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.

- Why kCFA as the subject of the investigation?
  "Some form of CFA is used in most forms of analyses for higher-order languages."

  (Heintze and McAllester 1997)

Introduction

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

- Why a complexity theoretic investigation?

  "There is an important analogy to complexity theory here. [. . .] The notions of logspace reduction between problems and of NP-complete problems lead to realising that problems may appear unrelated at first sight and nonetheless be so closely related that a good algorithm or heuristics for one will give rise to a good algorithm or heuristics of the other."

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

  (Nielson et al. 1999)

Outline

- Part 0: 0CFA
  - What is Control Flow Analysis?
  - What is an exact analysis?
  - Symmetric boolean logic gates
  - PTIME-completeness of 0CFA
- Part G: A Graphical View of CFA
- Part k: kCFA
  - What is kCFA?
  - Approximation and non-determinism
  - NP-hardness of kCFA
- Part n: nCFA
  - EXPTIME-completeness of nCFA

CFA Primer

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

A fundamental notion of CFA has emerged – the monovariant form of CFA defined over the pure lambda calculus.

(Heintze and McAllester 1997)

Part 0: 0CFA

Preliminaries: the Language

The language:

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

$t ::= x \mid (e\ e) \mid (\lambda x.e)$ 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: the Analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps a label to a set of terms.

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

$$\widehat{\mathsf{C}}(\ell) = \{(\lambda y.\cdots), (\lambda z.\cdots)\}$$

Preliminaries: the Analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps a label to a set of terms.

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

$$\widehat{\mathsf{C}}(\ell) = \{\lambda y, \lambda z\}$$ shorthand

Preliminaries: the Analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps a label to a set of terms.

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

$$\widehat{\mathsf{C}}(x) = \{\lambda y, \lambda z\}$$ overloading

0CFA Acceptability

$$
\begin{aligned}
\widehat{\mathsf{C}} &\models x^{\ell} \text{ iff } \widehat{\mathsf{C}}(x) \subseteq \widehat{\mathsf{C}}(\ell) \\
\widehat{\mathsf{C}} &\models (\lambda x.e)^{\ell} \text{ iff } (\lambda x.e) \in \widehat{\mathsf{C}}(\ell) \\
\widehat{\mathsf{C}} &\models (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell} \text{ iff } \widehat{\mathsf{C}} \models t_1^{\ell_1} \wedge \widehat{\mathsf{C}} \models t_2^{\ell_2} \wedge \\
&\qquad \forall (\lambda x.t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \\
&\qquad\quad \widehat{\mathsf{C}} \models t_0^{\ell_0} \wedge \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(x) \wedge \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell)
\end{aligned}
$$

We are concerned only with *least* analyses according to the partial order:

$$
\widehat{\mathsf{C}} \sqsubseteq_e \widehat{\mathsf{C}}' \quad \text{iff} \quad \forall \ell \in \mathbf{Lab}_e : \widehat{\mathsf{C}}(\ell) \subseteq \widehat{\mathsf{C}}'(\ell)
$$

Decision problem

Control Flow Problem (0CFA): Given a term and a label ℓ, does that term flow into program point ℓ?

$$\lambda x \in \widehat{\mathsf{C}}(\ell)?$$

Exact analysis

An analysis is a table $\widehat{\mathsf{C}}$ that maps labels to sets of terms.

$$\widehat{\mathsf{C}}(\ell) = \{\lambda y, \lambda z\}$$

The term labeled ℓ evaluates to either the term λy, or λz.

Exact analysis

An *exact* analysis is a table $\widehat{\mathsf{C}}$ that maps labels to *a* term.

$$\widehat{\mathsf{C}}(\ell) = \{\lambda y\}$$

The term labeled ℓ evaluates to either the term λy.

Exact analysis

An *exact* analysis is a table $\widehat{\mathsf{C}}$ that maps labels to *a* term.

$$\widehat{\mathsf{C}}(\ell) = \{\lambda y\}$$

The term labeled ℓ evaluates to either the term λy.

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

Exact analysis

An *exact* analysis is a table $\widehat{\mathsf{C}}$ that maps labels to *a* term.

$$\widehat{\mathsf{C}}(\ell) = \{\lambda y\}$$

The term labeled ℓ evaluates to either the term λy.

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

Exact analysis is Normalization.

Evaluator (exact, $k = 0$)

$\mathcal{E}[\![t^\ell]\!]$ Evaluate the term t, and write the result into $\widehat{\mathsf{C}}(\ell)$:

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

This is an evaluator, but for what language?

Evaluator (exact, $k = 0$)

$\mathcal{E}[\![t^\ell]\!]$ Evaluate the term t, and write the result into $\widehat{\mathsf{C}}(\ell)$:

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

This is an evaluator, but for what language?

The linear λ-calculus.

Normalization in Linear λ-calculus

0CFA of linear λ-terms is just normalization, so a lower-bound on the complexity is the expressiveness of linear λ.

Normalization of a linear λ-term is complete for PTIME*.*
(Mairson 2004)

Hardness follows from reduction of CVP to normalization:

Circuit Value Problem: Given a Boolean circuit C of n inputs and one output, and truth values $\vec{x} = x_1, \ldots, x_n$, is $\vec{x}$ accepted by C?

Given a circuit C, compile it (using LOGSPACE) into a linear λ-term ϕ s.t. $\phi\vec{x} = \texttt{True}$ iff $\vec{x}$ is accepted by C.

