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.

$$f1 = \lambda \times 1 \cdot \cdots$$

$$f2 = \lambda \times 2 \cdot \cdots$$

id =
$$\lambda \times . \times$$

res1	=	id	f1
res2	=	id	f2

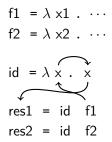
$$f1 = \lambda \times 1 \dots$$

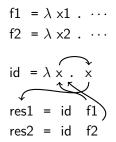
$$f2 = \lambda \times 2 \dots$$

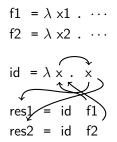
$$id = \lambda \times . \times$$

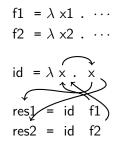
res1 = id f1
res2 = id f2

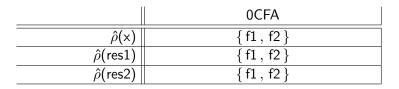
$$f1 = \lambda \times 1 \dots \\ f2 = \lambda \times 2 \dots \\ id = \lambda \times . \times \\ res1 = id f1 \\ res2 = id f2$$







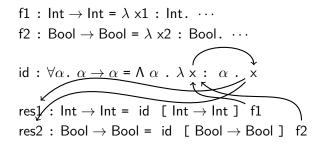


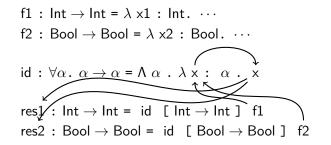


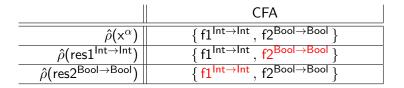
0CFA — "classic" monovariant/context-insensitive control-flow analysis

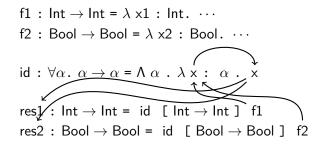
f1 : Int \rightarrow Int = λ x1 : Int. ... f2 : Bool \rightarrow Bool = λ x2 : Bool. ...

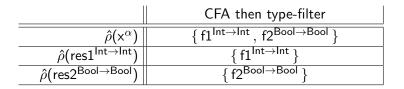
$$\mathsf{id} \, : \, \forall \alpha . \, \, \alpha \to \alpha = \Lambda \, \alpha \, \, . \, \, \lambda \, \mathsf{x} \, : \, \, \alpha \, \, . \, \, \mathsf{x}$$



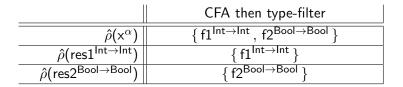


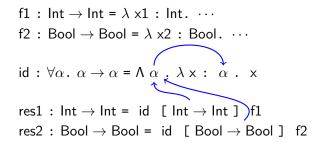


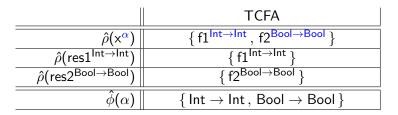




$$\begin{array}{l} {\rm f1}\,:\,{\rm Int}\rightarrow{\rm Int}=\lambda\,\,{\rm x1}\,:\,{\rm Int.}\,\,\cdots\\ {\rm f2}\,:\,{\rm Bool}\rightarrow{\rm Bool}=\lambda\,\,{\rm x2}\,:\,{\rm Bool.}\,\,\cdots\\ {\rm id}\,:\,\forall\alpha.\,\,\alpha\rightarrow\alpha=\Lambda\,\,\alpha\,\,.\,\,\lambda\,\,{\rm x}\,:\,\,\alpha\,\,.\,\,{\rm x}\\ {\rm res1}\,:\,{\rm Int}\rightarrow{\rm Int}=\,{\rm id}\,\,\left[\,\,{\rm Int}\rightarrow{\rm Int}\,\,\right]\,\,{\rm f1}\\ {\rm res2}\,:\,{\rm Bool}\rightarrow{\rm Bool}=\,{\rm id}\,\,\left[\,\,{\rm Bool}\rightarrow{\rm Bool}\,\,\right]\,\,{\rm f2} \end{array}$$







$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ &\Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

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

 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \; h1 \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \; h2 \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \; [{\sf str}] \; [{\sf int} \rightarrow {\sf int}] \; h3 \; "zzz" \end{array}$

$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ & \Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

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

 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \ [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \ h1 \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \ [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \ h2 \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \ [{\sf str}] \ [{\sf int} \rightarrow {\sf int}] \ h3 \ "zzz" \end{array}$

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

$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ &\Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

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

 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \ [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \ h1 \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \ [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \ h2 \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \ [{\sf str}] \ [{\sf int} \rightarrow {\sf int}] \ h3 \ "zzz" \end{array}$

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

$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ &\Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

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

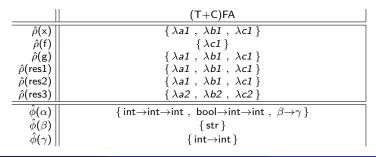
 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \ [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \ h1 \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \ [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \ h2 \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \ [{\sf str}] \ [{\sf int} \rightarrow {\sf int}] \ h3 \ "zzz" \end{array}$

