Deciding $k$CFA 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 ($\text{EXPTIME}$).

This theorem:

- gives an exact characterization of the computational complexity of the $k$-CFA hierarchy
- validates empirical observations that such control flow analysis is intractable
Plan

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

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

$k$CFA 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.

$k$CFA 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.

$k$CFA 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 $k$CFA problem,
Proving lower bounds

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

$k$CFA 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 $k$CFA 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.

\( k \text{CFA} \) is \textit{provably intractable} (EXPTIME-hard)

The \textit{proof} goes by construction:

\begin{itemize}
  \item given the description of a Turing machine and its input,
  \item produce an instance of the \( k \text{CFA} \) problem,
  \item whose analysis faithfully simulates the TM on the input
  \item for an exponential number of steps.
\end{itemize}
Proving lower bounds

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

$k$CFA 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 $k$CFA 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.

$k$CFA 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 $k$CFA problem,
★ whose analysis faithfully simulates the TM on the input
★ for an exponential number of steps.

A (weird) compiler!
Strange animal

A compiler:

★ Source language: exponential TMs with input
★ Target language: the $\lambda$-calculus
★ Interpreter: $k$CFA (as TM simulator)

$\therefore$ $k$CFA is complete for **EXPTIME**.
More strange animals

Other compilers (ICFP’07):

★ Source language: Boolean formulas
★ Target language: the $\lambda$-calculus
★ Interpreter: $k$CFA (as SAT solver)

∴ $k$CFA is NP-hard.
More strange animals

Other compilers (ICFP’07):
★ Source language: circuit with inputs
★ Target language: the linear $\lambda$-calculus
★ Interpreter: 0CFA (as $\lambda$ evaluator)
∴ 0CFA is complete for $\text{PTIME}$.
More strange animals

Other compilers (SAS’08):

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

∴ Simple closure analysis is complete for $\text{PTIME}$. 
More strange animals

Other compilers (Mairson, JFP’04):

- Source language: circuit with inputs
- Target language: the linear $\lambda$-calculus
- Interpreter: type inference (as $\lambda$ evaluator)

∴ Simple type inference is complete for \textbf{PTIME}.
More strange animals

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

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

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

International Conference on Functional Programming (ICFP), 2008
More strange animals

Other examples (Mairson, POPL’89):

★ Source language: exponential TMs with input
★ Target language: ML
★ Interpreter: type inference (as ML evaluator)

∴ ML type inference is complete for EXPTIME.
A complexity zoo of static analysis

\[ 0\text{CFA} \equiv \text{Simple closure analysis} \equiv \text{Sub-0CFA} \equiv \text{Simple type inference} \equiv \text{Linear } \lambda\text{-calculus} \equiv \text{MLL} \ldots \]
\[ \subset \]
\[ k\text{CFA} \equiv \text{ML type inference} \ldots \]
\[ \subset \]
\[ \text{Rank-}k \text{ intersection type inference} \ldots \]
\[ \subset \]
\[ \text{Exact CFA} \equiv \text{Simply typed } \lambda\text{-calculus} \ldots \]
\[ \subset \]
\[ \infty\text{CFA} \equiv \text{The } \lambda\text{-calculus} \ldots \]

“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 $k$CFA?
  — Linearity and precision
  — Non-linearity and an exponential iterator
- Simulating exponential Turing machines with $k$CFA
- Conclusions
Flow analysis

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

Analysis answers decision problems such as:

\[ \text{does expression } e \text{ possibly evaluate to value } v? \]

- 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.)
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.*

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

\[ \text{Exp} \quad e ::= t^\ell \quad \text{expressions (or labeled terms)} \]
\[ \text{Term} \quad t ::= x \mid e \cdot e \mid \lambda x. e \quad \text{terms (or unlabeled expressions)} \]

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

Write the result into location \( (\ell, \delta) \) of the cache \( C \).