Symmetric logic gates

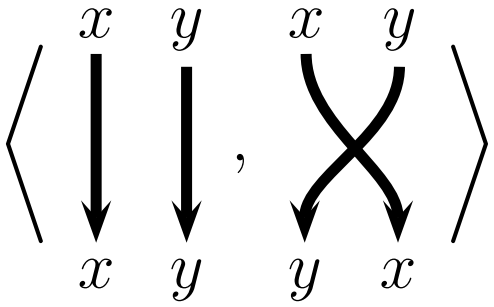
```
- fun TT(x:'a,y:'a)= (x,y);
val TT= fn : 'a * 'a -> 'a * 'a
- fun FF(x:'a,y:'a)= (y,x);
val FF= fn : 'a * 'a -> 'a * 'a
```

Booleans built out of constants TT, FF

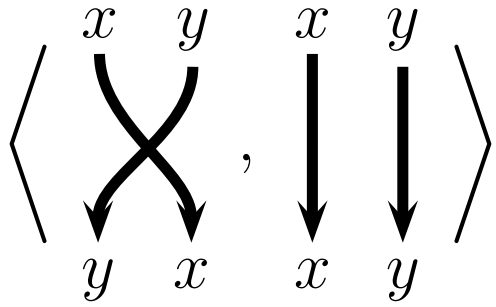
```
- val True= (TT: ('a * 'a -> 'a * 'a),  FF: ('a * 'a -> 'a * 'a));
val True = (fn,fn) : ('a * 'a -> 'a * 'a) * ('a * 'a -> 'a * 'a)

- 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=



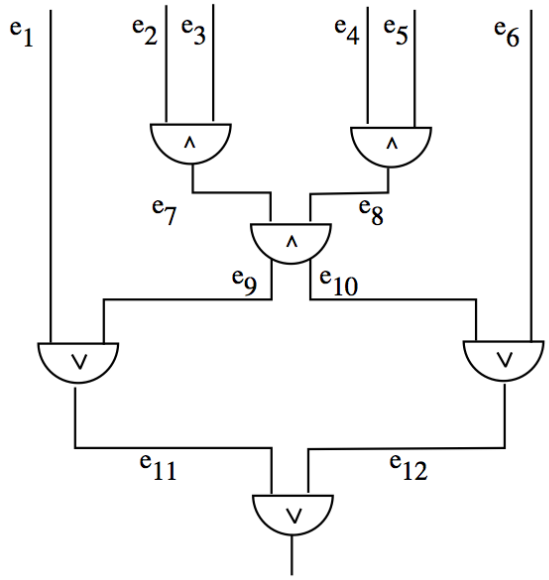
To *twist*, or not to *twist*: that is the question.

Symmetric logic gates

```
- fun Andgate p q k= ···
- fun Orgate p q k= ···
- fun Notgate p k= ···
- fun Copygate p k= ···
```

linear ML terms

(Mairson 2004)




```
- fun Circuit e1 e2 e3 e4 e5 e6=
  (Andgate e2 e3 (fn e7=>
  (Andgate e4 e5 (fn e8=>
  (Andgate e7 e8 (fn f=>
  (Copygate f (fn (e9,e10)=>
  (Orgate e1 e9 (fn e11=>
  (Orgate e10 e6 (fn e12=>
  (Orgate e11 e12 (fn Output=> Output))))))))))));
val Circuit = fn : < big type... >
```

The Widget

Let $E =$

```
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 $E[\phi\vec{x}]$: Does a flow as an argument to f?

- Clearly, a flows as an argument to x.
- So does f flow into x?
- (f,g) flows into (x,y) iff TT flows into u.
- TT flows into u iff $\phi\vec{x} =$ True.
- Either $\widehat{\mathsf{C}}(\mathsf{x}) = \{\mathsf{f}\}$ or $\widehat{\mathsf{C}}(\mathsf{x}) = \{\mathsf{g}\}$, but not $\widehat{\mathsf{C}}(\mathsf{x}) = \{\mathsf{f}, \mathsf{g}\}$

Evaluator (inexact, $k = 0$)

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

- Read $\widehat{\mathsf{C}}(\ell) \leftarrow \{\lambda x, \lambda y\}$ as $\widehat{\mathsf{C}}(\ell) := \widehat{\mathsf{C}}(\ell) \cup \{\lambda x, \lambda y\}$.
- Many terms may flow into an application: apply them all.
- Iterate $\mathcal{E}$ until the table reaches a fixed point.

PTIME Completeness of 0CFA

- Hardness: LOGSPACE-reducible to Circuit Value.
- Inclusion: 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

Part G: A Graphical View of CFA

An alternative view

The use of graphical constructs in analysis is becoming increasingly common.

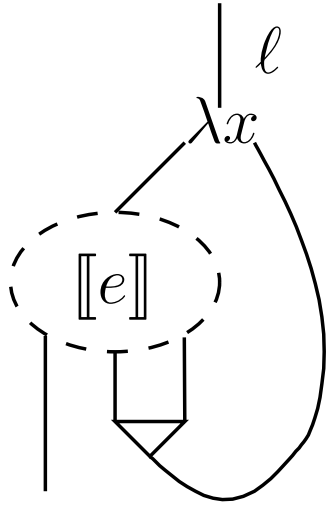
(Heintze and McAllester 1997)

- Graph based algorithm using “sharing graphs”.
- Draws connection between CFA and linear logic.
- Insights from linear logic lead to LOGSPACE CFA algorithms for a restricted set of programs.
- Connection between normalization and CFA becomes transparent.
- Accommodates languages with first-class control in a direct manner.

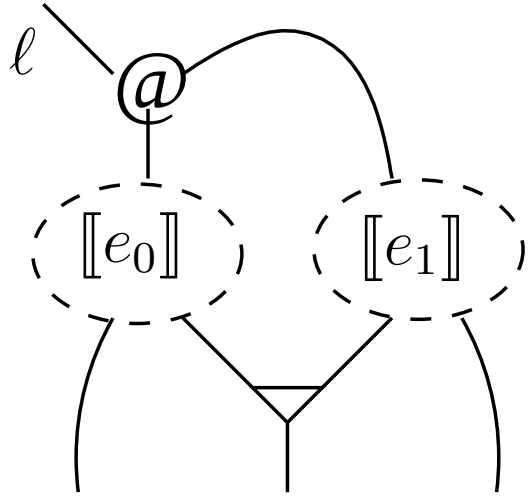
Graph Codings

A graphical representation of expressions:

$[\![x^\ell]\!]$

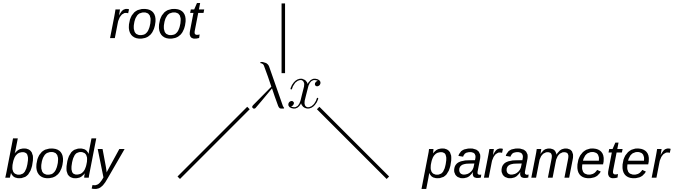


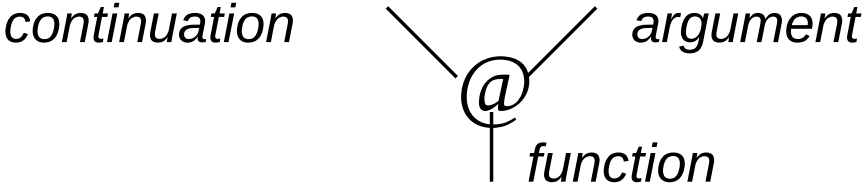
$[\![(\lambda x.e)^\ell]\!]$



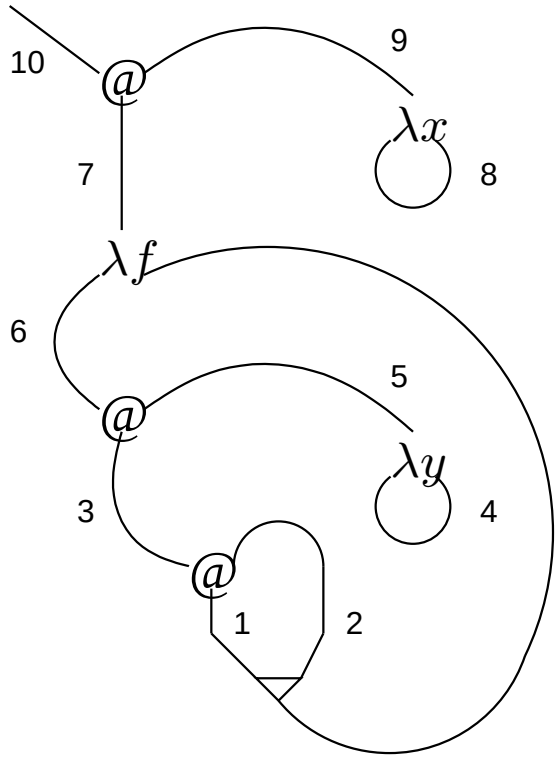
$[\![(e_0\ e_1)^\ell]\!]$

Port names, orientation:





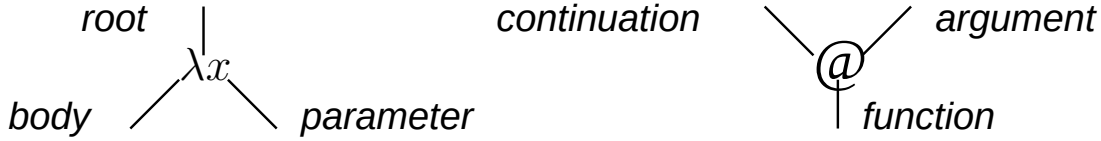
Graph coding example



Idea: *formulate a graph based CFA.*

- Add paths corresponding to flows, ie.
  $\widehat{\mathsf{C}}(\ell) \subseteq \widehat{\mathsf{C}}(\ell') \Rightarrow PATH(\ell, \ell')$.
- A λ flows into ℓ iff there is a path from ℓ to λ (root).

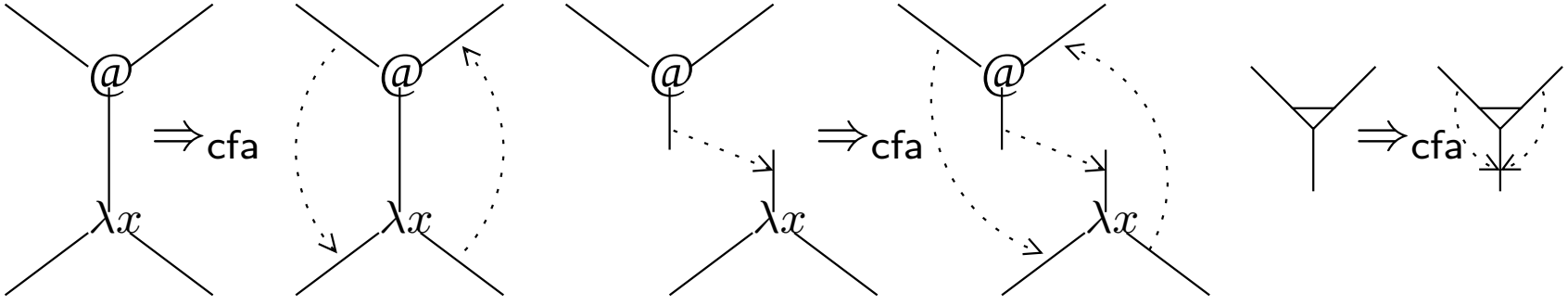
Don't forget:



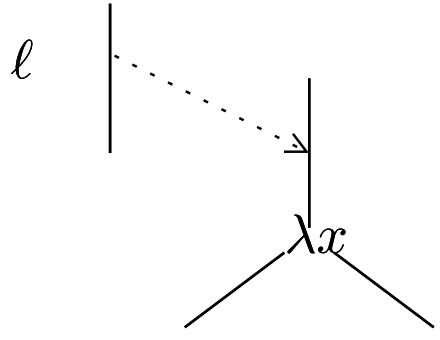
$$((\lambda f.((f^1 f^2)^3(\lambda y.y^4)^5)^6)^7(\lambda x.x^8)^9)^{10}$$

CFA Graphs

CFA Graphs are constructed by adding edges:

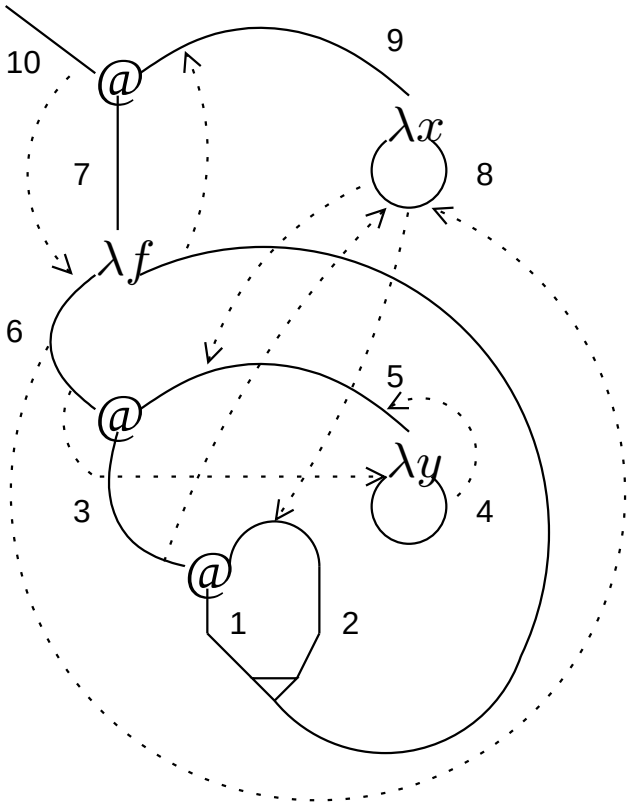


this rule is implicit



$\lambda x \in \widehat{\mathsf{C}}(\ell)$ iff

CFA example



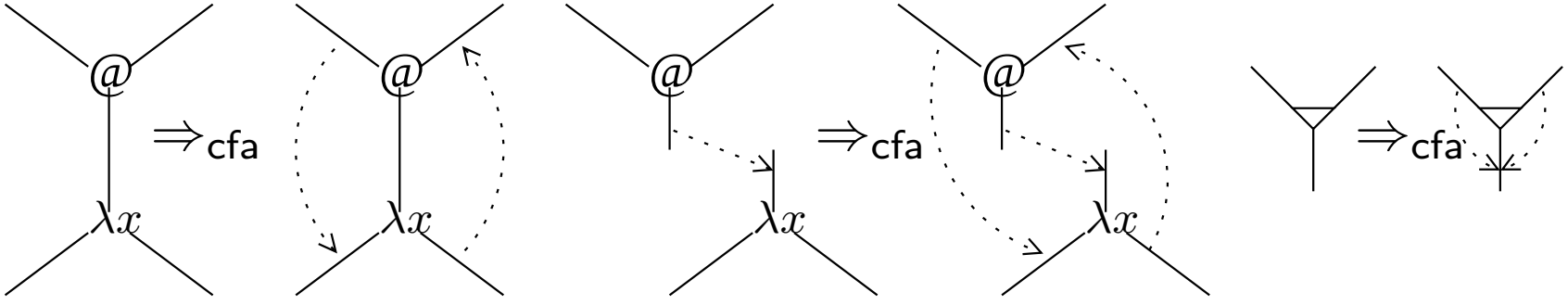
- Paths correspond to flows, ie. $\widehat{\mathsf{C}}(\ell) \subseteq \widehat{\mathsf{C}}(\ell') \Rightarrow PATH(\ell, \ell')$.
- A λ flows into ℓ iff there is a path from ℓ to λ (root).

$$\widehat{\mathsf{C}}(10) = \{\lambda x, \lambda y\}$$

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

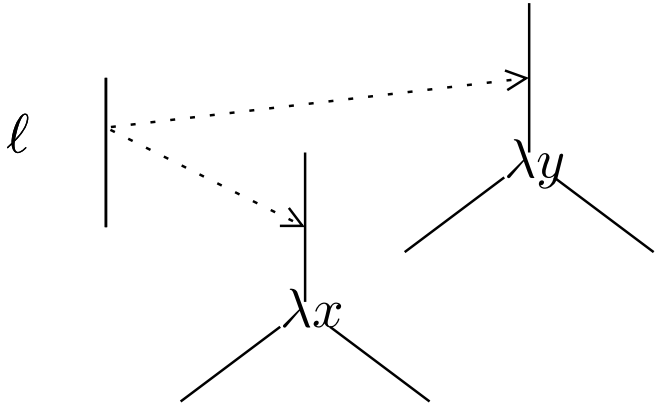
Approximation in CFA Graphs

CFA Graphs are constructed by adding edges:

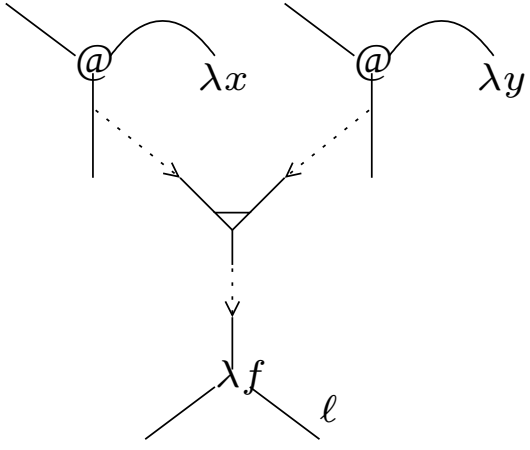


this rule is implicit

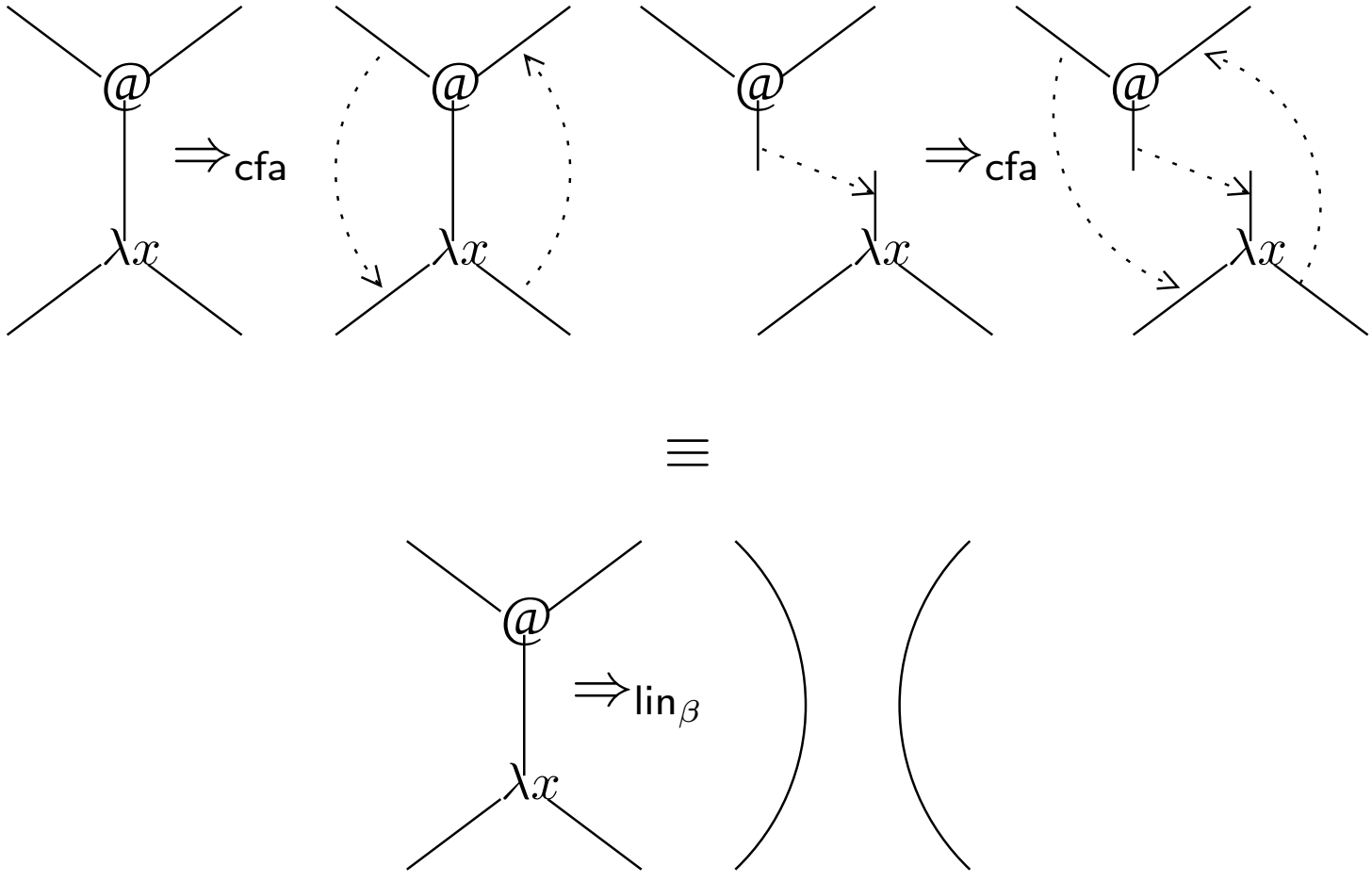
When is an analysis inexact?



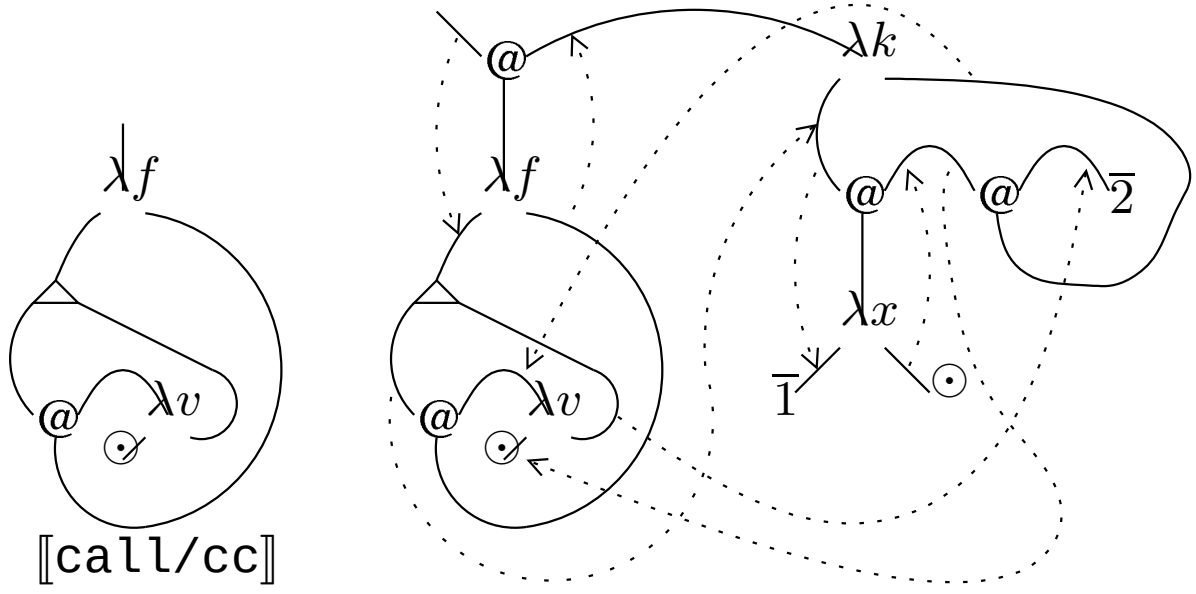
When does this occur?



