



Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Annotated Type Systems

Slides from Stefan Holdermans and Jurriaan Hage

Dept. of Information and Computing Sciences, Utrecht University

P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

E-mail: i.g.dewolff@uu.nl

Type and effect systems - Introduction



Static program analysis

- ▶ Static program analysis: **compile-time** techniques for **approximating** the set of values or behaviours that arise at run-time when a program is executed.
- ▶ Applications: **verification**, **optimization**.
- ▶ Different approaches: data-flow analysis, constraint-based analysis, abstract interpretation, **type-based analysis**.



Previously: monotone frameworks

- ▶ Propagate information over control flow graph.
- ▶ For procedures (interprocedural), **embellished instances** to propagate over balanced (valid) paths
- ▶ Procedures are analyzed per context.
- ▶ Targets of calls were statically known. How about languages with **higher order functions**?

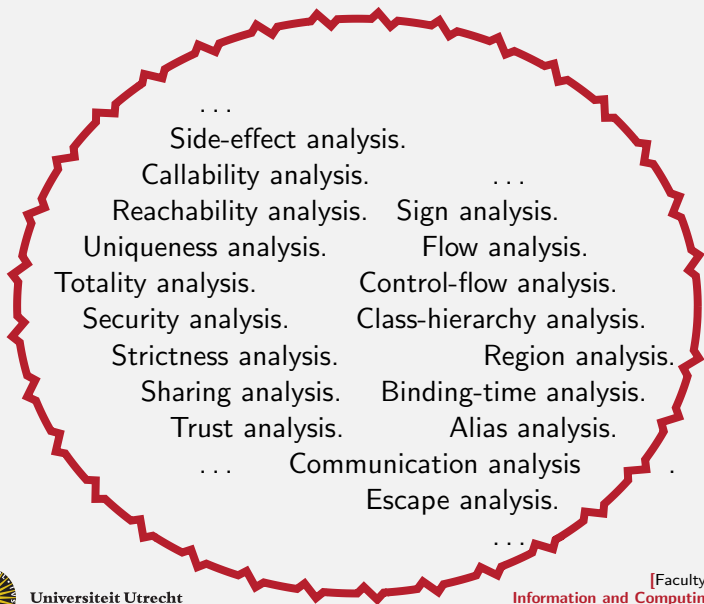


Type-based approaches to static program analysis

- ▶ Type-based analysis: equipping a programming language with a **nonstandard type system** that keeps track of some properties of interest.
- ▶ Advantages: reuse of **tools**, **techniques**, and **infrastructure** (polymorphism, subtyping, type inference, ...).

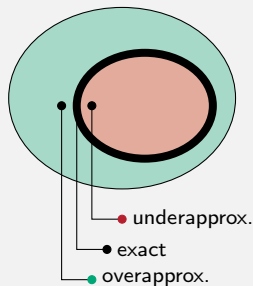


Examples



Accuracy

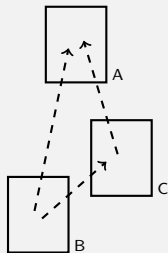
- ▶ Establishing nontrivial behavioural properties of programs is in general **undecidable** (halting problem, Rice's theorem).
- ▶ In static analysis we have to settle for “useful” **approximations** of properties.
- ▶ “Useful” means: **sound** (“erring at the safe side”) and **accurate** (as precise as possible).



Modularity

- ▶ Breaking up a (large) program in smaller units or **modules** is generally considered good programming style.
- ▶ **Separate compilation**: compile each module in isolation.
- ▶ Advantage: only modules that have been edited need to be **recompiled**.
- ▶ To facilitate separate compilation, each unit of compilation needs to be analysed in isolation, i.e., without knowledge of how it's **used** from within the rest of the program.

👉 Tension between **accuracy** and **modularity**: whole-program analysis typically yields more precise results.



Roadmap

- ▶ Type system formalization (Hindley-Milner)
- ▶ Inference algorithm (Algorithm W)
- ▶ Annotated type systems



Hindley-Milner and Algorithm W



A simple functional language

$f, x \in \mathbf{Var}$ variables

$t \in \mathbf{Tm}$ terms

$t ::=$ $| x | \lambda x. t_1$
 $| t_1 t_2$
 $|$



A simple functional language

$f, x \in \mathbf{Var}$ variables

$\pi \in \mathbf{Pnt}$ program points

$t \in \mathbf{Tm}$ terms

$$t ::= \quad \mid x \mid \lambda_{\pi} x. t_1$$
$$\quad \mid t_1 t_2$$
$$\quad \mid$$


A simple functional language

$f, x \in \mathbf{Var}$ variables

$\pi \in \mathbf{Pnt}$ program points

$t \in \mathbf{Tm}$ terms

$$t ::= \quad \mid x \mid \lambda_{\pi} x. t_1$$

$$\quad \mid t_1 t_2 \quad \mid \mathbf{let } x = t_1 \mathbf{ in } t_2$$

$$\quad \mid$$


A simple functional language

$f, x \in \mathbf{Var}$ variables

$\pi \in \mathbf{Pnt}$ program points

$t \in \mathbf{Tm}$ terms

$t ::=$

	x	$\lambda_{\pi} x. t_1$	$\mu f. \lambda_{\pi} x. t_1$
	$t_1 t_2$	let $x = t_1$ in t_2	



