Type- and Control-Flow Analysis

Matthew Fluet
mtf@cs.rit.edu

Department of Computer Science
Rochester Institute of Technolgy

December 15, 2015
Advances in Programming Languages and Systems (APLS)

Introduction

Control-flow analysis

A compile-time approximation of the “flow” of functions in program: which functions might be bound to a given variable at run time.

- an enabling analysis for the compilation of functional languages
  - because control flow is not syntactically apparent
- typically formulated for dynamically- or simply-typed languages

Introduction

Control-flow analysis

A compile-time approximation of the "flow" of functions in program: which functions might be bound to a given variable at run time.

- an enabling analysis for the compilation of functional languages
  - because control flow is not syntactically apparent
- typically formulated for dynamically- or simply-typed languages

- popular statically-typed functional languages are richly-typed
- System F (and extensions) are popular intermediate languages

Seek a control-flow analysis formulated for System F (and extensions).
Exploit well-typedness to obtain more precise control-flow information.

Example

$$
\begin{aligned}
\mathsf{f1} &= \lambda\, \mathsf{x1}\ .\ \cdots \\
\mathsf{f2} &= \lambda\, \mathsf{x2}\ .\ \cdots
\end{aligned}
$$

$$
\mathsf{id} = \lambda\, \mathsf{x}\ .\ \ \mathsf{x}
$$

$$
\begin{aligned}
\mathsf{res1} &= \mathsf{id}\quad \mathsf{f1} \\
\mathsf{res2} &= \mathsf{id}\quad \mathsf{f2}
\end{aligned}
$$

Example

$$
\begin{aligned}
\mathsf{f1} &= \lambda\, \mathsf{x1}\ .\ \cdots \\
\mathsf{f2} &= \lambda\, \mathsf{x2}\ .\ \cdots
\end{aligned}
$$

$$
\mathsf{id} = \lambda\, \mathsf{x}\ .\ \ \mathsf{x}
$$

$$
\begin{aligned}
\mathsf{res1} &= \mathsf{id}\ \ \mathsf{f1} \\
\mathsf{res2} &= \mathsf{id}\ \ \mathsf{f2}
\end{aligned}
$$

Example

$$
\begin{aligned}
&\mathsf{f1} \;=\; \lambda\ \mathsf{x1}\ .\ \cdots \\
&\mathsf{f2} \;=\; \lambda\ \mathsf{x2}\ .\ \cdots \\
&\mathsf{id} \;=\; \lambda\ \mathsf{x}\ .\ \mathsf{x} \\
&\mathsf{res1} \;=\; \mathsf{id}\ \ \mathsf{f1} \\
&\mathsf{res2} \;=\; \mathsf{id}\ \ \mathsf{f2}
\end{aligned}
$$

Example

$$
\begin{aligned}
\mathsf{f1} &= \lambda\ \mathsf{x1}\ .\ \cdots \\
\mathsf{f2} &= \lambda\ \mathsf{x2}\ .\ \cdots
\end{aligned}
$$

$$
\mathsf{id} = \lambda\ \mathsf{x}\ .\ \mathsf{x}
$$

$$
\begin{aligned}
\mathsf{res1} &= \mathsf{id}\quad \mathsf{f1} \\
\mathsf{res2} &= \mathsf{id}\quad \mathsf{f2}
\end{aligned}
$$

Example

$$
\begin{aligned}
\mathsf{f1} &= \lambda\, \mathsf{x1}\ .\ \cdots \\
\mathsf{f2} &= \lambda\, \mathsf{x2}\ .\ \cdots \\[1em]
\mathsf{id} &= \lambda\, \mathsf{x}\ .\ \mathsf{x} \\[1em]
\mathsf{res1} &= \mathsf{id}\ \ \mathsf{f1} \\
\mathsf{res2} &= \mathsf{id}\ \ \mathsf{f2}
\end{aligned}
$$

Example

$$
\begin{aligned}
\text{f1} &= \lambda\ \text{x1}\ .\ \cdots \\
\text{f2} &= \lambda\ \text{x2}\ .\ \cdots
\end{aligned}
$$

$$
\text{id} = \lambda\ \text{x}\ .\ \text{x}
$$

$$
\begin{aligned}
\text{res1} &= \text{id} \quad \text{f1} \\
\text{res2} &= \text{id} \quad \text{f2}
\end{aligned}
$$

Example

$$\begin{aligned}
\mathsf{f1} &= \lambda\, \mathsf{x1}\ .\ \cdots \\
\mathsf{f2} &= \lambda\, \mathsf{x2}\ .\ \cdots \\[1em]
\mathsf{id} &= \lambda\, \mathsf{x}\ .\ \mathsf{x} \\[1em]
\mathsf{res1} &= \mathsf{id}\ \ \mathsf{f1} \\
\mathsf{res2} &= \mathsf{id}\ \ \mathsf{f2}
\end{aligned}$$

	0CFA
$\hat{\rho}(\mathsf{x})$	{ f1 , f2 }
$\hat{\rho}(\mathsf{res1})$	{ f1 , f2 }
$\hat{\rho}(\mathsf{res2})$	{ f1 , f2 }

0CFA — "classic" monovariant/context-insensitive control-flow analysis

Example

$f1 : Int \rightarrow Int = \lambda\, x1 : Int.\ \cdots$

$f2 : Bool \rightarrow Bool = \lambda\, x2 : Bool.\ \cdots$

$id : \forall\alpha.\ \alpha \rightarrow \alpha = \Lambda\, \alpha\ .\ \lambda\, x :\ \alpha\ .\ \ x$