CFA as normalization of linear λ-terms



CFA for languages with `call/cc`



$$(\texttt{call/cc}\ \ (\lambda k.((\lambda x.\overline{1})\ \overline{2})))^{\ell}$$

$$\widehat{\mathsf{C}}(\ell) = \{\overline{1}, \overline{2}\}$$

Connections to Linear Logic

- Graph codings are just the proofnets of MELL (without boxes, brackets, or crossaints).
- Graph codings of linear terms are just MLL proofnets.
- MLL normalization algorithms can give insights into CFA algorithms.
- CFA paths are the “well-balanced” paths of (Asperti and Laneve 1995).

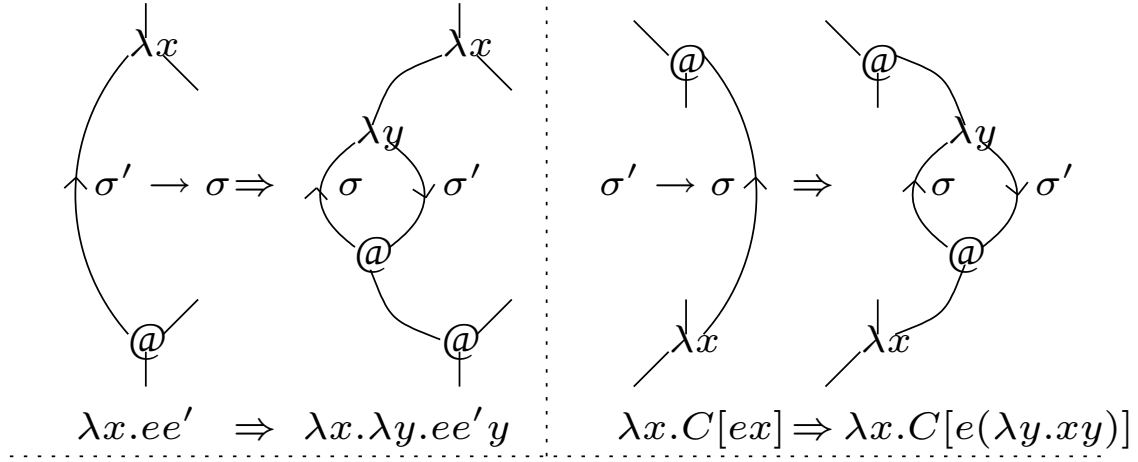
LOGSPACE and η-expansion

0CFA of simply-typed, η-expanded programs is complete for LOGSPACE.

$$\frac{}{A \otimes B, A^{\perp} \parr B^{\perp}} \quad \Rightarrow \quad \frac{\dfrac{\dfrac{}{A, A^{\perp}} \quad \dfrac{}{B, B^{\perp}}}{A \otimes B, A^{\perp}, B^{\perp}}}{A \otimes B, A^{\perp} \parr B^{\perp}}$$

LOGSPACE and η-expansion

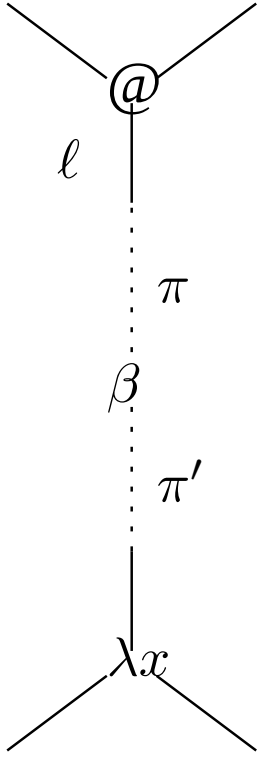
0CFA of simply-typed, η-expanded programs is complete for LOGSPACE.



LOGSPACE and η-expansion

0CFA of simply-typed, η-expanded programs is complete for LOGSPACE.

If $\lambda x \in \widehat{\mathsf{C}}(\ell)$, then:

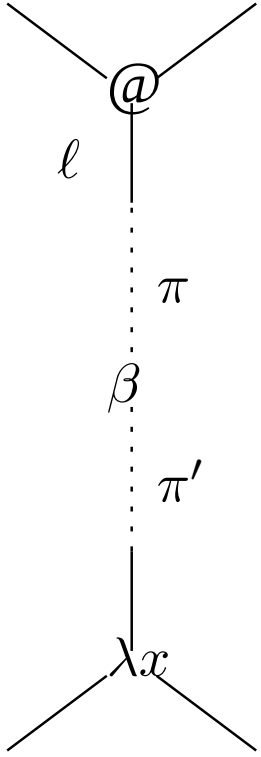


0CFA$_\eta$ is in LOGSPACE

LOGSPACE and η-expansion

0CFA of simply-typed, η-expanded programs is complete for LOGSPACE.

If $\lambda x \in \widehat{\mathsf{C}}(\ell)$, then:



Normalization (of linear terms) is LOGSPACE-hard (Terui 2002; Mairson 2006).

0CFA$_\eta$ is LOGSPACE-complete

Part k: kCFA

kCFA

"It did not take long to discover that the basic analysis, for any $k > 0$, was intractably slow for large programs."

(Shivers 2004)

Preliminaries: Contours

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

$$\delta \in \Delta = \mathbf{Lab}^{\leq k}$$

$$ce \in \mathbf{CEnv} = \mathbf{Var} \to \Delta$$

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.

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

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

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.

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

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

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

Acceptability

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

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

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:

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

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

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

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

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 $ce'_0 = ce_0[x \mapsto \delta_0]$

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.