A simple functional language

n	\in	Num = \mathbb{N}	numerals
f, x	\in	Var	variables
π	\in	Pnt	program points
t	\in	Tm	terms

t	$::=$	n		x		$\lambda_{\pi} x. t_1$		$\mu f. \lambda_{\pi} x. t_1$	
									let $x = t_1$ in t_2



A simple functional language

n	\in	Num = \mathbb{N}	numerals
f, x	\in	Var	variables
π	\in	Pnt	program points
t	\in	Tm	terms

$t ::= n \mid \text{false} \mid \text{true} \mid x \mid \lambda_{\pi} x. t_1 \mid \mu f. \lambda_{\pi} x. t_1$
 $\quad \mid t_1 t_2 \mid \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid \text{let } x = t_1 \text{ in } t_2$
 $\quad \mid$



A simple functional language

n	\in	Num = \mathbb{N}	numerals
f, x	\in	Var	variables
\oplus	\in	Op	binary operators
π	\in	Pnt	program points
t	\in	Tm	terms

$t ::= n \mid \text{false} \mid \text{true} \mid x \mid \lambda_{\pi}x. t_1 \mid \mu f. \lambda_{\pi}x. t_1$
| $t_1 t_2 \mid \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid \text{let } x = t_1 \text{ in } t_2$
| $t_1 \oplus t_2$



A simple functional language

n	\in	Num = \mathbb{N}	numerals
f, x	\in	Var	variables
\oplus	\in	Op	binary operators
π	\in	Pnt	program points
t	\in	Tm	terms

$t ::= n \mid \text{false} \mid \text{true} \mid x \mid \lambda_{\pi}x. t_1 \mid \mu f. \lambda_{\pi}x. t_1$
 $\quad \mid t_1 t_2 \mid \text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid \text{let } x = t_1 \text{ in } t_2$
 $\quad \mid t_1 \oplus t_2$

Example:

let $fac = \mu f. \lambda_{\mathbf{F}}x. \text{if } x \equiv 0 \text{ then } 1 \text{ else } x * f (x - 1)$
in $fac\ 6$



Monomorphic types

$\tau \in \mathbf{Ty}$ types

$\tau ::= \mathit{Nat} \mid \mathit{Bool} \mid \tau_1 \rightarrow \tau_2$



Monomorphic types

$\tau \in \mathbf{Ty}$ types
 $\Gamma \in \mathbf{TyEnv}$ type environments

$\tau ::= \mathit{Nat} \mid \mathit{Bool} \mid \tau_1 \rightarrow \tau_2$
 $\Gamma ::= [] \mid \Gamma_1[x \mapsto \tau]$



Monomorphic types

$\tau \in \mathbf{Ty}$ types
 $\Gamma \in \mathbf{TyEnv}$ type environments

$\tau ::= \mathit{Nat} \mid \mathit{Bool} \mid \tau_1 \rightarrow \tau_2$
 $\Gamma ::= [] \mid \Gamma_1[x \mapsto \tau]$

Typing judgements:

$\Gamma \vdash_{\text{UL}} t : \tau$ typing

“Term t has type τ assuming that any of its free variables has the type given by Γ .”



Monomorphic type system: constants

$$\frac{}{\Gamma \vdash_{\text{UL}} n : \text{Nat}} \quad [t\text{-num}]$$



Monomorphic type system: constants

$$\frac{}{\Gamma \vdash_{\text{UL}} n : \text{Nat}} \quad [t\text{-num}]$$

$$\frac{}{\Gamma \vdash_{\text{UL}} \text{false} : \text{Bool}} \quad [t\text{-false}]$$

$$\frac{}{\Gamma \vdash_{\text{UL}} \text{true} : \text{Bool}} \quad [t\text{-true}]$$



Monomorphic type system: variables

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash_{\text{UL}} x : \tau} \text{ [t-var]}$$



Monomorphic type system: functions

$$\frac{\Gamma[x \mapsto \tau_1] \vdash_{\text{UL}} t_1 : \tau_2}{\Gamma \vdash_{\text{UL}} \lambda_{\pi} x. t_1 : \tau_1 \rightarrow \tau_2} \quad [t\text{-lam}]$$



Monomorphic type system: functions

$$\frac{\Gamma[x \mapsto \tau_1] \vdash_{\text{UL}} t_1 : \tau_2}{\Gamma \vdash_{\text{UL}} \lambda_{\pi} x. t_1 : \tau_1 \rightarrow \tau_2} \quad [t\text{-lam}]$$

$$\frac{\Gamma[f \mapsto (\tau_1 \rightarrow \tau_2)][x \mapsto \tau_1] \vdash_{\text{UL}} t_1 : \tau_2}{\Gamma \vdash_{\text{UL}} \mu f. \lambda_{\pi} x. t_1 : \tau_1 \rightarrow \tau_2} \quad [t\text{-mu}]$$