$res1 : Int \rightarrow Int = \ id\ \ [\, Int \rightarrow Int\,]\ \ f1$

$res2 : Bool \rightarrow Bool = \ id\ \ [\, Bool \rightarrow Bool\,]\ \ f2$

Example

$$\mathsf{f1} : \mathsf{Int} \to \mathsf{Int} = \lambda\, \mathsf{x1} : \mathsf{Int}.\ \cdots$$
$$\mathsf{f2} : \mathsf{Bool} \to \mathsf{Bool} = \lambda\, \mathsf{x2} : \mathsf{Bool}.\ \cdots$$

$$\mathsf{id} : \forall\alpha.\ \alpha \to \alpha = \Lambda\, \alpha\ .\ \lambda\, \mathsf{x} : \alpha\ .\ \mathsf{x}$$

$$\mathsf{res1} : \mathsf{Int} \to \mathsf{Int} = \mathsf{id}\ [\ \mathsf{Int} \to \mathsf{Int}\ ]\ \mathsf{f1}$$
$$\mathsf{res2} : \mathsf{Bool} \to \mathsf{Bool} = \mathsf{id}\ [\ \mathsf{Bool} \to \mathsf{Bool}\ ]\ \mathsf{f2}$$

Example

$$\mathsf{f1} : \mathsf{Int} \to \mathsf{Int} = \lambda\, \mathsf{x1} : \mathsf{Int}.\ \cdots$$

$$\mathsf{f2} : \mathsf{Bool} \to \mathsf{Bool} = \lambda\, \mathsf{x2} : \mathsf{Bool}.\ \cdots$$

$$\mathsf{id} : \forall\alpha.\ \alpha \to \alpha = \Lambda\, \alpha\ .\ \lambda\, \mathsf{x} : \alpha\ .\ \mathsf{x}$$

$$\mathsf{res1} : \mathsf{Int} \to \mathsf{Int} = \mathsf{id}\ [\,\mathsf{Int} \to \mathsf{Int}\,]\ \mathsf{f1}$$

$$\mathsf{res2} : \mathsf{Bool} \to \mathsf{Bool} = \mathsf{id}\ [\,\mathsf{Bool} \to \mathsf{Bool}\,]\ \mathsf{f2}$$

	CFA
$\hat{\rho}(\mathsf{x}^{\alpha})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}},\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$
$\hat{\rho}(\mathsf{res1}^{\mathsf{Int}\to\mathsf{Int}})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}},\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$
$\hat{\rho}(\mathsf{res2}^{\mathsf{Bool}\to\mathsf{Bool}})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}},\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$

Example

$\mathsf{f1} : \mathsf{Int} \to \mathsf{Int} = \lambda\, \mathsf{x1} : \mathsf{Int}.\ \cdots$

$\mathsf{f2} : \mathsf{Bool} \to \mathsf{Bool} = \lambda\, \mathsf{x2} : \mathsf{Bool}.\ \cdots$

$\mathsf{id} : \forall\alpha.\ \alpha \to \alpha = \Lambda\, \alpha\ .\ \lambda\, \mathsf{x} : \alpha\ .\ \mathsf{x}$

$\mathsf{res1} : \mathsf{Int} \to \mathsf{Int} = \mathsf{id}\ [\ \mathsf{Int} \to \mathsf{Int}\ ]\ \mathsf{f1}$

$\mathsf{res2} : \mathsf{Bool} \to \mathsf{Bool} = \mathsf{id}\ [\ \mathsf{Bool} \to \mathsf{Bool}\ ]\ \mathsf{f2}$

	CFA then type-filter
$\hat{\rho}(\mathsf{x}^{\alpha})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}},\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$
$\hat{\rho}(\mathsf{res1}^{\mathsf{Int}\to\mathsf{Int}})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}}\ \}$
$\hat{\rho}(\mathsf{res2}^{\mathsf{Bool}\to\mathsf{Bool}})$	$\{\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$

Example

$$\text{f1} : \text{Int} \to \text{Int} = \lambda\, \text{x1} : \text{Int}.\ \cdots$$
$$\text{f2} : \text{Bool} \to \text{Bool} = \lambda\, \text{x2} : \text{Bool}.\ \cdots$$

$$\text{id} : \forall\alpha.\ \alpha \to \alpha = \Lambda\, \alpha\ .\ \lambda\, \text{x} :\ \alpha\ .\ \text{x}$$

$$\text{res1} : \text{Int} \to \text{Int} = \text{id}\ [\ \text{Int} \to \text{Int}\ ]\ \text{f1}$$
$$\text{res2} : \text{Bool} \to \text{Bool} = \text{id}\ [\ \text{Bool} \to \text{Bool}\ ]\ \text{f2}$$

	CFA then type-filter
$\hat{\rho}(\text{x}^{\alpha})$	$\{\ \text{f1}^{\text{Int}\to\text{Int}},\ \text{f2}^{\text{Bool}\to\text{Bool}}\ \}$
$\hat{\rho}(\text{res1}^{\text{Int}\to\text{Int}})$	$\{\ \text{f1}^{\text{Int}\to\text{Int}}\ \}$
$\hat{\rho}(\text{res2}^{\text{Bool}\to\text{Bool}})$	$\{\ \text{f2}^{\text{Bool}\to\text{Bool}}\ \}$

