Flow Analysis, Linearity and PTIME

David Van Horn and Harry Mairson

What is a lower bound?

Higher-order control-flow analysis in retrospect: Lessons learned, lessons abandoned. Shivers, Best of PLDI retrospective:

“In the ensuing years, researchers have expended a great deal of effort deriving clever ways to tame the cost of the analysis.”

But a fundamental questions remains:

To what extent is this possible?

Lower bounds on AI

Observation. *For any (useful) abstract interpretation, there exists a class of programs on which AI corresponds with concrete interpretation.*

Programming within this class *may* lead to interesting lower bounds.

Goal

Give a transparent proof of the correspondence between evaluation and flow analysis for linear programs.

Outline:

- Inspirations
- Flow analysis and linearity
- Consequences

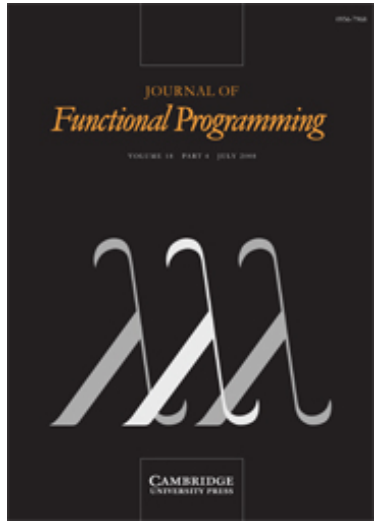
Inspirations

Inspiration

Linear lambda calculus and PTIME-completeness, Mairson, JFP Special Issue on Functional Pearls.

The **normal form** of a linear λ-term is isomorphic to its **most general type**.

- Simple typing: surely a kind of static analysis.
- What happens with other kinds?



Linearity and simple types

When a program is linear, *there is no approximation*.

- approximation: $\lambda x : \sigma. \ldots x \ldots x \ldots$
- *both* occurrences of x must be of type σ
- *linearity* avoids the approximation

$$\begin{aligned} T &: \alpha \to \beta \to \alpha \\ F &: \alpha \to \beta \to \beta \\ (Not\ T) &: \alpha \to \beta \to \beta \equiv F \end{aligned}$$

The same is true for flow analyses (roughly).

Inspiration, II

Control-flow Analysis of Functional Programs, Midtgaard, ACM Computing Surveys. (To appear. BRICS tech. report RS-07-18.)

Flow analysis **abound**. (170+ citations and growing).

- Flow analysis: surely a kind of static analysis.
- What happens in face of linearity?

Outline

- Inspirations ✓
- Flow analysis and linearity
- Consequences

Flow analysis and linearity

What is flow analysis?

An approximation to operational behavior of programs.

$$\texttt{let } f \texttt{ = } \lambda x.x \texttt{ in } (f\,f)(\lambda y.y)$$

- f may be bound to $\lambda x.x$.
- x may be bound to $\lambda y.y$ or $\lambda x.x$.
- $(f\,f)$ may eval to $\lambda y.y$ or $\lambda x.x$.
- ...
- program may eval to $\lambda y.y$ or $\lambda x.x$.

A typical example showing the *imprecision* of analysis.

Abstract cache

We have a simple language with labelled syntax:

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

$t ::= x \mid ee \mid \lambda x.e$ terms (or unlabeled expressions)

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

- $\widehat{\mathsf{C}}(\ell) = \{\lambda x, \ldots\}$ "subexpression ℓ may eval to $\lambda x, \ldots$"
- $\widehat{\mathsf{C}}(y) = \{\lambda x, \ldots\}$ "variable y may be bound to $\lambda x, \ldots$"

Abstraction of evaluation

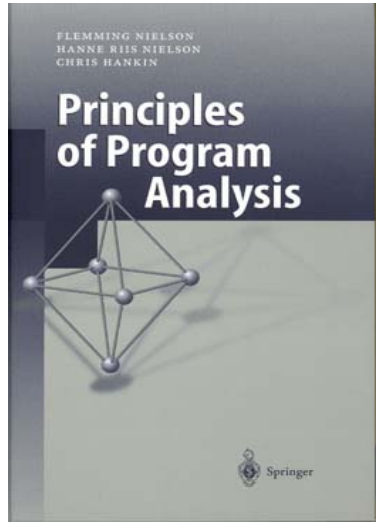
$$
\begin{array}{lcl}
\mathcal{E}[\![x]\!]\rho & = & \rho(x) \\
\mathcal{E}[\![\lambda x.e]\!]\rho & = & \langle \lambda x.e, \rho \restriction \mathbf{fv}(e) \rangle \\
\mathcal{E}[\![e_1 e_2]\!]\rho & = & \textsf{let } \langle \lambda x.e_0, \rho' \rangle = \mathcal{E}[\![e_1]\!] \textsf{ in} \\
 & & \textsf{let } c = \mathcal{E}[\![e_2]\!] \textsf{ in} \\
 & & \quad \mathcal{E}[\![e_0]\!]\rho'[x \mapsto c]
\end{array}
$$