Monomorphic type system: functions

$$\frac{\Gamma[x \mapsto \tau_1] \vdash_{\text{UL}} t_1 : \tau_2}{\Gamma \vdash_{\text{UL}} \lambda_{\pi} x. t_1 : \tau_1 \rightarrow \tau_2} \quad [t\text{-lam}]$$

$$\frac{\Gamma[f \mapsto (\tau_1 \rightarrow \tau_2)][x \mapsto \tau_1] \vdash_{\text{UL}} t_1 : \tau_2}{\Gamma \vdash_{\text{UL}} \mu f. \lambda_{\pi} x. t_1 : \tau_1 \rightarrow \tau_2} \quad [t\text{-mu}]$$

$$\frac{\Gamma \vdash_{\text{UL}} t_1 : \tau_2 \rightarrow \tau \quad \Gamma \vdash_{\text{UL}} t_2 : \tau_2}{\Gamma \vdash_{\text{UL}} t_1 t_2 : \tau} \quad [t\text{-app}]$$



Monomorphic type system: conditionals

$$\frac{\Gamma \vdash_{\text{UL}} t_1 : \textit{Bool} \quad \Gamma \vdash_{\text{UL}} t_2 : \tau \quad \Gamma \vdash_{\text{UL}} t_3 : \tau}{\Gamma \vdash_{\text{UL}} \textit{if } t_1 \textit{ then } t_2 \textit{ else } t_3 : \tau} \textit{[t-if]}$$



Monomorphic type system: local definitions

$$\frac{\Gamma \vdash_{\text{UL}} t_1 : \tau_1 \quad \Gamma[x \mapsto \tau_1] \vdash_{\text{UL}} t_2 : \tau}{\Gamma \vdash_{\text{UL}} \mathbf{let} \ x = t_1 \ \mathbf{in} \ t_2 : \tau} \quad [t\text{-let}]$$



Monomorphic type system: binary operators

$$\frac{\Gamma \vdash_{\text{UL}} t_1 : \tau_{\oplus}^1 \quad \Gamma \vdash_{\text{UL}} t_2 : \tau_{\oplus}^2}{\Gamma \vdash_{\text{UL}} t_1 \oplus t_2 : \tau_{\oplus}} [t\text{-op}]$$



Monomorphic type system: example

$$\Gamma \vdash_{\text{UL}} \mu f. \lambda_{\text{F}} x. \text{if } x \equiv 0 \text{ then } 1 \text{ else } x * f (x - 1) : \text{Nat} \rightarrow \text{Nat}$$


Monomorphic type system: example

$$\frac{\begin{array}{c} \vdots \\ \hline \Gamma_{\mathbf{F}} \vdash_{\text{UL}} x \equiv 0 : \mathit{Bool} \quad \Gamma_{\mathbf{F}} \vdash_{\text{UL}} 1 : \mathit{Nat} \quad \Gamma_{\mathbf{F}} \vdash_{\text{UL}} x * f (x - 1) : \mathit{Nat} \end{array}}{\Gamma_{\mathbf{F}} \vdash_{\text{UL}} \mathbf{if } x \equiv 0 \mathbf{ then } 1 \mathbf{ else } x * f (x - 1) : \mathit{Nat}}$$
$$\frac{\Gamma \vdash_{\text{UL}} \mu f. \lambda_{\mathbf{F}} x. \mathbf{if } x \equiv 0 \mathbf{ then } 1 \mathbf{ else } x * f (x - 1) : \mathit{Nat} \rightarrow \mathit{Nat}}$$

$$\Gamma_{\mathbf{F}} = \Gamma[f \mapsto (\mathit{Nat} \rightarrow \mathit{Nat})][x \mapsto \mathit{Nat}]$$



Polymorphic functions



Polymorphic functions

$$\lambda_{\mathbb{F}} x. x$$


Polymorphic functions

$$\lambda_{\mathbf{F}}x. x$$
$$\lambda_{\mathbf{F}}x. \lambda_{\mathbf{G}}y. x$$


Polymorphic functions

$$\lambda_{\mathbf{F}}x. x$$
$$\lambda_{\mathbf{F}}x. \lambda_{\mathbf{G}}y. x$$
$$\lambda_{\mathbf{F}}f. \lambda_{\mathbf{G}}x. f x$$


Polymorphic functions

$$\lambda_{\mathbf{F}}x. x$$
$$\lambda_{\mathbf{F}}x. \lambda_{\mathbf{G}}y. x$$
$$\lambda_{\mathbf{F}}f. \lambda_{\mathbf{G}}x. f x$$
$$\mu f. \lambda_{\mathbf{F}}g. \lambda_{\mathbf{G}}x. \lambda_{\mathbf{H}}y. \mathbf{if } x \equiv 0 \mathbf{ then } y \mathbf{ else } f g (x - 1) (g y)$$