Example

f1 : Int $\to$ Int = λ x1 : Int. $\cdots$

f2 : Bool $\to$ Bool = λ x2 : Bool. $\cdots$

id : $\forall\alpha.\ \alpha \to \alpha$ = $\Lambda\ \alpha$. λ x : α . x

res1 : Int $\to$ Int = id [Int $\to$ Int] f1

res2 : Bool $\to$ Bool = id [Bool $\to$ Bool] f2

	TCFA
$\hat{\rho}(\mathsf{x}^{\alpha})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}}\ ,\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$
$\hat{\rho}(\mathsf{res1}^{\mathsf{Int}\to\mathsf{Int}})$	$\{\ \mathsf{f1}^{\mathsf{Int}\to\mathsf{Int}}\ \}$
$\hat{\rho}(\mathsf{res2}^{\mathsf{Bool}\to\mathsf{Bool}})$	$\{\ \mathsf{f2}^{\mathsf{Bool}\to\mathsf{Bool}}\ \}$
$\hat{\phi}(\alpha)$	$\{\ \mathsf{Int}\to\mathsf{Int}\ ,\ \mathsf{Bool}\to\mathsf{Bool}\ \}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	CFA
$\hat{\rho}$(x)	$\{ \lambda a1\ ,\ \lambda b1\ ,\ \lambda c1 \}$
$\hat{\rho}$(f)	$\{ \lambda c1 \}$
$\hat{\rho}$(g)	$\{ \lambda a1\ ,\ \lambda b1\ ,\ \lambda c1 \}$
$\hat{\rho}$(res1)	$\{ \lambda a1\ ,\ \lambda b1\ ,\ \lambda c1 \}$
$\hat{\rho}$(res2)	$\{ \lambda a1\ ,\ \lambda b1\ ,\ \lambda c1 \}$
$\hat{\rho}$(res3)	$\{ \lambda a2\ ,\ \lambda b2\ ,\ \lambda c2 \}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	CFA then type-filter
$\hat{\rho}$(x)	{ $\lambda a1$, $\lambda b1$, $\lambda c1$ }
$\hat{\rho}$(f)	{ $\lambda c1$ }
$\hat{\rho}$(g)	{ $\lambda a1$, $\lambda b1$, $\lambda c1$ }
$\hat{\rho}$(res1)	{ $\lambda a1$, $\lambda b1$, $\lambda c1$ }
$\hat{\rho}$(res2)	{ $\lambda a1$, $\lambda b1$, $\lambda c1$ }
$\hat{\rho}$(res3)	{ $\lambda a2$, $\lambda b2$, $\lambda c2$ }

Example

$\texttt{id}:\forall\alpha.\ \alpha \to \alpha$ = $\Lambda\alpha.\ \lambda\texttt{x}:\alpha.\ \texttt{x}$
$\texttt{app}:\forall\beta.\ \forall\gamma.\ (\beta \to \gamma) \to \beta \to \gamma$ =
$\quad\Lambda\beta.\Lambda\gamma.\lambda\texttt{f}:\beta \to \gamma.\lambda\texttt{z}:\beta.$ let $\texttt{g}:\beta \to \gamma$ = id $[\beta \to \gamma]$ f in g z

$\texttt{h1}:\text{int} \to \text{int} \to \text{int}$ = $\lambda\texttt{a1}:\text{int}.\lambda\texttt{a2}:\text{int}.$ a1+a2
$\texttt{h2}:\text{bool} \to \text{int} \to \text{int}$ = $\lambda\texttt{b1}:\text{bool}.\lambda\texttt{b2}:\text{int}.$ if b1 then b2+1 else b2
$\texttt{h3}:\text{str} \to \text{int} \to \text{int}$ = $\lambda\texttt{c1}:\text{str}.\lambda\texttt{c2}:\text{int}.$ len(c1)+c2

$\texttt{res1}:\text{int} \to \text{int} \to \text{int}$ = id $[\text{int} \to \text{int} \to \text{int}]$ h1
$\texttt{res2}:\text{bool} \to \text{int} \to \text{int}$ = id $[\text{bool} \to \text{int} \to \text{int}]$ h2
$\texttt{res3}:\text{int} \to \text{int}$ = app [str] $[\text{int} \to \text{int}]$ h3 "zzz"