\[ E \llbracket t^\ell \rrbracket_{ce} \]

\[ C(\ell, \delta) = v \text{ means } t^\ell \text{ evaluates to } v \text{ in context } \delta. \]
A cache-based evaluator:

\[ C \in \text{Cache} = (\text{Lab} + \text{Var}) \times \text{Lab}^* \rightarrow (\text{Term} \times \text{Env}) \]

\[ \mathcal{E}[(t^{\ell_1} t^{\ell_2})^{\ell}]_{\delta}^c e = \mathcal{E}[t^{\ell_1}]_{\delta}^c e ; \mathcal{E}[t^{\ell_2}]_{\delta}^c e ; \]

let \langle \lambda x. t^{\ell_0} , ce' \rangle = C(\ell_1, \delta) in

\[ C(x, \delta \ell) \leftarrow C(\ell_2, \delta) ; \]

\[ \mathcal{E}[t^{\ell_0}]_{\delta \ell}^{ce'}[x \mapsto \delta \ell] \]

\[ C(\ell, \delta) \leftarrow C(\ell_0, \delta \ell) \]
An *abstraction* of the cache-based evaluator:

\[ \hat{C} \in \widehat{\text{Cache}} = (\text{Lab} + \text{Var}) \times \text{Lab}^{\leq k} \rightarrow \mathcal{P}(\text{Term} \times \text{Env}) \]

\[ A\llbracket (t^{\ell_1} t^{\ell_2})^\ell \rrbracket_{ce}^\delta = A\llbracket t^{\ell_1} \rrbracket_{ce}^\delta ; A\llbracket t^{\ell_2} \rrbracket_{ce}^\delta ; \]

foreach \( \langle \lambda x. t^{\ell_0}, ce' \rangle \in \hat{C}(\ell_1, \delta) \):

\[ \hat{C}(x, [\delta \ell]_k) \leftarrow \hat{C}(\ell_2, \delta) ; \]

\[ A\llbracket t^{\ell_0} \rrbracket_{ce'}^{\delta \ell}[x \mapsto [\delta \ell]_k] ; \]

\[ \hat{C}(\ell, \delta) \leftarrow \hat{C}(\ell_0, [\delta \ell]_k) \]
Plan

- Proving lower bounds — programming with analysis
  - What is \( k \text{CFA} \)?
  - Linearity and precision
  - Non-linearity and an exponential iterator
- Simulating exponential Turing machines with \( k \text{CFA} \)
- Conclusions
Linearity and evaluation

Since in a *linear* $\lambda$-term,

$\bullet$ each abstraction can be applied to at most one argument
$\bullet$ 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 $\lambda$-calculus (ICFP’07):

$$\begin{align*}
\text{TT} & \equiv \lambda p. \text{let } \langle x, y \rangle = p \text{ in } \langle x, y \rangle \\
\text{FF} & \equiv \lambda p. \text{let } \langle x, y \rangle = p \text{ in } \langle y, x \rangle \\
\text{True} & \equiv \langle \text{TT}, \text{FF} \rangle \\
\text{False} & \equiv \langle \text{FF}, \text{TT} \rangle
\end{align*}$$

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

Implies $\equiv \lambda b_1. \lambda b_2. \left( \text{let } \langle u_1, v_1 \rangle = b_1 \text{ in } \langle u_1 \langle u_2, \text{TT} \rangle, v_1 \langle \text{FF}, v_2 \rangle \rangle \text{ in } \langle p_1, q_1 \circ p_2 \circ q_2 \circ \text{FF} \rangle \right)$
Plan

★ Proving lower bounds — *programming with analysis*
  — What is $k$CFA?
  — Linearity and precision
  — Non-linearity and an exponential iterator
★ Simulating exponential Turing machines with $k$CFA
★ Conclusions
Approximation as power tool

Hardness of $\kappa$CFA 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

\[(\lambda f. (f \text{ True})(f \text{ False})) (\lambda x. (\lambda p. p(\lambda u. p(\lambda v. (\text{Implies } u v)))))(\lambda w. wx))\]
Toy calculation, with insights

Consider the following non-linear example

$$(\lambda f. (f \text{ True}) (f \text{ False}))$$

$$(\lambda x. (\lambda p. p (\lambda u. p (\lambda v. (\text{Implies } u v)))) (\lambda w. w x))$$

Q: What does $\text{Implies } u v$ evaluate to?
Toy calculation, with insights

Consider the following non-linear example

\[
(\lambda f. (f \ True)(f \ False))
(\lambda x. \\
(\lambda p.p(\lambda u.p(\lambda v.(Implies \ u \ v)))))(\lambda w.wx))
\]

Q: What does \texttt{Implies} \ u \ v evaluate to?  
A: \texttt{True}: it is equivalent to \texttt{Implies} \ x \ x, a tautology.
Consider the following non-linear example

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

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

Q: What flows out of $\text{Implies } u v$?
Consider the following *non-linear* example

\[
(\lambda f. (f \text{ True})(f \text{ False}))
(\lambda x.
(\lambda p.p(\lambda u.p(\lambda v.\text{Implies } u v))))(\lambda w.wx))
\]