Polymorphic types

$\tau \in \mathbf{Ty}$ types

$\Gamma \in \mathbf{TyEnv}$ type environments

$\tau ::= \quad | \mathit{Nat} | \mathit{Bool} | \tau_1 \rightarrow \tau_2$

$\Gamma ::= [] | \Gamma_1[x \mapsto \tau]$

$\Gamma \vdash_{\text{UL}} t : \tau$ typing



Polymorphic types

$\alpha \in \mathbf{TyVar}$ type variables

$\tau \in \mathbf{Ty}$ types

$\Gamma \in \mathbf{TyEnv}$ type environments

$\tau ::= \alpha \mid \mathit{Nat} \mid \mathit{Bool} \mid \tau_1 \rightarrow \tau_2$

$\Gamma ::= [] \mid \Gamma_1[x \mapsto \tau]$

$\Gamma \vdash_{\text{UL}} t : \tau$ typing



Polymorphic types

α	\in	TyVar	type variables
τ	\in	Ty	types
σ	\in	TyScheme	type schemes
Γ	\in	TyEnv	type environments

τ	$::=$	α <i>Nat</i> <i>Bool</i> $\tau_1 \rightarrow \tau_2$
σ	$::=$	τ $\forall\alpha. \sigma_1$
Γ	$::=$	$[]$ $\Gamma_1[x \mapsto \tau]$

$\Gamma \vdash_{\text{UL}} t : \tau$ typing



Polymorphic types

α	\in	TyVar	type variables
τ	\in	Ty	types
σ	\in	TyScheme	type schemes
Γ	\in	TyEnv	type environments

τ	$::=$	$\alpha \mid Nat \mid Bool \mid \tau_1 \rightarrow \tau_2$
σ	$::=$	$\tau \mid \forall \alpha. \sigma_1$
Γ	$::=$	$[] \mid \Gamma_1[x \mapsto \sigma]$

$\Gamma \vdash_{UL} t : \tau$ typing



Polymorphic types

α	\in	TyVar	type variables
τ	\in	Ty	types
σ	\in	TyScheme	type schemes
Γ	\in	TyEnv	type environments

τ	$::=$	$\alpha \mid Nat \mid Bool \mid \tau_1 \rightarrow \tau_2$
σ	$::=$	$\tau \mid \forall \alpha. \sigma_1$
Γ	$::=$	$[] \mid \Gamma_1[x \mapsto \sigma]$

Γ	\vdash_{UL}	$t : \sigma$	typing
----------	---------------	--------------	--------




Polymorphic types

α	\in	TyVar	type variables
τ	\in	Ty	types
σ	\in	TyScheme	type schemes
Γ	\in	TyEnv	type environments

τ	$::=$	$\alpha \mid Nat \mid Bool \mid \tau_1 \rightarrow \tau_2$
σ	$::=$	$\tau \mid \forall \alpha. \sigma_1$
Γ	$::=$	$[] \mid \Gamma_1[x \mapsto \sigma]$

$\Gamma \vdash_{UL} t : \sigma$ typing

 **Ty** \subseteq **TyScheme**



Polymorphic type system: generalisation and instantiation

Introduction:

$$\frac{\Gamma \vdash_{\text{UL}} t : \sigma_1 \quad \alpha \notin \text{ftv}(\Gamma)}{\Gamma \vdash_{\text{UL}} t : \forall \alpha. \sigma_1} \quad [t\text{-gen}]$$



Polymorphic type system: generalisation and instantiation

Introduction:

$$\frac{\Gamma \vdash_{\text{UL}} t : \sigma_1 \quad \alpha \notin \text{ftv}(\Gamma)}{\Gamma \vdash_{\text{UL}} t : \forall \alpha. \sigma_1} \quad [t\text{-gen}]$$

Elimination:

$$\frac{\Gamma \vdash_{\text{UL}} t : \forall \alpha. \sigma_1}{\Gamma \vdash_{\text{UL}} t : [\alpha \mapsto \tau_0] \sigma_1} \quad [t\text{-inst}]$$



Polymorphic type system: variables and local definitions

$$\frac{\Gamma(x) = \sigma}{\Gamma \vdash_{\text{UL}} x : \sigma} \text{ [t-var]}$$



Polymorphic type system: variables and local definitions

$$\frac{\Gamma(x) = \sigma}{\Gamma \vdash_{\text{UL}} x : \sigma} \quad [t\text{-var}]$$

$$\frac{\Gamma \vdash_{\text{UL}} t_1 : \sigma_1 \quad \Gamma[x \mapsto \sigma_1] \vdash_{\text{UL}} t_2 : \tau}{\Gamma \vdash_{\text{UL}} \mathbf{let} \ x = t_1 \ \mathbf{in} \ t_2 : \tau} \quad [t\text{-let}]$$