	CFA then type-filter
$\hat{\rho}(\text{x})$	$\{\lambda a1,\ \lambda b1,\ \lambda c1\}$
$\hat{\rho}(\text{f})$	$\{\lambda c1\}$
$\hat{\rho}(\text{g})$	$\{\lambda a1,\ \lambda b1,\ \lambda c1\}$
$\hat{\rho}(\text{res1})$	$\{\lambda a1\}$
$\hat{\rho}(\text{res2})$	$\{\lambda b1\}$
$\hat{\rho}(\text{res3})$	$\{\lambda a2,\ \lambda b2,\ \lambda c2\}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	(T+C)FA
$\hat{\rho}(\mathrm{x})$	$\{\, \lambda a1 \,,\ \lambda b1 \,,\ \lambda c1 \,\}$
$\hat{\rho}(\mathrm{f})$	$\{\, \lambda c1 \,\}$
$\hat{\rho}(\mathrm{g})$	$\{\, \lambda a1 \,,\ \lambda b1 \,,\ \lambda c1 \,\}$
$\hat{\rho}(\mathrm{res1})$	$\{\, \lambda a1 \,,\ \lambda b1 \,,\ \lambda c1 \,\}$
$\hat{\rho}(\mathrm{res2})$	$\{\, \lambda a1 \,,\ \lambda b1 \,,\ \lambda c1 \,\}$
$\hat{\rho}(\mathrm{res3})$	$\{\, \lambda a2 \,,\ \lambda b2 \,,\ \lambda c2 \,\}$
$\hat{\phi}(\alpha)$	$\{\, \mathrm{int}{\to}\mathrm{int}{\to}\mathrm{int} \,,\ \mathrm{bool}{\to}\mathrm{int}{\to}\mathrm{int} \,,\ \beta{\to}\gamma \,\}$
$\hat{\phi}(\beta)$	$\{\, \mathrm{str} \,\}$
$\hat{\phi}(\gamma)$	$\{\, \mathrm{int}{\to}\mathrm{int} \,\}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	(T+C)FA then type-filter
$\hat{\rho}(\mathsf{x})$	$\{\, \lambda a1 \,,\ \lambda b1 \,,\ \lambda c1 \,\}$
$\hat{\rho}(\mathsf{f})$	$\{\, \lambda c1 \,\}$
$\hat{\rho}(\mathsf{g})$	$\{\, \color{red}{\lambda a1} \,,\ \color{red}{\lambda b1} \,,\ \lambda c1 \,\}$
$\hat{\rho}(\mathsf{res1})$	$\{\, \lambda a1 \,,\ \color{red}{\lambda b1} \,,\ \color{red}{\lambda c1} \,\}$
$\hat{\rho}(\mathsf{res2})$	$\{\, \color{red}{\lambda a1} \,,\ \lambda b1 \,,\ \color{red}{\lambda c1} \,\}$
$\hat{\rho}(\mathsf{res3})$	$\{\, \lambda a2 \,,\ \lambda b2 \,,\ \lambda c2 \,\}$
$\hat{\phi}(\alpha)$	$\{\, \mathsf{int}{\to}\mathsf{int}{\to}\mathsf{int} \,,\ \mathsf{bool}{\to}\mathsf{int}{\to}\mathsf{int} \,,\ \beta{\to}\gamma \,\}$
$\hat{\phi}(\beta)$	$\{\, \color{blue}{\mathsf{str}} \,\}$
$\hat{\phi}(\gamma)$	$\{\, \color{blue}{\mathsf{int}{\to}\mathsf{int}} \,\}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	(T+C)FA then type-filter
$\hat{\rho}(\mathrm{x})$	$\{\, \lambda a1\ ,\ \lambda b1\ ,\ \lambda c1 \,\}$
$\hat{\rho}(\mathrm{f})$	$\{\, \lambda c1 \,\}$
$\hat{\rho}(\mathrm{g})$	$\{\, \lambda c1 \,\}$
$\hat{\rho}(\mathrm{res1})$	$\{\, \lambda a1 \,\}$
$\hat{\rho}(\mathrm{res2})$	$\{\, \lambda b1 \,\}$
$\hat{\rho}(\mathrm{res3})$	$\{\, \lambda a2\ ,\ \lambda b2\ ,\ \lambda c2 \,\}$
$\hat{\phi}(\alpha)$	$\{\, \mathrm{int}\to\mathrm{int}\to\mathrm{int}\ ,\ \mathrm{bool}\to\mathrm{int}\to\mathrm{int}\ ,\ \beta\to\gamma \,\}$
$\hat{\phi}(\beta)$	$\{\, \mathrm{str} \,\}$
$\hat{\phi}(\gamma)$	$\{\, \mathrm{int}\to\mathrm{int} \,\}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	(T&C)FA
$\hat{\rho}(\mathsf{x})$	$\{\, \lambda a1 \,,\, \lambda b1 \,,\, \lambda c1 \,\}$
$\hat{\rho}(\mathsf{f})$	$\{\, \lambda c1 \,\}$
$\hat{\rho}(\mathsf{g})$	$\{\, \lambda c1 \,\}$
$\hat{\rho}(\mathsf{res1})$	$\{\, \lambda a1 \,\}$
$\hat{\rho}(\mathsf{res2})$	$\{\, \lambda b1 \,\}$
$\hat{\rho}(\mathsf{res3})$	$\{\, \lambda a2 \,,\, \lambda b2 \,,\, \lambda c2 \,\}$
$\hat{\phi}(\alpha)$	$\{\, \mathsf{int}{\to}\mathsf{int}{\to}\mathsf{int} \,,\, \mathsf{bool}{\to}\mathsf{int}{\to}\mathsf{int} \,,\, \beta{\to}\gamma \,\}$
$\hat{\phi}(\beta)$	$\{\, \mathsf{str} \,\}$
$\hat{\phi}(\gamma)$	$\{\, \mathsf{int}{\to}\mathsf{int} \,\}$

Example

```
id:∀α. α → α = Λα. λx:α. x
app:∀β. ∀γ. (β → γ) → β → γ =
    Λβ.Λγ.λf:β → γ.λz:β. let g:β → γ = id [β → γ] f in g z

h1:int → int → int = λa1:int.λa2:int. a1+a2
h2:bool → int → int = λb1:bool.λb2:int. if b1 then b2+1 else b2
h3:str → int → int = λc1:str.λc2:int. len(c1)+c2

res1:int → int → int = id [int → int → int] h1
res2:bool → int → int = id [bool → int → int] h2
res3:int → int = app [str] [int → int] h3 "zzz"
```

	(T&C)FA
$\hat{\rho}(\mathsf{x})$	$\{\lambda a1, \lambda b1, \lambda c1\}$
$\hat{\rho}(\mathsf{f})$	$\{\lambda c1\}$
$\hat{\rho}(\mathsf{g})$	$\{\lambda c1\}$
$\hat{\rho}(\mathsf{res1})$	$\{\lambda a1\}$
$\hat{\rho}(\mathsf{res2})$	$\{\lambda b1\}$
$\hat{\rho}(\mathsf{res3})$	$\{\lambda c2\}$
$\hat{\phi}(\alpha)$	$\{\mathsf{int}\to\mathsf{int}\to\mathsf{int}, \mathsf{bool}\to\mathsf{int}\to\mathsf{int}, \beta\to\gamma\}$
$\hat{\phi}(\beta)$	$\{\mathsf{str}\}$
$\hat{\phi}(\gamma)$	$\{\mathsf{int}\to\mathsf{int}\}$

Introduction

A type- and control-flow analysis for System F

Type-flow analysis

A compile-time approximation of the “flow” of types in program: which types might be bound to a given type variable at run time

$\Uparrow$ determines type abstractions flowing to type applications

$\Downarrow$ rejects flows incompatible with static typing

Control-flow analysis

A compile-time approximation of the “flow” of functions in program: which functions might be bound to a given variable at run time

Introduction

A type- and control-flow analysis for System F

Types improve the precision of the control-flow analysis;
improve the effectiveness of optimizations based on the analysis.