$$
\mathcal{E}[\![P^\ell]\!] = \langle \lambda x.e, \rho \rangle \Longrightarrow \lambda x.e \in \widehat{\mathsf{C}}(\ell)
$$

A typical specification

$\widehat{\mathsf{C}} \models P$ "$\widehat{\mathsf{C}}$ is a valid analysis of P"

$$
\begin{array}{lcl}
\widehat{\mathsf{C}} \models x^{\ell} & \Longleftrightarrow & \widehat{\mathsf{C}}(x) \subseteq \widehat{\mathsf{C}}(\ell) \\
\widehat{\mathsf{C}} \models (\lambda x.e)^{\ell} & \Longleftrightarrow & \lambda x.e \in \widehat{\mathsf{C}}(\ell) \\
\widehat{\mathsf{C}} \models (t^{\ell_1}\ t^{\ell_2})^{\ell} & \Longleftrightarrow & \widehat{\mathsf{C}} \models t^{\ell_1} \wedge \widehat{\mathsf{C}} \models t^{\ell_2} \\
 & & \wedge \forall \lambda x.t^{\ell_0} \in \widehat{\mathsf{C}}(\ell_1) : \\
 & & \quad \widehat{\mathsf{C}} \models t^{\ell_0} \wedge \\
 & & \quad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(x) \wedge \\
 & & \quad \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell)
\end{array}
$$



Single and loving it

Jagannathan, et al. POPL 98

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

"Subexpression ℓ ~~may~~ *must* eval to λx."

(An idea also used by Might and Shivers, ICFP 06, "abstract counting".)

Staying single

Key to staying single: *linearity* (1 is the loneliest number).

Each variable occurs at most once, so each abstraction $\lambda x.e$ can be applied to at most one argument, and the abstracted value can be bound to at most one argument.

Theorem. *If* $\widehat{\mathsf{C}} \models e$ *and* e *is linear,* $\widehat{\mathsf{C}}$ *is a* complete *description of running the program.*

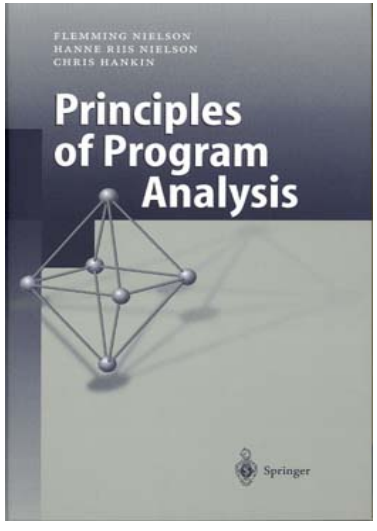
$$\begin{array}{l}\texttt{let } f_1 \texttt{ = } \lambda x_1.x_1 \texttt{ in} \\ \texttt{let } f_2 \texttt{ = } \lambda x_2.x_2 \texttt{ in} \\ \quad (f_1 f_2)(\lambda y.y)\end{array}$$

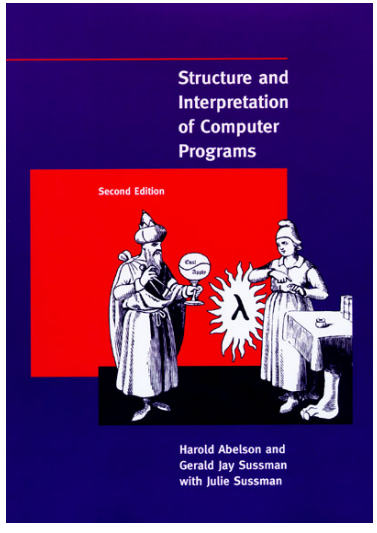
0CFA $\equiv \mathcal{E}[\![\cdot]\!]$?