Polymorphic types: example

$$\lambda_{\mathbf{F}}x. x : \forall\alpha. \alpha \rightarrow \alpha$$
$$\lambda_{\mathbf{F}}x. \lambda_{\mathbf{G}}y. x : \forall\alpha_1. \forall\alpha_2. \alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_1$$
$$\lambda_{\mathbf{F}}f. \lambda_{\mathbf{G}}x. f x : \forall\alpha_1. \forall\alpha_2. (\alpha_1 \rightarrow \alpha_2) \rightarrow \alpha_1 \rightarrow \alpha_2$$
$$\begin{aligned} \mu f. \lambda_{\mathbf{F}}g. \lambda_{\mathbf{G}}x. \lambda_{\mathbf{H}}y. \text{if } x \equiv 0 \text{ then } y \text{ else } f g (x - 1) (g y) \\ : \forall\alpha. (\alpha \rightarrow \alpha) \rightarrow \text{Nat} \rightarrow \alpha \rightarrow \alpha \end{aligned}$$


Inference algorithm

$\theta \in \mathbf{TySubst} = \mathbf{TyVar} \rightarrow_{\text{fin}} \mathbf{Ty}$ type substitution

$generalise_{UL} : \mathbf{TyEnv} \times \mathbf{Ty} \rightarrow \mathbf{TyScheme}$

$instantiate_{UL} : \mathbf{TyScheme} \rightarrow \mathbf{Ty}$

$\mathcal{U}_{UL} : \mathbf{Ty} \times \mathbf{Ty} \rightarrow \mathbf{TySubst}$

$\mathcal{W}_{UL} : \mathbf{TyEnv} \times \mathbf{Tm} \rightarrow \mathbf{Ty} \times \mathbf{TySubst}$



Inference algorithm: constants

$$\mathcal{W}_{\text{UL}}(\Gamma, n) = (\text{Nat}, \text{id})$$



Inference algorithm: constants

$$\mathcal{W}_{\text{UL}}(\Gamma, n) = (\text{Nat}, \text{id})$$

$$\mathcal{W}_{\text{UL}}(\Gamma, \text{false}) = (\text{Bool}, \text{id})$$

$$\mathcal{W}_{\text{UL}}(\Gamma, \text{true}) = (\text{Bool}, \text{id})$$



Inference algorithm: variables

$$\mathcal{W}_{\text{UL}}(\Gamma, x) = (\textit{instantiate}_{\text{UL}}(\Gamma(x)), \textit{id})$$

- ▶ The instantiation rule is built into the case for variables.
- ▶ By choosing fresh type variables, we commit to nothing,
- ▶ and let the actual types be determined by future unifications.



Inference algorithm: functions

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, \lambda_{\pi} x. t_1) = & \text{let } \alpha_1 \text{ be fresh} \\ & (\tau_2, \theta) = \mathcal{W}_{\text{UL}}(\Gamma[x \mapsto \alpha_1], t_1) \\ & \text{in } ((\theta \alpha_1) \rightarrow \tau_2, \theta) \end{aligned}$$



Inference algorithm: functions

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, \lambda_{\pi} x. t_1) = & \text{let } \alpha_1 \text{ be fresh} \\ & (\tau_2, \theta) = \mathcal{W}_{\text{UL}}(\Gamma[x \mapsto \alpha_1], t_1) \\ & \text{in } ((\theta \alpha_1) \rightarrow \tau_2, \theta) \end{aligned}$$

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, \mu f. \lambda_{\pi} x. t_1) = & \\ & \text{let } \alpha_1, \alpha_2 \text{ be fresh} \\ & (\tau_2, \theta_1) = \mathcal{W}_{\text{UL}}(\Gamma[f \mapsto (\alpha_1 \rightarrow \alpha_2)][x \mapsto \alpha_1], t_1) \\ & \theta_2 = \mathcal{U}_{\text{UL}}(\tau_2, \theta_1 \alpha_2) \\ & \text{in } (\theta_2 (\theta_1 \alpha_1) \rightarrow \theta_2 \tau_2, \theta_2 \circ \theta_1) \end{aligned}$$



Inference algorithm: functions

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, \lambda_{\pi} x. t_1) &= \text{let } \alpha_1 \text{ be fresh} \\ &\quad (\tau_2, \theta) = \mathcal{W}_{\text{UL}}(\Gamma[x \mapsto \alpha_1], t_1) \\ &\quad \text{in } ((\theta \alpha_1) \rightarrow \tau_2, \theta) \end{aligned}$$

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, \mu f. \lambda_{\pi} x. t_1) &= \\ \text{let } \alpha_1, \alpha_2 \text{ be fresh} & \\ (\tau_2, \theta_1) = \mathcal{W}_{\text{UL}}(\Gamma[f \mapsto (\alpha_1 \rightarrow \alpha_2)][x \mapsto \alpha_1], t_1) & \\ \theta_2 = \mathcal{U}_{\text{UL}}(\tau_2, \theta_1 \alpha_2) & \\ \text{in } (\theta_2 (\theta_1 \alpha_1) \rightarrow \theta_2 \tau_2, \theta_2 \circ \theta_1) & \end{aligned}$$

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, t_1 t_2) &= \text{let } (\tau_1, \theta_1) = \mathcal{W}_{\text{UL}}(\Gamma, t_1) \\ &\quad (\tau_2, \theta_2) = \mathcal{W}_{\text{UL}}(\theta_1 \Gamma, t_2) \\ &\quad \alpha \text{ be fresh} \\ &\quad \theta_3 = \mathcal{U}_{\text{UL}}(\theta_2 \tau_1, \tau_2 \rightarrow \alpha) \\ \text{in } (\theta_3 \alpha, \theta_3 \circ \theta_2 \circ \theta_1) & \end{aligned}$$