- inlining, copy propagation, dead-code elimination, ...

Type-flow analysis enables novel optimizations.

- polymorphic functions used at a finite number of types can be optimized to monomorphic instances
- optimization of intensional polymorphism (i.e., $\texttt{typecase}\ \alpha\ \texttt{of}\ \cdots$)

Overview

- Specification-Based Formulation of TCFA (IFL'12)

  A collection of declarative constraints that a valid analysis result must satisfy;
  given a proposed analysis result, verify that it satisfies the constraints.

  - Soundness, Existence, Decidability, Computability, Complexity

- Flow-Graph-Based Formulation of TCFA (IFL'14)

  A graph with edges corresponding to flow of abstract values;
  analysis result determined by graph reachability.

  - Soundness, Algorithm, Complexity

- Related and Future Work

Specification-based Formulation: Acceptability

$\boxed{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}$ "$\hat{\phi}$ and $\hat{\rho}$ are *acceptable* for e"

$\hat{\phi}$ and $\hat{\rho}$ approximate every run-time type and value environment that arises during evaluation

Abstract type environments map type variables to sets of (syntactic) types.
Abstract value environments map variables to sets of (syntactic) values.

$$\begin{array}{rcccl} \mathit{ATEnv} = \mathit{TyVar} \to \mathcal{P}(\mathit{Type}) & \ni & \hat{\phi} & ::= & \{\alpha \mapsto \{\tau, \ldots\}, \ldots\} \\ \mathit{AVEnv} = \mathit{Var} \to \mathcal{P}(\mathit{Val}) & \ni & \hat{\rho} & ::= & \{x \mapsto \{v, \ldots\}, \ldots\} \end{array}$$

Specification-based Formulation: Acceptability

$\boxed{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}$ "$\hat{\phi}$ and $\hat{\rho}$ are an *acceptable* type- and control-flow analysis for e"

Specification-based Formulation: Acceptability

$\boxed{\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e}$ "$\hat{\phi}$ and $\hat{\rho}$ are an *acceptable* type- and control-flow analysis for e"

$$\frac{}{\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} x}$$

$$\frac{\lambda z{:}\tau_z\,.\,e_b \in \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e_b \qquad \hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \lambda z{:}\tau_z\,.\,e_b\ \texttt{in}\ e} \qquad \frac{\Lambda\beta\,.\,e_b \in \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e_b \qquad \hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \Lambda\beta\,.\,e_b\ \texttt{in}\ e}$$

$$\frac{\displaystyle\bigwedge_{\lambda z:\tau_z\,.\,e_b \in \hat{\rho}(x_f)} \begin{pmatrix} \forall v_a \in \hat{\rho}(x_a)\ .\ v_a \in \hat{\rho}(z) \land \\ \forall v_b \in \hat{\rho}(\mathsf{ResOf}(e_b))\ .\ v_b \in \hat{\rho}(x) \end{pmatrix} \qquad \hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e}$$

$$\frac{\displaystyle\bigwedge_{\Lambda\beta\,.\,e_b \in \hat{\rho}(x_f)} \left(\forall v_b \in \hat{\rho}(\mathsf{ResOf}(e_b))\ .\ v_b \in \hat{\rho}(x) \right) \qquad \hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ [\tau_a]\ \texttt{in}\ e}$$

Specification-based Formulation: Acceptability

$\boxed{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}$ "$\hat{\phi}$ and $\hat{\rho}$ are an *acceptable* type- and control-flow analysis for e"

$$\frac{}{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} x}$$

$$\frac{\lambda z{:}\tau_z\,.\,e_b \in \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e_b \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ \lambda z{:}\tau_z\,.\,e_b\ \mathtt{in}\ e}
\qquad
\frac{\Lambda\beta\,.\,e_b \in \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e_b \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ \Lambda\beta\,.\,e_b\ \mathtt{in}\ e}$$

$$\frac{\displaystyle\bigwedge_{\lambda z{:}\tau_z\,.\,e_b \in \hat{\rho}(x_f)} \begin{pmatrix} \forall v_a \in \hat{\rho}(x_a)\ .\ v_a \in \hat{\rho}(z) \wedge \\ \forall v_b \in \hat{\rho}(\mathrm{ResOf}(e_b))\ .\ v_b \in \hat{\rho}(x) \end{pmatrix} \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e}$$

$$\frac{\displaystyle\bigwedge_{\Lambda\beta\,.\,e_b \in \hat{\rho}(x_f)} \begin{pmatrix} \tau_a \in \hat{\phi}(\beta) \wedge \\ \forall v_b \in \hat{\rho}(\mathrm{ResOf}(e_b))\ .\ v_b \in \hat{\rho}(x) \end{pmatrix} \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} \mathtt{let}\ x{:}\tau_x\ \mathtt{=}\ x_f\ [\tau_a]\ \mathtt{in}\ e}$$

Specification-based Formulation: Acceptability

$\boxed{\hat{\phi}; \hat{\rho} \vDash_S e}$ "$\hat{\phi}$ and $\hat{\rho}$ are an *acceptable* type- and control-flow analysis for e"

$$\frac{}{\hat{\phi}; \hat{\rho} \vDash_S x}$$

$$\frac{\lambda z{:}\tau_z\,.\,e_b \in \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash_S e_b \qquad \hat{\phi}; \hat{\rho} \vDash_S e}{\hat{\phi}; \hat{\rho} \vDash_S \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \lambda z{:}\tau_z\,.\,e_b\ \texttt{in}\ e} \qquad \frac{\Lambda\beta\,.\,e_b \in \hat{\rho}(x) \qquad \hat{\phi}; \hat{\rho} \vDash_S e_b \qquad \hat{\phi}; \hat{\rho} \vDash_S e}{\hat{\phi}; \hat{\rho} \vDash_S \texttt{let}\ x{:}\tau_x\ \texttt{=}\ \Lambda\beta\,.\,e_b\ \texttt{in}\ e}$$