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

$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ & \Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

 $\begin{array}{l} \texttt{h1:int} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{a1:int}, \lambda\texttt{a2:int}, \texttt{a1+a2} \\ \texttt{h2:bool} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{b1:bool}, \lambda\texttt{b2:int}, \texttt{if b1 then b2+1 else b2} \\ \texttt{h3:str} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{c1:str}, \lambda\texttt{c2:int}, \texttt{len(c1)+c2} \end{array}$

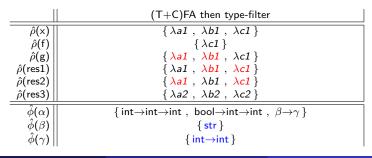
 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h1} \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h2} \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \; [{\sf str}] \; [{\sf int} \rightarrow {\sf int}] \; {\sf h3} \; "zzz" \end{array}$



$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ & \Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

 $\begin{array}{l} \texttt{h1:int} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{a1:int}, \lambda\texttt{a2:int}, \texttt{a1+a2} \\ \texttt{h2:bool} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{b1:bool}, \lambda\texttt{b2:int}, \texttt{if b1 then b2+1 else b2} \\ \texttt{h3:str} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{c1:str}, \lambda\texttt{c2:int}, \texttt{len(c1)+c2} \end{array}$

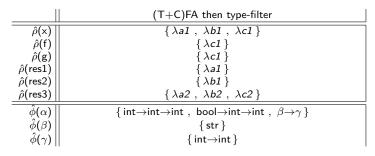
 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h1} \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h2} \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \; [{\sf str}] \; [{\sf int} \rightarrow {\sf int}] \; {\sf h3} \; "zzz" \end{array}$



$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ & \Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

 $\begin{array}{l} \texttt{h1:int} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{a1:int}, \lambda\texttt{a2:int}, \texttt{a1+a2} \\ \texttt{h2:bool} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{b1:bool}, \lambda\texttt{b2:int}, \texttt{if b1 then b2+1 else b2} \\ \texttt{h3:str} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{c1:str}, \lambda\texttt{c2:int}, \texttt{len(c1)+c2} \end{array}$

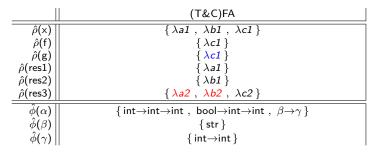
 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h1} \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h2} \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \; [{\sf str}] \; [{\sf int} \rightarrow {\sf int}] \; {\sf h3} \; "zzz" \end{array}$



$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ & \Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

 $\begin{array}{l} \texttt{h1:int} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{a1:int}, \lambda\texttt{a2:int}, \texttt{a1+a2} \\ \texttt{h2:bool} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{b1:bool}, \lambda\texttt{b2:int}, \texttt{if b1 then b2+1 else b2} \\ \texttt{h3:str} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{c1:str}, \lambda\texttt{c2:int}, \texttt{len(c1)+c2} \end{array}$

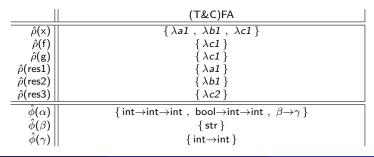
 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h1} \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h2} \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \; [{\sf str}] \; [{\sf int} \rightarrow {\sf int}] \; {\sf h3} \; "zzz" \end{array}$



$$\begin{split} &\text{id}: \forall \alpha. \ \alpha \to \alpha = \Lambda \alpha. \ \lambda x: \alpha. \ x \\ &\text{app}: \forall \beta. \ \forall \gamma. \ (\beta \to \gamma) \to \beta \to \gamma = \\ & \Lambda \beta. \Lambda \gamma. \lambda f: \beta \to \gamma. \lambda z: \beta. \ \text{let} \ g: \beta \to \gamma = \text{id} \ [\beta \to \gamma] \ f \ \text{in} \ g \ z \end{split}$$

 $\begin{array}{l} \texttt{h1:int} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{a1:int}, \lambda\texttt{a2:int}, \texttt{a1+a2} \\ \texttt{h2:bool} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{b1:bool}, \lambda\texttt{b2:int}, \texttt{if b1 then b2+1 else b2} \\ \texttt{h3:str} \rightarrow \texttt{int} \rightarrow \texttt{int} = \lambda\texttt{c1:str}, \lambda\texttt{c2:int}, \texttt{len(c1)+c2} \end{array}$

 $\begin{array}{l} {\sf res1:int} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf int} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h1} \\ {\sf res2:bool} \rightarrow {\sf int} \rightarrow {\sf int} = {\sf id} \; [{\sf bool} \rightarrow {\sf int} \rightarrow {\sf int}] \; {\sf h2} \\ {\sf res3:int} \rightarrow {\sf int} = {\sf app} \; [{\sf str}] \; [{\sf int} \rightarrow {\sf int}] \; {\sf h3} \; "zzz" \end{array}$



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

determines type abstractions flowing to type applications

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

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., typecase α of \cdots)

• 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.