Unification

- ▶ To combine (join) two given types we apply **unification**
- ▶ I.e., in case rule for applications, $\mathcal{U}_{UL}(\theta_2 \tau_1, \tau_2 \rightarrow \alpha)$
- ▶ Unification computes a substitution from two types:
 $\mathcal{U}_{UL} : \mathbf{Ty} \times \mathbf{Ty} \rightarrow \mathbf{TySubst}$
- ▶ If $\mathcal{U}_{UL}(\tau_1, \tau_2) = \theta$ then $\theta \tau_1 = \theta \tau_2$
 - ▶ And θ is the least such substitution
- ▶ Ex. $\mathcal{U}_{UL}(\alpha_1 \rightarrow Nat \rightarrow Bool, Nat \rightarrow Nat \rightarrow \alpha_2)$ equals θ with $\theta(\alpha_1) = Nat$ and $\theta(\alpha_2) = Bool$
- ▶ Note: unification is basically the \sqcup in the lattice of monotypes



Unification Algorithm

$$\mathcal{U}_{\text{UL}} (\text{Nat}, \text{Nat}) = \text{id}$$

$$\mathcal{U}_{\text{UL}} (\text{Bool}, \text{Bool}) = \text{id}$$

$$\mathcal{U}_{\text{UL}} (\tau_1 \rightarrow \tau_2, \tau_3 \rightarrow \tau_4) = \theta_2 \circ \theta_1$$

where

$$\theta_1 = \mathcal{U}_{\text{UL}} (\tau_1, \tau_3)$$

$$\theta_2 = \mathcal{U}_{\text{UL}} (\theta_1 \tau_2, \theta_1 \tau_4)$$

$$\mathcal{U}_{\text{UL}} (\alpha, \tau) = [\alpha \mapsto \tau] \text{ if } \text{chk} (\alpha, \tau)$$

$$\mathcal{U}_{\text{UL}} (\tau, \alpha) = [\alpha \mapsto \tau] \text{ if } \text{chk} (\alpha, \tau)$$

$$\mathcal{U}_{\text{UL}} (-, -) = \text{fail}$$

Here, $\text{chk} (\alpha, \tau)$ returns true if $\tau = \alpha$ or α is not a free variable in τ .



Inference algorithm: conditionals

$$\begin{aligned} & \mathcal{W}_{UL}(\Gamma, \text{if } t_1 \text{ then } t_2 \text{ else } t_3) = \\ & \text{let } (\tau_1, \theta_1) = \mathcal{W}_{UL}(\Gamma, t_1) \\ & \quad (\tau_2, \theta_2) = \mathcal{W}_{UL}(\theta_1 \Gamma, t_2) \\ & \quad (\tau_3, \theta_3) = \mathcal{W}_{UL}(\theta_2 (\theta_1 \Gamma), t_3) \\ & \quad \theta_4 = \mathcal{U}_{UL}(\theta_3 (\theta_2 \tau_1), \text{Bool}) \\ & \quad \theta_5 = \mathcal{U}_{UL}(\theta_4 (\theta_3 \tau_2), \theta_4 \tau_3) \\ & \text{in } (\theta_5 (\theta_4 \tau_3), \quad \theta_5 \circ \theta_4 \circ \theta_3 \circ \theta_2 \circ \theta_1) \end{aligned}$$

- ▶ Substitutions are applied as soon as possible.
- ▶ Error prone process of putting the right composition of substitutions everywhere.
- ▶ Substitutions are **idempotent**: blindly applying all of them all the time can only influence efficiency.



Inference algorithm: local definitions

$$\begin{aligned} \mathcal{W}_{\text{UL}}(\Gamma, \text{let } x = t_1 \text{ in } t_2) = \\ \text{let } (\tau_1, \theta_1) = \mathcal{W}_{\text{UL}}(\Gamma, t_1) \\ (\tau, \theta_2) = \mathcal{W}_{\text{UL}}((\theta_1 \Gamma)[x \mapsto \text{generalise}_{\text{UL}}(\theta_1 \Gamma, \tau_1)], t_2) \\ \text{in } (\tau, \theta_2 \circ \theta_1) \end{aligned}$$

$\text{generalise}_{\text{UL}}$ generalizes all variables absent in $\theta_1 \Gamma$ at once.



Inference algorithm: binary operators

$$\begin{aligned} & \mathcal{W}_{\text{UL}}(\Gamma, t_1 \oplus t_2) = \\ & \text{let } (\tau_1, \theta_1) = \mathcal{W}_{\text{UL}}(\Gamma, t_1) \\ & \quad (\tau_2, \theta_2) = \mathcal{W}_{\text{UL}}(\theta_1 \Gamma, t_2) \\ & \quad \theta_3 = \mathcal{U}_{\text{UL}}(\theta_2 \tau_1, \tau_{\oplus}^1) \\ & \quad \theta_4 = \mathcal{U}_{\text{UL}}(\theta_3 \tau_2, \tau_{\oplus}^2) \\ & \text{in } (\tau_{\oplus}, \theta_4 \circ \theta_3 \circ \theta_2 \circ \theta_1) \end{aligned}$$


Control-flow Analysis with Annotated Types



Control-flow analysis

Control-flow analysis (or closure analysis) determines:

For each function application, which functions may be applied.