$$\frac{\displaystyle\bigwedge_{\lambda z:\tau_z\,.\,e_b \in \hat{\rho}(x_f)} \begin{pmatrix} \forall v_a \in \hat{\rho}(x_a)\,.\,\hat{\phi} \vDash_S \mathsf{TyOf}(v_a) \approxeq \tau_z \Rightarrow v_a \in \hat{\rho}(z) \wedge \\ \forall v_b \in \hat{\rho}(\mathsf{ResOf}(e_b))\,.\,\hat{\phi} \vDash_S \mathsf{TyOf}(v_b) \approxeq \tau_x \Rightarrow v_b \in \hat{\rho}(x) \end{pmatrix} \qquad \hat{\phi}; \hat{\rho} \vDash_S e}{\hat{\phi}; \hat{\rho} \vDash_S \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ x_a\ \texttt{in}\ e}$$

$$\frac{\displaystyle\bigwedge_{\Lambda\beta\,.\,e_b \in \hat{\rho}(x_f)} \begin{pmatrix} \tau_a \in \hat{\phi}(\beta) \wedge \\ \forall v_b \in \hat{\rho}(\mathsf{ResOf}(e_b))\,.\,\hat{\phi} \vDash_S \mathsf{TyOf}(v_b) \approxeq \tau_x \Rightarrow v_b \in \hat{\rho}(x) \end{pmatrix} \qquad \hat{\phi}; \hat{\rho} \vDash_S e}{\hat{\phi}; \hat{\rho} \vDash_S \texttt{let}\ x{:}\tau_x\ \texttt{=}\ x_f\ [\tau_a]\ \texttt{in}\ e}$$

Specification-based Formulation: Type Compatibility

$\boxed{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2}$ “τ_1 and τ_2 are *compatible* under $\hat{\phi}$”

Specification-based Formulation: Type Compatibility

$\boxed{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approxeq \tau_2}$ "τ_1 and τ_2 are *compatible* under $\hat{\phi}$"

$$\frac{\hat{\phi} \models_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \models_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 = \theta_2}{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approxeq \tau_2}$$

Specification-based Formulation: Type Compatibility

$\boxed{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2}$ "τ_1 and τ_2 are *compatible* under $\hat{\phi}$"

$$\frac{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \vDash_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 = \theta_2}{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2}$$

τ_1 and τ_2 "expand" to a common closed type;
$\theta_1 = \theta_2$ is syntactic equality

Specification-based Formulation: Type Compatibility

$\boxed{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2}$ "τ_1 and τ_2 are *compatible* under $\hat{\phi}$"

$$\frac{\hat{\phi} \models_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \models_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 = \theta_2}{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2}$$

Is $\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2$ decidable?

"Recursion" in abstract type environment foils exhaustive enumeration; may be infinitely many θ such that $\hat{\phi} \models_{\mathsf{S}} \tau \Rrightarrow \theta$.

Specification-based Formulation: Type Compatibility

$\boxed{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2}$ "τ_1 and τ_2 are *compatible* under $\hat{\phi}$"

$$\frac{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \vDash_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 = \theta_2}{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2}$$

$\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2$ is decidable!

Interpret $\hat{\phi}$ as (productions for) a *regular-tree grammar*;
a derivation of $\hat{\phi} \vDash_{\mathsf{S}} \alpha \Rrightarrow \theta$ is a parse tree.

Regular-tree grammars are closed under intersection
and emptiness of regular-tree grammars is decidable.

Specification-based Formulation: Type Compatibility

$\boxed{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2}$ "τ_1 and τ_2 are *compatible* under $\hat{\phi}$"

$$\frac{\hat{\phi} \models_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \models_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 = \theta_2}{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2}$$

$\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2$ is decidable!

Interpret $\hat{\phi}$ as (productions for) a *regular-tree grammar*;
a derivation of $\hat{\phi} \models_{\mathsf{S}} \alpha \Rrightarrow \theta$ is a parse tree.

Regular-tree grammars are closed under intersection
and emptiness of regular-tree grammars is decidable.

Least $\hat{\rho}$ and $\hat{\phi}$ such that $\hat{\phi}; \hat{\rho} \models_{\mathsf{S}} e$ computable by lfp iteration.

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	
"classic" CFA	$O(n^5)$	
TCFA		

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	
"classic" CFA	$O(n^5)$	
TCFA	$O(n^{13})$	

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	
"classic" CFA	$O(n^5)$	
TCFA	$O(n^{13})$ $O(n^{10})$ amortized	

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	Flow-Graph-Based (work-queue iteration)
"classic" CFA	$O(n^5)$	$O(n^3)$
TCFA	$O(n^{13})$ $O(n^{10})$ amortized	

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	Flow-Graph-Based (work-queue iteration)
"classic" CFA	$O(n^5)$	$O(n^3)$
TCFA	$O(n^{13})$ $O(n^{10})$ amortized	$O(n^4)$

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	Flow-Graph-Based (work-queue iteration)
"classic" CFA	$O(n^5)$	$O(n^3)$
TCFA	$O(n^{13})$ $O(n^{10})$ amortized	$O(n^4)$ $O(l^3 + l^2 m^2 + m^4)$

- l: the size of variables, values, and calls
- m: the size of type variables and types

Considering Complexity

Is the type- and control-flow analysis *efficiently* computable?