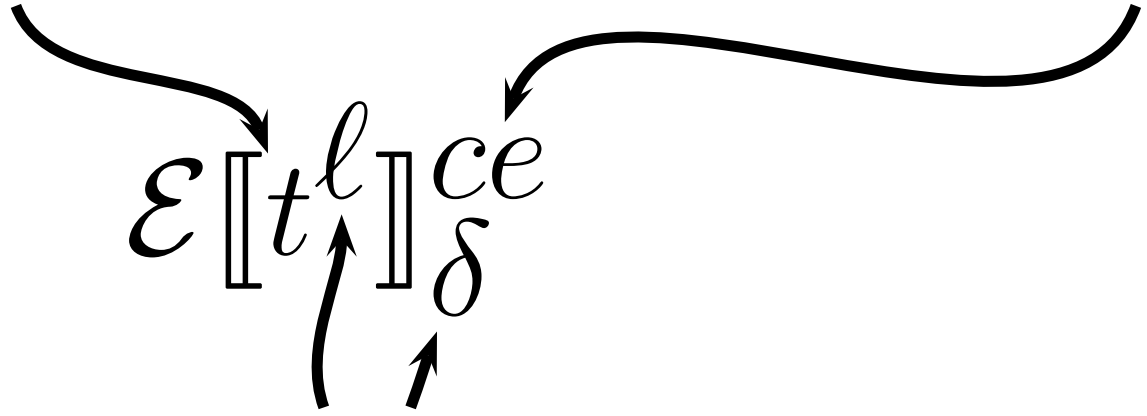
$$\widehat{\mathsf{C}}(\ell, \delta) = \{\langle \lambda y, ce \rangle\}$$

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.



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)

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

Mr. Natural: Exact analysis is normalization.

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

Evaluator (inexact)

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

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

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).

Exactness and complexity

Hardness of 1CFA 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.
   A less precise analysis "yields coarser approximations, and thus induces more merging. More merging leads to more propagation, which in turn leads to more reevaluation."

   (Wright and Jagannathan 1998)

1CFA as SAT solver

$$
\begin{array}{l}
(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False})) \\
(\lambda x_1. \\
\quad (\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False})) \\
\quad (\lambda x_2. \\
\quad\quad (\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False})) \\
\quad\quad (\lambda x_3. \\
\quad\quad\quad \cdots \\
\quad\quad\quad (\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False})) \\
\quad\quad\quad (\lambda x_n. \\
\quad\quad\quad\quad E[(\lambda v.\phi\ v)(\lambda w.wx_1x_2\cdots x_n)])\cdots))))
\end{array}
$$

1CFA as SAT solver

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

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

1CFA as SAT solver

$$
\begin{array}{l}
(\lambda f_1.(f_1\ \texttt{True})(f_1\ \texttt{False})) \\
(\lambda x_1. \\
\quad (\lambda f_2.(f_2\ \texttt{True})(f_2\ \texttt{False})) \\
\quad (\lambda x_2. \\
\quad\quad (\lambda f_3.(f_3\ \texttt{True})(f_3\ \texttt{False})) \\
\quad\quad (\lambda x_3. \\
\quad\quad\quad \cdots \\
\quad\quad\quad (\lambda f_n.(f_n\ \texttt{True})(f_n\ \texttt{False})) \\
\quad\quad\quad (\lambda x_n. \\
\quad\quad\quad\quad E[(\lambda v.\phi\ v)(\lambda w.wx_1x_2\cdots x_n)])\cdots))))
\end{array}
$$

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

1CFA as SAT solver

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

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

1CFA as SAT solver

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

Approximation of closures as non-deterministic computation!

The Widget, Again

$E =$

```
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 $E[(\lambda v.\phi\ v)(\lambda w.wx_1x_2\cdots x_n)]$:

f is applied to a iff ϕ is satisfiable.

1CFA is NP-hard

The Widget, Again

$E =$

```
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 $E[(\lambda v.\phi\ v)(\lambda w.wx_1x_2\cdots x_n)]$:

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 monotonic, 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?

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

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:

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

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.

Preprint Available, Comments Welcome

Relating Complexity and Precision in Control Flow Analysis

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

The End

MLL and (linear) functional programs

Curry-Howard for linear λ-calculus (LML) $=$ MLL:

$$\textsc{Ax}\ \frac{}{A, A^{\perp}} \qquad \textsc{Cut}\ \frac{\Gamma, A \quad A^{\perp}, \Delta}{\Gamma, \Delta} \qquad \parr\ \frac{\Gamma, A, B}{\Gamma, A \parr B} \qquad \otimes\ \frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B}$$

- $\otimes$ is a linear pairing of
  - expressions ($cons$)
  - an expression and continuation (@)
- $\parr$ is a linear unpairing of
  - an expression (π, π')
  - an expression and continuation (λ).

MLL and (linear) functional programs

Curry-Howard for linear λ-calculus (LML) $=$ MLL:

$$\textsc{Ax}\ \frac{}{A, A^{\perp}} \qquad \textsc{Cut}\ \frac{\Gamma, A \quad A^{\perp}, \Delta}{\Gamma, \Delta} \qquad \parr\ \frac{\Gamma, A, B}{\Gamma, A \parr B} \qquad \otimes\ \frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B}$$

Note $A \multimap B = A^{\perp} \parr B$, $A^{\perp\perp} = A$, $A^{\perp}$ on left is like A in right, etc.

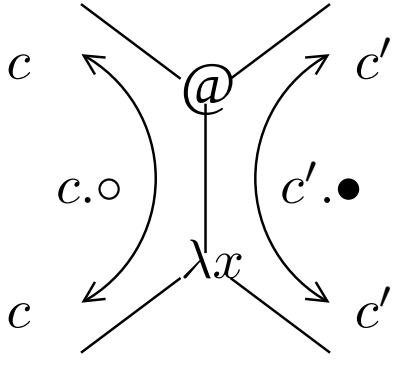
$$\frac{}{x : A \vdash x : A} \qquad \frac{\Gamma \vdash E : A \quad \Delta \vdash F : B}{\Gamma, \Delta \vdash (E, F) : A \otimes B}$$

$$\frac{\Gamma \vdash F : A \otimes B \quad \Delta, x : A, y : B \vdash E : \sigma}{\Gamma, \Delta \vdash \texttt{let}\ (x, y) = F\ \texttt{in}\ E : \sigma} \qquad \frac{\Gamma, x : A \vdash E : B}{\Gamma \vdash \lambda x.E : A \multimap B}$$

$$\frac{\Gamma \vdash E : A \multimap B \quad \Delta \vdash F : B}{\Gamma, \Delta \vdash (E\ \ F) : B}$$