Annotated types

$\varphi \in \mathbf{Ann}$ annotations

$\varphi ::= \emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$



Annotated types

φ	\in	\mathbf{Ann}	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types

φ	$::=$	\emptyset	$ $	$\{\pi\}$	$ $	$\varphi_1 \cup \varphi_2$		
$\hat{\tau}$	$::=$	α	$ $	Nat	$ $	$Bool$	$ $	$\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$



Annotated types

φ	\in	Ann	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types
$\hat{\sigma}$	\in	$\widehat{\mathbf{TyScheme}}$	annotated type schemes

φ	$::=$	\emptyset	$ $	$\{\pi\}$	$ $	$\varphi_1 \cup \varphi_2$		
$\hat{\tau}$	$::=$	α	$ $	<i>Nat</i>	$ $	<i>Bool</i>	$ $	$\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau}$	$ $	$\forall \alpha. \hat{\sigma}_1$				



Annotated types

φ	\in	Ann	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types
$\hat{\sigma}$	\in	$\widehat{\mathbf{TyScheme}}$	annotated type schemes
$\hat{\Gamma}$	\in	$\widehat{\mathbf{TyEnv}}$	annotated type environments

φ	$::=$	$\emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$
$\hat{\tau}$	$::=$	$\alpha \mid \mathit{Nat} \mid \mathit{Bool} \mid \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau} \mid \forall \alpha. \hat{\sigma}_1$
$\hat{\Gamma}$	$::=$	$[] \mid \hat{\Gamma}_1[x \mapsto \hat{\sigma}]$



Annotated types

φ	\in	Ann	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types
$\hat{\sigma}$	\in	$\widehat{\mathbf{TyScheme}}$	annotated type schemes
$\hat{\Gamma}$	\in	$\widehat{\mathbf{TyEnv}}$	annotated type environments

φ	$::=$	$\emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$
$\hat{\tau}$	$::=$	$\alpha \mid \mathit{Nat} \mid \mathit{Bool} \mid \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau} \mid \forall \alpha. \hat{\sigma}_1$
$\hat{\Gamma}$	$::=$	$[\] \mid \hat{\Gamma}_1[x \mapsto \hat{\sigma}]$

$\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\sigma}$ control-flow analysis



Control-flow analysis: constants

$$\frac{}{\widehat{\Gamma} \vdash_{\text{CFA}} n : \text{Nat}} \quad [\text{cfa-num}]$$



Control-flow analysis: constants

$$\frac{}{\widehat{\Gamma} \vdash_{\text{CFA}} n : \mathit{Nat}} \quad [\text{cfa-num}]$$

$$\frac{}{\widehat{\Gamma} \vdash_{\text{CFA}} \text{false} : \mathit{Bool}} \quad [\text{cfa-false}]$$

$$\frac{}{\widehat{\Gamma} \vdash_{\text{CFA}} \text{true} : \mathit{Bool}} \quad [\text{cfa-true}]$$



Control-flow analysis: variables

$$\frac{\hat{\Gamma}(x) = \hat{\sigma}}{\hat{\Gamma} \vdash_{\text{CFA}} x : \hat{\sigma}} \text{ [cfa-var]}$$



Control-flow analysis: functions

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_1] \vdash_{\text{CFA}} t_1 : \widehat{\tau}_2}{\widehat{\Gamma} \vdash_{\text{CFA}} \lambda_{\pi} x. t_1 : \widehat{\tau}_1 \xrightarrow{\{\pi\}} \widehat{\tau}_2} \text{ [cfa-lam]}$$



Control-flow analysis: functions

$$\frac{\hat{\Gamma}[x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2} \text{ [cfa-lam]}$$

$$\frac{\hat{\Gamma}[f \mapsto (\hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2)][x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \mu f. \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2} \text{ [cfa-mu]}$$



Control-flow analysis: functions

$$\frac{\hat{\Gamma}[x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2} \text{ [cfa-lam]}$$

$$\frac{\hat{\Gamma}[f \mapsto (\hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2)][x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \mu f. \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2} \text{ [cfa-mu]}$$

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t_1 : \hat{\tau}_2 \xrightarrow{\varphi} \hat{\tau} \quad \hat{\Gamma} \vdash_{\text{CFA}} t_2 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} t_1 t_2 : \hat{\tau}} \text{ [cfa-app]}$$

► φ describes what may be applied!