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

$$
\begin{array}{lcl}
\mathcal{E}[\![x]\!]\rho & = & \rho(x) \\
\mathcal{E}[\![\lambda x.e]\!]\rho & = & \langle \lambda x.e, \rho \restriction \mathbf{fv}(e) \rangle \\
\mathcal{E}[\![e_1 e_2]\!]\rho & = & \textbf{let } \langle \lambda x.e_0, \rho' \rangle = \mathcal{E}[\![e_1]\!] \textbf{ in} \\
 & & \textbf{let } c = \mathcal{E}[\![e_2]\!] \textbf{ in} \\
 & & \quad \mathcal{E}[\![e_0]\!]\rho'[x \mapsto c]
\end{array}
$$

How can we reconcile superficial distinctions between these two?





The path ahead

1. Specialize the 0CFA algorithm for linear terms.
2. Derive an evaluator for linear terms.
3. Un-specialize to reveal typical evaluator.

The path ahead

1. Specialize the 0CFA algorithm for linear terms.
2. Derive an evaluator for linear terms.
3. Un-specialize to reveal typical evaluator.

Keep in mind:

1. Single and loving it
2. Every subterm is evaluated just once.
3. Every variable is bound just once.

An algorithm for 0CFA

General 0CFA algorithm: mutates (initially empty) cache.

$$
\begin{array}{lcl}
\mathcal{A}[\![x^\ell]\!] & = & \widehat{\mathsf{C}}[\ell \mapsto^+ \widehat{\mathsf{C}}(x)] \\
\mathcal{A}[\![(\lambda x.e)^\ell]\!] & = & \widehat{\mathsf{C}}[\ell \mapsto \{\lambda x.e\}] \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^\ell]\!] & = & \mathcal{A}[\![t^{\ell_1}]\!];\ \ \mathcal{A}[\![t^{\ell_2}]\!]; \\
 & & \quad \textbf{for each } \lambda x.t^{\ell_0} \textbf{ in } \widehat{\mathsf{C}}(\ell_1) \textbf{ do} \\
 & & \quad\quad \widehat{\mathsf{C}}[x \mapsto^+ \widehat{\mathsf{C}}(\ell_2)];\ \mathcal{A}[\![t^{\ell_0}]\!];\ \widehat{\mathsf{C}}[\ell \mapsto^+ \widehat{\mathsf{C}}(\ell_0)]
\end{array}
$$

- $\widehat{\mathsf{C}}[\ell \mapsto \hat{v}]$ means update $\widehat{\mathsf{C}}$ so $\widehat{\mathsf{C}}(\ell) = \hat{v}$
- $\widehat{\mathsf{C}}[\ell \mapsto^+ \hat{v}]$ means update $\widehat{\mathsf{C}}$ so $\widehat{\mathsf{C}}(\ell) = \hat{v} \cup \widehat{\mathsf{C}}(\ell)$

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!] & = & \widehat{\mathsf{C}}[\ell \mapsto^{+} \widehat{\mathsf{C}}(x)] \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!] & = & \widehat{\mathsf{C}}[\ell \mapsto \{\lambda x.e\}] \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!] & = & \mathcal{A}[\![t^{\ell_1}]\!];\ \ \mathcal{A}[\![t^{\ell_2}]\!]; \\
 & & \textbf{for each } \lambda x.t^{\ell_0} \textbf{ in } \widehat{\mathsf{C}}(\ell_1) \textbf{ do} \\
 & & \quad \widehat{\mathsf{C}}[x \mapsto^{+} \widehat{\mathsf{C}}(\ell_2)];\ \mathcal{A}[\![t^{\ell_0}]\!];\ \widehat{\mathsf{C}}[\ell \mapsto^{+} \widehat{\mathsf{C}}(\ell_0)]
\end{array}
$$

Observation:

- By singularity, the "**for each**" is unneeded.