	Specification-Based (naïve lfp iteration)	Flow-Graph-Based (work-queue iteration)
"classic" CFA	$O(n^5)$	$O(n^3)$
TCFA	$O(n^{13})$ $O(n^{10})$ amortized	$O(n^4)$ $O(l^3 + l^2 m^2 + m^4)$ $O(l^3 + m^4)$ amortized

- l: the size of variables, values, and calls
- m: the size of type variables and types

Considering Complexity

$$O(l^3 + m^4)$$

Considering Complexity

$$O(l^3 + m^4)$$

- Expect size of programs of size n to be dominated by size l of variables, values, and calls, not by size m of type variables and types
  - A program may have many defs and uses of $\texttt{int} \rightarrow \texttt{int}$ funs, all sharing the same (top-level) type def

Considering Complexity

$$O(l^3 + m^4)$$

- Expect size of programs of size n to be dominated by size l of variables, values, and calls, not by size m of type variables and types
  - A program may have many defs and uses of `int` $\rightarrow$ `int` funs, all sharing the same (top-level) type def
- Not bad in theory; much better in practice

Considering Complexity

$$O(l^3 + m^4)$$

- Expect size of programs of size n to be dominated by size l of variables, values, and calls, not by size m of type variables and types
  - A program may have many defs and uses of `int` $\rightarrow$ `int` funs, all sharing the same (top-level) type def
- Not bad in theory; much better in practice
- Worst-case analysis assumes *every* value flows to *every* variable and *every* type flows to *every* type variable
  - But, CFA (and TCFA?) is useful and used in compilers
    - CFA finds many variables with small numbers (often 1) of values
    - TCFA finds many type variables with small numbers (often 1) of types
    - Algorithm only explores concl of found flows

Related Work (and Lack Thereof)

- Need to distinguish between
  - flow analyses expressed as sophisticated type systems (many)
  - flow analyses of languages with sophisticated type systems (few)

Related Work (and Lack Thereof)

- Need to distinguish between
  - flow analyses expressed as sophisticated type systems (many)
  - flow analyses of languages with sophisticated type systems (few)

- For simply-typed λ-calculus, 0CFA not improved by type filtering
- For rank-1 polymorphism (i.e., "`let`"-polymorphism), either use
  - monomorphisation (explicitly eliminate polymorphism before analysis)
    Flow-directed Closure Conversion [Cejtin, Jagannathan, & Weeks (ESOP'00)]
  - polyvariance (implicitly eliminate polymorphism during analysis)
    Set-based Program Analysis of ML Programs [Heintze (LFP'94)]
- For full System F, assume type-based analyses are "good enough"

Related Work (and Lack Thereof)

- Need to distinguish between
  - flow analyses expressed as sophisticated type systems (many)
  - flow analyses of languages with sophisticated type systems (few)

- Type-Directed Flow Analysis for Typed Intermediate Languages; Jagannathan, Weeks, & Wright (SAS'97)
  - limited to predicative System F with recursion
  - polyvariant analysis
  - diverges on programs using polymorphic recursion

- Type-sensitive Control-Flow Analysis; Reppy (ML'06)
  - Suggests mapping polymorphism to $\top$

Future Work

- Improve precision of type- and control-flow analysis
- Extend type- and control-flow to System F_ω
- Applications of type- and control-flow analysis

Future Work

Improve precision of type- and control-flow analysis:

- Adapt well-known improvements to control-flow analyses:
  - polyvariant/context-sensitive analysis
  - reachability/demand-driven analysis
  - abstract reachability & abstract cardinality
  - ...

Future Work

Improve precision of type- and control-flow analysis:

- Explore improvements motivated by well-typedness:

$$\frac{\bigwedge_{\lambda z:\tau_z \,.\, e_b \in \hat{\rho}(x_f)} \left(\begin{array}{l} \forall v_a \in \hat{\rho}(x_a) \;.\; \hat{\phi} \vDash_{\mathsf{S}} \mathrm{TyOf}(v) \approx \tau_z \Rightarrow v_a \in \hat{\rho}(z) \wedge \\ \left(\begin{array}{l} \exists v_a \in \hat{\rho}(x_a) \;.\; \hat{\phi} \vDash_{\mathsf{S}} \mathrm{TyOf}(v) \approx \tau_z \Rightarrow \\ \forall v_b \in \hat{\rho}(\mathrm{ResOf}(e_b)) \;.\; \hat{\phi} \vDash_{\mathsf{S}} \mathrm{TyOf}(v) \approx \tau_x \Rightarrow v_b \in \hat{\rho}(x) \end{array} \right) \end{array} \right) \qquad \hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e}{\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} \mathtt{let}\ x\mathtt{:}\alpha_x\ \mathtt{=}\ x_f\ x_a\ \mathtt{in}\ e}$$

  - if no actual arguments flow to formal argument,
    then function could not be called (and no flow of function result)

Future Work

Improve precision of type- and control-flow analysis:

- Explore improvements motivated by well-typedness:

$$\begin{array}{l} \texttt{rec}\ \mathsf{loop} = \Lambda\alpha.\ \lambda\mathsf{x}{:}\alpha.\ \mathsf{loop}\ [\alpha \times \alpha]\ (\mathsf{x},\mathsf{x}) \\ \mathsf{loop}\ [\mathsf{Int}]\ 1 \end{array}$$

$$\hat{\phi}(\alpha) = \{\, \mathsf{Int}\,,\ \alpha \times \alpha \,\} \qquad \hat{\phi} \vDash_{\mathsf{S}} \alpha \Rrightarrow (\mathsf{Int} \times \mathsf{Int}) \times \mathsf{Int}$$

  - but all dynamic instantiations of α are "perfect binary trees"
  - interpret (portions of) abstract type environment as an L-system?

