b\rangle\langle T, S', H', C', b'\rangle = \langle T+1, \delta_Q(S, b), \delta_{LR}(S, H, b), H, \delta_\Sigma(S, b)\rangle$$

2) Communicate:

$$\delta\langle T+1, S, H, C, b\rangle\langle T, S', H', C', b'\rangle = \langle T+1, S, H, C', b'\rangle$$

$$(H' \neq C')$$

Jigsaw puzzles, Machines

$\langle T, S, H, C, b\rangle$: *"At time T, machine is in state S, the head is at cell H, and cell C holds symbol b"*

1) Compute:

$$\delta\langle T, S, H, H, b\rangle\langle T, S', H', C', b'\rangle = \langle T+1, \delta_Q(S, b), \delta_{LR}(S, H, b), H, \delta_\Sigma(S, b)\rangle$$

2) Communicate:

$$\delta\langle T+1, S, H, C, b\rangle\langle T, S', H', C', b'\rangle = \langle T+1, S, H, C', b'\rangle$$

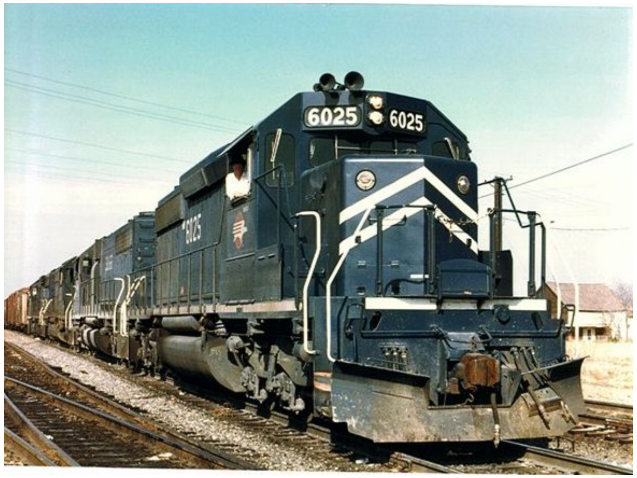
$$(H' \neq C')$$

3) Otherwise:

$$\delta\langle T, S, H, C, b\rangle\langle T', S', H', C', b'\rangle = \langle\text{some goofy}$$  $$\text{null value}\rangle$$

$$(T \neq T' \text{ and } T \neq T' + 1)$$

The real deal



Setting up initial ID, iterator, and test:

$$(\lambda f_1.(f_1\ \mathbf{0})(f_1\ \mathbf{1}))$$
$$(\lambda z_1.$$
$$(\lambda f_2.(f_2\ \mathbf{0})(f_2\ \mathbf{1}))$$
$$(\lambda z_2.$$
$$\ldots$$
$$(\lambda f_N.(f_N\ \mathbf{0})(f_N\ \mathbf{1}))$$
$$(\lambda z_N.$$

(let $\Phi =$ *coding of transition function of TM* in

$\texttt{Widget}[\texttt{Extract}(Y\ \Phi\ (\lambda w.w\ \mathbf{0}\ldots\mathbf{0}\ Q_0\ H_0\ z_1 z_2\ldots z_N\ \mathbf{0}))])) \ldots))$

$\langle T, S, H, \qquad C, b\rangle$

The real deal

$\ldots$let $\Phi =$ *coding of transition function of TM* in

$$\texttt{Widget}[\texttt{Extract}(Y\ \Phi\ (\lambda w.w\ \mathbf{0}\ldots\mathbf{0}\ Q_0\ H_0\ z_1 z_2\ldots z_N\ \mathbf{0}))]\ldots$$

$$\langle T, S, H, \qquad C, b\rangle$$

$$\Phi \equiv (\lambda p.p(\lambda x_1.\lambda x_2.\ldots.\lambda x_m.p(\lambda y_1.\lambda y_2\ldots.\lambda y_m.$$
$$(\phi x_1 x_2 \ldots x_m y_1 y_2 \ldots y_m))))$$

The real deal

. . . let $\Phi =$ *coding of transition function of TM* in

$$\texttt{Widget}[\texttt{Extract}(Y\ \Phi\ (\lambda w.w\ \mathbf{0}\ldots\mathbf{0}\ Q_0\ H_0\ z_1 z_2\ldots z_N\ \mathbf{0}))]\ldots$$

$$\langle T, S, H, \qquad C, b\rangle$$

$$\Phi \equiv (\lambda p.p(\lambda x_1.\lambda x_2.\ldots.\lambda x_m.p(\lambda y_1.\lambda y_2\ldots.\lambda y_m.$$
$$(\phi x_1 x_2 \ldots x_m y_1 y_2 \ldots y_m))))$$

$\texttt{Widget}[E] \equiv \ldots f \ldots a \ldots$, where a flows as an argument to f iff a True value flows out of E.