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto^{+} \widehat{\mathrm{C}}(x)] \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \{\lambda x.e\}] \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!] & = & \mathcal{A}[\![t^{\ell_1}]\!];\ \ \mathcal{A}[\![t^{\ell_2}]\!]; \\
 & & \textbf{let } \{\lambda x.t^{\ell_0}\} = \widehat{\mathrm{C}}(\ell_1) \textbf{ in} \\
 & & \quad \widehat{\mathrm{C}}[x \mapsto^{+} \widehat{\mathrm{C}}(\ell_2)];\ \mathcal{A}[\![t^{\ell_0}]\!];\ \widehat{\mathrm{C}}[\ell \mapsto^{+} \widehat{\mathrm{C}}(\ell_0)]
\end{array}
$$

Observation:

- By singularity, the "**for each**" is unneeded.
- Again by singularity, the $\mapsto^{+}$ can be replaced with $\mapsto$.

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \widehat{\mathrm{C}}(x)] \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \{\lambda x.e\}] \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!] & = & \mathcal{A}[\![t^{\ell_1}]\!];\ \ \mathcal{A}[\![t^{\ell_2}]\!]; \\
 & & \textbf{let } \{\lambda x.t^{\ell_0}\} = \widehat{\mathrm{C}}(\ell_1) \textbf{ in} \\
 & & \quad \widehat{\mathrm{C}}[x \mapsto \widehat{\mathrm{C}}(\ell_2)];\ \mathcal{A}[\![t^{\ell_0}]\!];\ \widehat{\mathrm{C}}[\ell \mapsto \widehat{\mathrm{C}}(\ell_0)]
\end{array}
$$

Observation:

- By singularity, the "**for each**" is unneeded.
- Again by singularity, the $\mapsto^{+}$ can be replaced with $\mapsto$.

The path ahead

1. Specialize the 0CFA algorithm for linear terms. ✓
2. Derive an evaluator for linear terms.
3. Un-specialize to reveal typical evaluator.

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \widehat{\mathrm{C}}(x)] \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \{\lambda x.e\}] \\
\mathcal{A}[\![(t^{\ell_1}\, t^{\ell_2})^{\ell}]\!] & = & \mathcal{A}[\![t^{\ell_1}]\!];\ \ \mathcal{A}[\![t^{\ell_2}]\!]; \\
 & & \textbf{let } \{\lambda x.t^{\ell_0}\} = \widehat{\mathrm{C}}(\ell_1) \textbf{ in} \\
 & & \quad \widehat{\mathrm{C}}[x \mapsto \widehat{\mathrm{C}}(\ell_2)];\ \mathcal{A}[\![t^{\ell_0}]\!];\ \widehat{\mathrm{C}}[\ell \mapsto \widehat{\mathrm{C}}(\ell_0)]
\end{array}
$$

Observation:

- Values aren't returned; they're written into the cache.
- The cache is playing the role of the environment.

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \widehat{\rho}(x)] \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!] & = & \widehat{\mathrm{C}}[\ell \mapsto \{\lambda x.e\}] \\
\mathcal{A}[\![(t^{\ell_1}\, t^{\ell_2})^{\ell}]\!] & = & \mathcal{A}[\![t^{\ell_1}]\!];\ \ \mathcal{A}[\![t^{\ell_2}]\!]; \\
 & & \textbf{let } \{\lambda x.t^{\ell_0}\} = \widehat{\mathrm{C}}(\ell_1) \textbf{ in} \\
 & & \quad \widehat{\rho}[x \mapsto \widehat{\mathrm{C}}(\ell_2)];\ \mathcal{A}[\![t^{\ell_0}]\!];\ \widehat{\mathrm{C}}[\ell \mapsto \widehat{\mathrm{C}}(\ell_0)]
\end{array}
$$

Observation:

- Values aren't returned; they're written into the cache.
- The cache is playing the role of the environment.

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!] & = & \widehat{\rho}(x) \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!] & = & \lambda x.e \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!] & = & \textbf{let } \lambda x.t^{\ell_0} = \mathcal{A}[\![t^{\ell_1}]\!] \textbf{ in} \\
 & & \textbf{let } v = \mathcal{A}[\![t^{\ell_2}]\!] \textbf{ in} \\
 & & \quad \widehat{\rho}[x \mapsto v]; \\
 & & \quad \mathcal{A}[\![t^{\ell_0}]\!]
\end{array}
$$

Observation:

- Each variable is bound just once.
- The cache is playing the role of a *global* environment

An algorithm for linear 0CFA

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!]\rho & = & \rho(x) \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!]\rho & = & \langle \lambda x.e, \rho \rangle \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]\rho & = & \textbf{let } \langle \lambda x.t^{\ell_0}, \rho' \rangle = \mathcal{A}[\![t^{\ell_1}]\!]\rho_1 \textbf{ in} \\
 & & \textbf{let } c = \mathcal{A}[\![t^{\ell_2}]\!]\rho_2 \textbf{ in} \\
 & & \quad \mathcal{A}[\![t^{\ell_0}]\!]\rho'[x \mapsto c] \\
 & & \quad \text{where } \rho_i = \rho \restriction \mathbf{fv}(t^{\ell_i}) \\
 & & \quad \text{and } \rho_1 \cap \rho_2 = \emptyset
\end{array}
$$

- Each variable is bound just once.
- The cache is playing the role of a *global* environment.