Future Work

Extend type- and control-flow analysis to System F_ω:

$$\mathit{Kind} \ni \kappa ::= \star \mid \kappa \Rightarrow \kappa \qquad \mathit{Type} \ni \tau ::= \tau \to \tau \mid \alpha \mid \forall\alpha{:}\kappa.\ \tau \mid \lambda\alpha{:}\kappa.\ \tau \mid \tau\ \tau$$

- flow soundness with modified type-compatibility judgement:

$$\frac{\hat{\phi} \models_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \models_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 \equiv_{\beta\eta} \theta_2}{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2}$$

Future Work

Extend type- and control-flow analysis to System F_ω:

$$\mathit{Kind} \ni \kappa ::= \star \mid \kappa \Rightarrow \kappa \qquad \mathit{Type} \ni \tau ::= \tau \to \tau \mid \alpha \mid \forall \alpha{:}\kappa.\ \tau \mid \lambda \alpha{:}\kappa.\ \tau \mid \tau\ \tau$$

- flow soundness with modified type-compatibility judgement:

$$\frac{\hat{\phi} \models_{\mathsf{S}} \tau_1 \Rrightarrow \theta_1 \qquad \hat{\phi} \models_{\mathsf{S}} \tau_2 \Rrightarrow \theta_2 \qquad \theta_1 \equiv_{\beta\eta} \theta_2}{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approxeq \tau_2}$$

- is type compatibility decidable?

Future Work

Extend type- and control-flow analysis to System F_ω:

$$\mathit{Kind} \ni \kappa ::= \star \mid \kappa \Rightarrow \kappa \qquad \mathit{Type} \ni \tau ::= \tau \to \tau \mid \alpha \mid \forall\alpha{:}\kappa.\ \tau \mid \lambda\alpha{:}\kappa.\ \tau \mid \tau\ \tau$$

- flow soundness with modified type-compatibility judgement:

$$\frac{\hat{\phi} \models_{\mathsf{S}} \tau_1 \twoheadrightarrow \theta_1 \qquad \hat{\phi} \models_{\mathsf{S}} \tau_2 \twoheadrightarrow \theta_2 \qquad \theta_1 \equiv_{\beta\eta} \theta_2}{\hat{\phi} \models_{\mathsf{S}} \tau_1 \approx \tau_2}$$

- is type compatibility decidable?
  - $\theta_1 \equiv_{\beta\eta} \theta_2$ is decidable for well-kinded types (by normalization)
  - regular-tree grammar intersection/emptiness not directly applicable

Future Work

Extend type- and control-flow analysis to System F_ω:

$$\mathit{Kind} \ni \kappa ::= \star \mid \kappa \Rightarrow \kappa \qquad \mathit{Type} \ni \tau ::= \tau \to \tau \mid \alpha \mid \forall\alpha{:}\kappa.\ \tau \mid \lambda\alpha{:}\kappa.\ \tau \mid \tau\ \tau$$

- flow soundness with modified type-compatibility judgement:

$$\frac{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \twoheadrightarrow \theta_1 \qquad \hat{\phi} \vDash_{\mathsf{S}} \tau_2 \twoheadrightarrow \theta_2 \qquad \theta_1 \equiv_{\beta\eta} \theta_2}{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approxeq \tau_2}$$

- is type compatibility decidable?
  - run a control-flow analysis over types to approximate applications; reduce to regular-tree grammar intersection/emptiness, with a loss of precision and completeness (wrt. type-compatibility judgement)

Future Work

Extend type- and control-flow analysis to System F_ω:

$$\mathit{Kind} \ni \kappa ::= \star \mid \kappa \Rightarrow \kappa \qquad \mathit{Type} \ni \tau ::= \tau \to \tau \mid \alpha \mid \forall\alpha{:}\kappa.\ \tau \mid \lambda\alpha{:}\kappa.\ \tau \mid \tau\ \tau$$

- flow soundness with modified type-compatibility judgement:

$$\frac{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \twoheadrightarrow \theta_1 \qquad \hat{\phi} \vDash_{\mathsf{S}} \tau_2 \twoheadrightarrow \theta_2 \qquad \theta_1 \equiv_{\beta\eta} \theta_2}{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2}$$

- is type compatibility decidable?
  - algorithm for regular-tree grammar intersection somewhat similar to type equivalence with type definitions by weak-head normalization; combine the two algorithms in a decision procedure?

Future Work

Extend type- and control-flow analysis to System F_ω:

$$\textit{Kind} \ni \kappa ::= \star \mid \kappa \Rightarrow \kappa \qquad \textit{Type} \ni \tau ::= \tau \to \tau \mid \alpha \mid \forall\alpha{:}\kappa.\ \tau \mid \lambda\alpha{:}\kappa.\ \tau \mid \tau\ \tau$$

- flow soundness with modified type-compatibility judgement:

$$\frac{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \twoheadrightarrow \theta_1 \qquad \hat{\phi} \vDash_{\mathsf{S}} \tau_2 \twoheadrightarrow \theta_2 \qquad \theta_1 \equiv_{\beta\eta} \theta_2}{\hat{\phi} \vDash_{\mathsf{S}} \tau_1 \approxeq \tau_2}$$

- is type compatibility decidable?
  - canonizing substitution is alternative approach to type equivalence, maintaining types in a canonical β-normal/η-long form; exploit this added structure in a decision procedure?

Future Work

Applications of type- and control-flow analysis:

- Flow Directed Defunctionalization of System F
  - compile higher-order polymorphic source language
    to first-order polymorphic target language
  - type- and control-flow analysis to minimize dispatches

  - Flow-directed Closure Conversion [Cejtin, Jagannathan, & Weeks (ESOP'00)]
  - Polymorphic Typed Defunctionalization [Pottier & Gauthier (POPL'04)]
  - Defunctionalizing Polymorphic Types [Midtgaard (PhD Dissertation'07)]

Type- and Control-Flow Analysis for System F

- Exploit types to obtain more precise control-flow information.
- Type-flow and control-flow are mutually beneficial.
- Sound analysis via specification-based formulation.
- Computable analysis via interpretation as regular-tree grammar.
- Efficient algorithm via flow-graph-based formulation.
- Many directions for future work.

Questions?