The real deal

…let Φ = *coding of transition function of TM* in

$$\texttt{Widget}[\texttt{Extract}(Y\ \Phi\ (\lambda w.w\ \mathbf{0}\ldots\mathbf{0}\ Q_0\ H_0\ z_1 z_2\ldots z_N\ \mathbf{0}))]\ldots$$

$$\langle T, S, H, \qquad C, b\rangle$$

$$\Phi \equiv (\lambda p.p(\lambda x_1.\lambda x_2.\ldots.\lambda x_m.p(\lambda y_1.\lambda y_2\ldots\lambda y_m.$$
$$(\phi x_1 x_2 \ldots x_m y_1 y_2 \ldots y_m))))$$

$\texttt{Widget}[E] \equiv \ldots f \ldots a \ldots$, where a flows as an argument to f iff a True value flows out of E.

Theorem In kCFA, a flow to f iff TM accept in 2^n steps.

The real deal

$\ldots$let $\Phi =$ *coding of transition function of TM* in

$$\texttt{Widget}[\texttt{Extract}(Y\ \Phi\ (\lambda w.w\ \mathbf{0}\ldots\mathbf{0}\ Q_0\ H_0\ z_1 z_2\ldots z_N\ \mathbf{0}))]\ldots$$

$$\langle T, S, H, \qquad C, b\rangle$$

$$\Phi \equiv (\lambda p.p(\lambda x_1.\lambda x_2.\ldots.\lambda x_m.p(\lambda y_1.\lambda y_2\ldots\lambda y_m.$$
$$(\phi x_1 x_2 \ldots x_m y_1 y_2 \ldots y_m))))$$

$\texttt{Widget}[E] \equiv \ldots f \ldots a \ldots$, where a flows as an argument to f iff a True value flows out of E.

Theorem In kCFA, a flow to f iff TM accept in 2^n steps.
Theorem kCFA decision problem is complete for **EXPTIME**.

Plan

- ⋆ Proving lower bounds — *programming with analysis*
  - — What is kCFA?
  - — Linearity and precision
  - — Non-linearity and an exponential iterator
- ⋆ Simulating exponential Turing machines with kCFA
- ⋆ Conclusions

What makes kCFA hard?

This is not just a replaying of the previous proofs.

What makes kCFA hard?

This is not just a replaying of the previous proofs.

- ⋆ If the analysis were simulating evaluation,

What makes kCFA hard?

This is not just a replaying of the previous proofs.

- ⋆ If the analysis were simulating evaluation,
- ⋆ there would be one entry in each cache location,

What makes kCFA hard?

This is not just a replaying of the previous proofs.

- ⋆ If the analysis were simulating evaluation,
- ⋆ there would be one entry in each cache location,
- ⋆ therefore bounded by a polynomial!

What makes kCFA hard?

This is not just a replaying of the previous proofs.

- $\star$ If the analysis were simulating evaluation,
- $\star$ there would be one entry in each cache location,
- $\star$ therefore bounded by a polynomial!

Might and Shivers' observation:
improved precision leads to analyzer speedups.

What makes kCFA hard?

This is not just a replaying of the previous proofs.

- ⋆ If the analysis were simulating evaluation,
- ⋆ there would be one entry in each cache location,
- ⋆ therefore bounded by a polynomial!

Might and Shivers' observation:
improved precision leads to analyzer speedups.

Analytic understanding:

What you pay for in kCFA is the junk (spurious flows).

Doggie bag

- ⋆ There is no tractable algorithm for kCFA
- ⋆ Linearity is key in understanding static analysis
- ⋆ The approximation of kCFA is what makes it hard

The End

Thank you.