A linear evaluator

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!]\rho & = & \rho(x) \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!]\rho & = & \langle \lambda x.e, \rho \rangle \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]\rho & = & \textbf{let } \langle \lambda x.t^{\ell_0}, \rho' \rangle = \mathcal{A}[\![t^{\ell_1}]\!]\rho_1 \textbf{ in} \\
 & & \textbf{let } c = \mathcal{A}[\![t^{\ell_2}]\!]\rho_2 \textbf{ in} \\
 & & \quad \mathcal{A}[\![t^{\ell_0}]\!]\rho'[x \mapsto c] \\
 & & \quad \text{where } \rho_i = \rho \restriction \mathbf{fv}(t^{\ell_i}) \\
 & & \quad \text{and } \rho_1 \cap \rho_2 = \emptyset
\end{array}
$$

Observation:

- This is just an *evaluator for linear programs*.

The path ahead

1. Specialize the 0CFA algorithm for linear terms. ✓
2. Derive an evaluator for linear terms. ✓
3. Un-specialize to reveal typical evaluator.

A linear evaluator

$$
\begin{array}{lcl}
\mathcal{E}[\![x^{\ell}]\!]\rho & = & \rho(x) \\
\mathcal{E}[\![(\lambda x.e)^{\ell}]\!]\rho & = & \langle \lambda x.e, \rho \rangle \\
\mathcal{E}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]\rho & = & \textbf{let } \langle \lambda x.t^{\ell_0}, \rho' \rangle = \mathcal{E}[\![t^{\ell_1}]\!]\rho_1 \textbf{ in} \\
 & & \textbf{let } c = \mathcal{E}[\![t^{\ell_2}]\!]\rho_2 \textbf{ in} \\
 & & \quad \mathcal{E}[\![t^{\ell_0}]\!]\rho'[x \mapsto c] \\
 & & \quad \text{where } \rho_i = \rho \restriction \mathbf{fv}(t^{\ell_i}) \\
 & & \quad \text{and } \rho_1 \cap \rho_2 = \emptyset
\end{array}
$$

Observation:

- This is just an *evaluator for linear programs*.

A ~~linear~~ evaluator

$$
\begin{array}{lcl}
\mathcal{E}[\![x^{\ell}]\!]\rho & = & \rho(x) \\
\mathcal{E}[\![(\lambda x.e)^{\ell}]\!]\rho & = & \langle \lambda x.e, \rho \rangle \\
\mathcal{E}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]\rho & = & \textbf{let } \langle \lambda x.t^{\ell_0}, \rho' \rangle = \mathcal{E}[\![t^{\ell_1}]\!]\rho_1 \textbf{ in} \\
 & & \textbf{let } c = \mathcal{E}[\![t^{\ell_2}]\!]\rho_2 \textbf{ in} \\
 & & \quad \mathcal{E}[\![t^{\ell_0}]\!]\rho'[x \mapsto c] \\
 & & \quad \text{where } \rho_i = \rho \restriction \mathbf{fv}(t^{\ell_i}) \\
 & & \quad \text{and } \cancel{\rho_1 \cap \rho_2 = \emptyset}
\end{array}
$$

Observation:

- This is just an *evaluator*.