Geometry of Interaction

Normalization by GoI for MLL (LML):
Hilbert to Dilbert (Mairson 2002)



$\Sigma = \{\bullet, \circ\}$ tokens $\quad c \in \Sigma^*$ contexts

Eta expansion

$$\frac{}{A \otimes B, A^{\perp} \parr B^{\perp}} \quad \Rightarrow \quad \frac{\dfrac{\dfrac{}{A, A^{\perp}} \quad \dfrac{}{B, B^{\perp}}}{A \otimes B, A^{\perp}, B^{\perp}}}{A \otimes B, A^{\perp} \parr B^{\perp}}$$

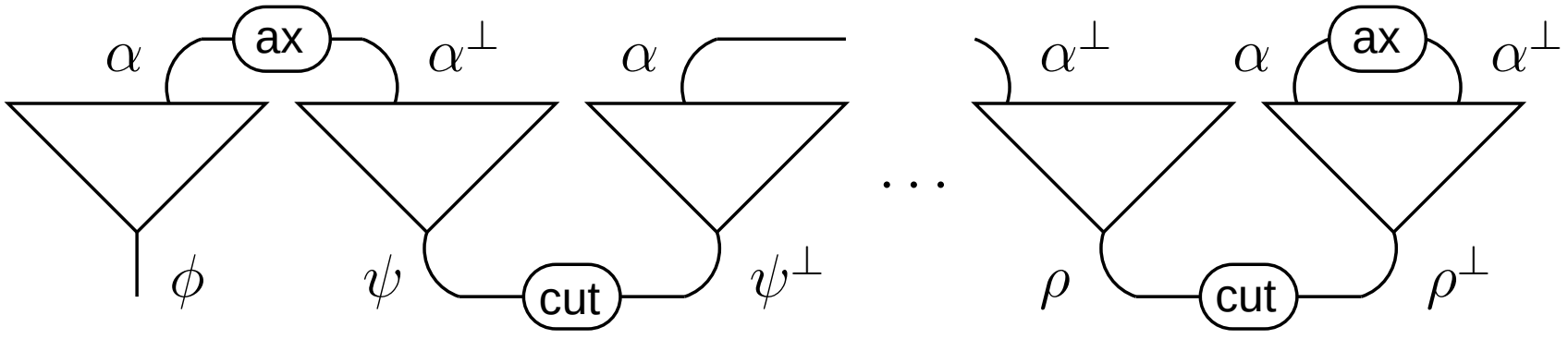
Axioms are atomic (α is a propositional variable):

$$\text{Ax}\ \frac{}{\alpha, \alpha^{\perp}}$$

What does this mean for GoI?

Stack is empty on all axiom wires.

Four finger



Stack is empty on all axiom wires $\Rightarrow$ No need for stack!

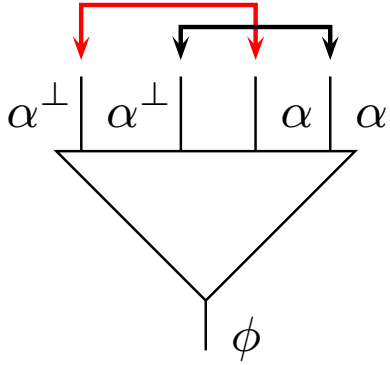
Computing Linear Normal Forms

Formula ϕ identifies the normal form up to placement of axiom wires at the same type, eg:

$$e : \sigma = \alpha \to \alpha \to (\alpha \to \alpha \to \beta) \to \beta$$

$$\phi = \alpha^{\perp} \parr (\alpha^{\perp} \parr (((\beta^{\perp} \otimes \alpha) \otimes \alpha) \parr \beta))$$

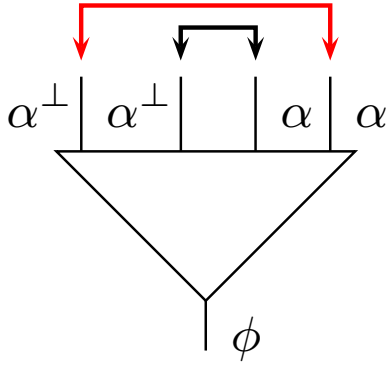
$$NF(e) : \sigma = \lambda x : \alpha . \lambda y : \alpha . \lambda z : \alpha \to \alpha \to \beta . zxy$$



or...

$$\phi = \alpha^{\perp} \parr (\alpha^{\perp} \parr (((\beta^{\perp} \otimes \alpha) \otimes \alpha) \parr \beta))$$

$$NF(e) : \sigma = \lambda x : \alpha . \lambda y : \alpha . \lambda z : \alpha \to \alpha \to \beta . zyx$$



Remember: $A \multimap B = A^{\perp} \parr B$, $A^{\perp\perp} = A$, $A^{\perp}$ ...

Symmetric garbage is self-annihilating

And (p,p′) (q,q′) ≡ (p∧q, p′∨q′) ≡ (p∧q, ¬(p′∧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

Citations

References

Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theor. Comput. Sci.*, 142(2):277–297, 1995.

Nevin Heintze and David McAllester. Linear-time subtransitive control flow analysis. In *PLDI '97: Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation*, pages 261–272. ACM Press, 1997.

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. Springer-Verlag, 2002.

Harry G. Mairson. Linear lambda calculus and PTIME-completeness. *Journal of Functional Programming*, 14(6):623–633, 2004.

Harry G. Mairson. Axiom-sensitive normalization bounds for multiplicative linear logic, 2006. Unpublished manuscript.

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

Olin Shivers. Higher-order control-flow analysis in retrospect: lessons learned, lessons abandoned. *SIGPLAN Not.*, 39(4): 257–269, 2004.

Kazushige Terui. On the complexity of cut-elimination in linear logic, July 2002. Invited talk at LL2002 (LICS2002 affiliated workshop), Copenhagen.

Andrew K. Wright and Suresh Jagannathan. Polymorphic splitting: an effective polyvariant flow analysis. *ACM Trans. Program. Lang. Syst.*, 20(1):166–207, 1998.