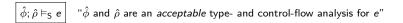
- Soundness, Algorithm, Complexity
- Related and Future Work

$$\hat{\phi}; \hat{\rho} \vDash_{\mathsf{S}} e$$
 " $\hat{\phi}$ 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.

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



Specification-based Formulation: Type Compatibility

$$\widehat{\phi} \vDash_{\mathsf{S}} \tau_1 \approx \tau_2 \quad \text{``} \tau_1 \text{ and } \tau_2 \text{ are } \textit{compatible under } \widehat{\phi}$$

Specification-based Formulation: Type Compatibility

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

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

Specification-based Formulation: Type Compatibility

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

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

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

Specification-based Formulation: Type Compatibility

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

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

Is
$$\hat{\phi} \vDash_{S} \tau_{1} \approx \tau_{2}$$
 decidable?

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

Specification-based Formulation: Type Compatibility

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

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

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

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

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

Specification-based Formulation: Type Compatibility

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

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

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

Interpret $\hat{\phi}$ as (productions for) a *regular-tree grammar*, a derivation of $\hat{\phi} \models_{\mathsf{S}} \alpha \Longrightarrow \theta$ 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 $\hat{\phi}$; $\hat{\rho} \vDash_{S} e$ computable by lfp iteration.

	Specification-Based (naïve lfp iteration)	
"classic" CFA	<i>O</i> (<i>n</i> ⁵)	
TCFA		

	Specification-Based (naïve lfp iteration)	
"classic" CFA	$O(n^5)$	
TCFA	<i>O</i> (<i>n</i> ¹³)	

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

	Specification-Based (naïve Ifp iteration)	Flow-Graph-Based (work-queue iteration)
"classic" CFA	$O(n^5)$	<i>O</i> (<i>n</i> ³)
TCFA	$O(n^{13}) \ O(n^{10})$ amortized	

	Specification-Based (naïve Ifp iteration)	Flow-Graph-Based (work-queue iteration)
"classic" CFA	$O(n^5)$	<i>O</i> (<i>n</i> ³)
TCFA	$O(n^{13}) \ O(n^{10})$ amortized	<i>O</i> (<i>n</i> ⁴)

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

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

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

Fluet (RIT)

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

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

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

Fluet (RIT)

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

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

- 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 \to int funs, all sharing the same (top-level) type def

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

- 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 \to int funs, all sharing the same (top-level) type def
- Not bad in theory; much better in practice

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

- 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)

- \bullet For simply-typed $\lambda\text{-calculus},$ 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)
 - $\bullet\,$ Suggests mapping polymorphism to $\top\,$

• Improve precision of type- and control-flow analysis

 \bullet Extend type- and control-flow to System F_ω

• Applications of type- and control-flow analysis

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

• . . .

Improve precision of type- and control-flow analysis:

• Explore improvements motivated by well-typedness:

$$\underbrace{\bigwedge_{\lambda_{Z}:\tau_{z}\,\cdot\,e_{b}\in\hat{\rho}(x_{f})} \begin{pmatrix} \forall v_{a}\in\hat{\rho}(x_{a})\,\cdot\,\hat{\phi}\vDash_{S}\operatorname{TyOf}(v)\otimes\tau_{z}\Rightarrow v_{a}\in\hat{\rho}(z)\wedge\\ \begin{pmatrix} \exists v_{a}\in\hat{\rho}(x_{a})\,\cdot\,\hat{\phi}\vDash_{S}\operatorname{TyOf}(v)\otimes\tau_{z}\Rightarrow\\ \forall v_{b}\in\hat{\rho}(\operatorname{ResOf}(e_{b}))\,\cdot\,\hat{\phi}\vDash_{S}\operatorname{TyOf}(v)\otimes\tau_{x}\Rightarrow v_{b}\in\hat{\rho}(x) \end{pmatrix}}_{\hat{\phi};\,\hat{\rho}\vDash_{S}\operatorname{let} x:\alpha_{x}=x_{f}\,x_{a}\operatorname{in} e}$$

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

Improve precision of type- and control-flow analysis:

• Explore improvements motivated by well-typedness:

$$\label{eq:recloop} \begin{split} &\operatorname{rec}\,\operatorname{loop}\,=\Lambda\alpha.\;\lambda {\bf x}\!:\!\alpha.\;\operatorname{loop}\,\left[\alpha\times\alpha\right]\;({\bf x},{\bf x})\\ &\operatorname{loop}\,\left[\operatorname{Int}\right]\;\mathbf{1} \end{split}$$

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

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

Extend type- and control-flow analysis to System F_{ω} :

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

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

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:

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

• is type compatibility decidable?

Extend type- and control-flow analysis to System F_{ω} :

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

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

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

Extend type- and control-flow analysis to System F_{ω} :

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

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

- is type compatibility decidable?
 - 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)

Extend type- and control-flow analysis to System F_{ω} :

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

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

- 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?

Extend type- and control-flow analysis to System F_{ω} :

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

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

- is type compatibility decidable?
 - 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?

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?