A ~~linear~~ evaluator

$$
\begin{array}{lcl}
\mathcal{E}[\![x^{\ell}]\!]\rho & = & \rho(x) \\
\mathcal{E}[\![(\lambda x.e)^{\ell}]\!]\rho & = & \langle \lambda x.e, \rho' \rangle \text{ where } \rho' = \rho \restriction \mathbf{fv}(e) \\
\mathcal{E}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]\rho & = & \textbf{let } \langle \lambda x.t^{\ell_0}, \rho' \rangle = \mathcal{E}[\![t^{\ell_1}]\!]\rho \textbf{ in} \\
 & & \textbf{let } c = \mathcal{E}[\![t^{\ell_2}]\!]\rho \textbf{ in} \\
 & & \quad \mathcal{E}[\![t^{\ell_0}]\!]\rho'[x \mapsto c] \\
 & & \quad \sout{\text{where } \rho_i = \rho \restriction \mathbf{fv}(t^{\ell_i})} \\
 & & \quad \sout{\text{and } \rho_1 \cap \rho_2 = \emptyset}
\end{array}
$$

Observation:

- This is just an *evaluator*.

A typical evaluator

$$
\begin{array}{lcl}
\mathcal{E}[\![x^{\ell}]\!]\rho & = & \rho(x) \\
\mathcal{E}[\![(\lambda x.e)^{\ell}]\!]\rho & = & \langle \lambda x.e, \rho' \rangle \text{ where } \rho' = \rho \restriction \mathbf{fv}(e) \\
\mathcal{E}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]\rho & = & \textbf{let } \langle \lambda x.t^{\ell_0}, \rho' \rangle = \mathcal{E}[\![t^{\ell_1}]\!]\rho \textbf{ in} \\
 & & \textbf{let } c = \mathcal{E}[\![t^{\ell_2}]\!]\rho \textbf{ in} \\
 & & \quad \mathcal{E}[\![t^{\ell_0}]\!]\rho'[x \mapsto c]
\end{array}
$$

Outline

- Inspirations ✓
- Flow analysis and linearity ✓
- Consequences

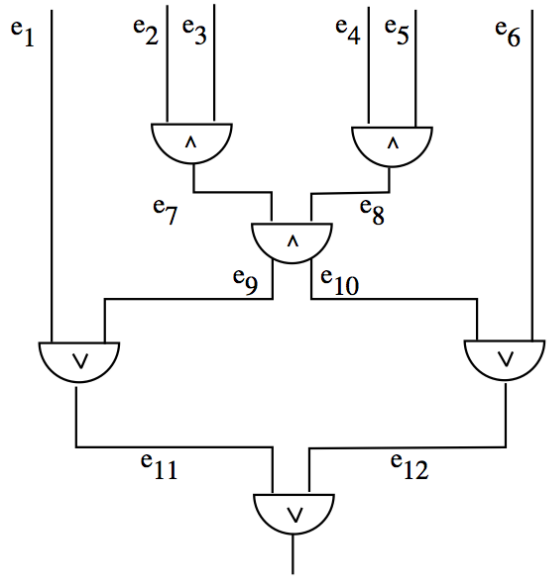
Consequences

PTIME

Because evaluation of linear terms is complete for PTIME, so is 0CFA.

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

} linear ML terms



- Difficult to parallelize effectively.
- Difficult to solve in limited space.

Further approximations

The best 0CFA algorithm is cubic (and unlikely to be improved: Heintze and McAllester, LICS 97).
Several further approximations to 0CFA have been developed:

- Henglein's simple closure analysis
- Ashley and Dybvig's Sub-0CFA
- Heintze and McAllester "subtransitive" flow analysis
- Mossin flow graphs

But. . . *on linear programs* they are all equivalent to each other and to evaluation. They are all complete for PTIME.

Henglein's simple flow analysis

Idea: replace $\subseteq$ with $=$.

But: Makes no difference for a linear program!

Ashley and Dybvig's Sub-0CFA

Idea: limit the number of passes over the program.

But: Just 1 will do for a linear program!

Beyond 0CFA

kCFA ($k > 0$) is complete for EXPTIME (ICFP 2008).

There is no tractable algorithm for kCFA. Theory and practice meet.

But the exact subset of kCFA is only complete for PTIME, i.e. today's technique does not lead to interesting lower bounds.

What has been done?

- Given a transparent proof of the correspondence between evaluation and flow analysis for linear programs.
- Identified a core language on which several variants of flow analysis (and evaluation) coincide.
- Shown all of these variants are PTIME complete.
- Sketched some ideas for abstract interpretation in general.

What has been done?

- Given a transparent proof of the correspondence between evaluation and flow analysis for linear programs.
- Identified a core language on which several variants of flow analysis (and evaluation) coincide.
- Shown all of these variants are PTIME complete.
- Sketched some ideas for abstract interpretation in general.

Moltes Gràcies