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

Q: What \text{flows out of} \text{Implies } u v?
A: both \text{True} and \text{False}: \text{Not true evaluation}!
Toy calculation, with insights

Consider the following non-linear example

\[
(\lambda f.(f \ True)(f \ False))
\]

\[
(\lambda x. (\lambda p.p(\lambda u. p(\lambda v. (\text{Implies } u \ v)))))(\lambda w.wx))
\]

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

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

We are \text{computing with the approximation (spurious flows)}. 
Plan

★ Proving lower bounds — *programming with analysis*
   — What is $k$CFA?
   — Linearity and precision
   — Non-linearity and an exponential iterator
★ Simulating exponential Turing machines with $k$CFA
★ 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

\[ \langle T, S, H, C, b \rangle: \text{“At time } T, \text{ machine is in state } S, \text{ the head is at cell } H, \text{ and cell } C \text{ holds symbol } b \rangle \]

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

\[ \langle T, S, H, C, b \rangle: \text{“At time } T, \text{ machine is in state } S, \text{ the head is at cell } H, \text{ and cell } C \text{ holds symbol } b \text{”} \]

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:
\[
\delta \langle T + 1, S, H, C, b \rangle \langle T, S', H', C', b' \rangle = \langle T + 1, S, H, C', b' \rangle \quad (H' \neq C')
\]
Jigsaw puzzles, Machines

\[ \langle T, S, H, C, b \rangle: \text{“At time } T, \text{ machine is in state } S, \text{ the head is at cell } H, \text{ and cell } C \text{ holds symbol } b \rangle \]

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:
\[
\delta \langle T, S, H, C, b \rangle \langle T', S', H', C, b' \rangle = \langle \text{some goofy null value} \rangle
\]
\( (T \neq T' \text{ and } T \neq T' + 1) \)
The real deal

Setting up initial ID, iterator, and test:

\[(\lambda f_1. (f_1 \ 0)(f_1 \ 1))\]
\[(\lambda z_1.\]
\[(\lambda f_2. (f_2 \ 0)(f_2 \ 1))\]
\[(\lambda z_2.\]
\[\ldots\]
\[(\lambda f_N. (f_N \ 0)(f_N \ 1))\]
\[(\lambda z_N.\]

\[\text{(let } \Phi = \text{coding of transition function of TM in \ Widget[Extract}(Y \ \Phi (\lambda w. w \ 0 \ldots 0 \ Q_0 \ H_0 \ z_1 z_2 \ldots z_N \ 0))))\ldots))))\]
\[\langle T, S, H, \quad C, b \rangle\]
The real deal

...let $\Phi = \text{coding of transition function of } TM$ in

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

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

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

... let $\Phi = \text{coding of transition function of TM in}$

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

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

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

$(\phi x_1 x_2 \cdots x_m y_1 y_2 \cdots y_m))))$

$\text{Widget}[E] \equiv \ldots f \ldots a \ldots$, where $a$ flows as an argument to $f$

iff a True value flows out of $E$. 
The real deal

...let $\Phi = \text{coding of transition function of TM in}$

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

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

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

$$(\phi x_1 x_2 \ldots x_m y_1 y_2 \ldots y_m))))$$

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

**Theorem** In $k$CFA, $a$ flow to $f$ iff TM accept in $2^n$ steps.
The real deal

...let \( \Phi = \text{coding of transition function of TM in} \)
\[
\text{Widget}[\text{Extract}(Y \Phi (\lambda w.w 0 \ldots 0 Q_0 H_0 z_1 z_2 \ldots z_N 0))] \ldots
\begin{array}{c}
\langle T, S, H, C, b \rangle
\end{array}
\]

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

\[
\text{Widget}[E] \equiv \ldots f \ldots a \ldots, \text{ where } a \text{ flows as an argument to } f
\]
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 $k$CFA?
  — Linearity and precision
  — Non-linearity and an exponential iterator
★ Simulating exponential Turing machines with $k$CFA
★ Conclusions
What makes $k$CFA hard?

This is not just a replaying of the previous proofs.
What makes $\kappa$CFA hard?

This is not just a replaying of the previous proofs.

⋆ If the analysis were simulating evaluation,
What makes $k$CFA 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 \( k \)CFA 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 $k$CFA 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.
What makes $k$CFA 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 $k$CFA is the junk (spurious flows).
Doggie bag

- There is no tractable algorithm for $k$CFA
- Linearity is key in understanding static analysis
- The approximation of $k$CFA is what makes it hard
The End

Thank you.