Control-flow analysis: conditionals

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t_1 : \textit{Bool} \quad \hat{\Gamma} \vdash_{\text{CFA}} t_2 : \hat{\tau} \quad \hat{\Gamma} \vdash_{\text{CFA}} t_3 : \hat{\tau}}{\hat{\Gamma} \vdash_{\text{CFA}} \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : \hat{\tau}} \quad [\textit{cfa-if}]$$



Control-flow analysis: local definitions

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t_1 : \hat{\sigma}_1 \quad \hat{\Gamma}[x \mapsto \hat{\sigma}_1] \vdash_{\text{CFA}} t_2 : \hat{\tau}}{\hat{\Gamma} \vdash_{\text{CFA}} \mathbf{let } x = t_1 \mathbf{ in } t_2 : \hat{\tau}} \quad [\text{cfa-let}]$$



Control-flow analysis: binary operators

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t_1 : \tau_{\oplus}^1 \quad \hat{\Gamma} \vdash_{\text{CFA}} t_2 : \tau_{\oplus}^2}{\hat{\Gamma} \vdash_{\text{CFA}} t_1 \oplus t_2 : \tau_{\oplus}} \quad [\text{cfa-op}]$$



Control-flow analysis: example

$$(\lambda_{\mathbf{F}}x. x) (\lambda_{\mathbf{G}}y. y)$$


Control-flow analysis: example

$$(\lambda_{\mathbf{F}}x. x) (\lambda_{\mathbf{G}}y. y)$$

$$\hat{\Gamma} \vdash_{\text{CFA}} (\lambda_{\mathbf{F}}x. x) (\lambda_{\mathbf{G}}y. y) : \forall \alpha. \alpha \xrightarrow{\{\mathbf{G}\}} \alpha$$


Control-flow analysis: example

$$(\lambda_F x. x) (\lambda_G y. y)$$

$$\begin{array}{c}
 \vdots \qquad \qquad \qquad \vdots \\
 \frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_G] \vdash_{\text{CFA}} x : \widehat{\tau}_G}{\widehat{\Gamma} \vdash_{\text{CFA}} \lambda_F x. x : \widehat{\tau}_G} \quad \frac{\widehat{\Gamma}[y \mapsto \alpha] \vdash_{\text{CFA}} y : \alpha}{\widehat{\Gamma} \vdash_{\text{CFA}} \lambda_G y. y : \widehat{\tau}_G} \\
 \frac{\widehat{\Gamma} \vdash_{\text{CFA}} \lambda_F x. x : \widehat{\tau}_G \xrightarrow{\{F\}} \widehat{\tau}_G \quad \widehat{\Gamma} \vdash_{\text{CFA}} \lambda_G y. y : \widehat{\tau}_G}{\widehat{\Gamma} \vdash_{\text{CFA}} (\lambda_F x. x) (\lambda_G y. y) : \widehat{\tau}_G} \\
 \frac{\widehat{\Gamma} \vdash_{\text{CFA}} (\lambda_F x. x) (\lambda_G y. y) : \widehat{\tau}_G}{\widehat{\Gamma} \vdash_{\text{CFA}} (\lambda_F x. x) (\lambda_G y. y) : \forall \alpha. \alpha \xrightarrow{\{G\}} \alpha}
 \end{array}$$

$$\widehat{\tau}_G = \alpha \xrightarrow{\{G\}} \alpha$$



Higher-order functions

```
let f = λFx. x + 1 in  
let g = λGy. y * 2 in  
let h = λHz. z 3 in  
h g + h f
```



Higher-order functions

```
let f = λFx. x + 1 in
let g = λGy. y * 2 in
let h = λHz. z 3 in
h g + h f
```

```
f : Nat  $\xrightarrow{\{F\}}$  Nat
g : Nat  $\xrightarrow{\{G\}}$  Nat
```



Higher-order functions

```
let f = λFx. x + 1 in
let g = λGy. y * 2 in
let h = λHz. z 3   in
h g + h f
```

```
f  :  Nat  $\xrightarrow{\{F\}}$  Nat
g  :  Nat  $\xrightarrow{\{G\}}$  Nat
h  :  (Nat  $\xrightarrow{??}$  Nat)  $\xrightarrow{\{H\}}$  Nat
```



Higher-order functions

```
let f = λFx. x + 1 in
let g = λGy. y * 2 in
let h = λHz. z 3   in
h g + h f
```

```
f  :  Nat  $\xrightarrow{\{F\}}$  Nat
g  :  Nat  $\xrightarrow{\{G\}}$  Nat
h  :  (Nat  $\xrightarrow{??}$  Nat)  $\xrightarrow{\{H\}}$  Nat
```

Should we have $h : (Nat \xrightarrow{\{F\}} Nat) \xrightarrow{\{H\}} Nat$ or
 $h : (Nat \xrightarrow{\{G\}} Nat) \xrightarrow{\{H\}} Nat$?



Conditionals

```
 $\lambda_{\mathbf{H}}z. \mathbf{if} \quad z \equiv 0$   
   $\mathbf{then} \lambda_{\mathbf{F}}x. x + 1$   
   $\mathbf{else} \lambda_{\mathbf{G}}y. y * 2$ 
```



Conditionals

```
 $\lambda_{\mathbf{H}}z. \mathbf{if} \quad z \equiv 0$   
 $\quad \mathbf{then} \lambda_{\mathbf{F}}x. x + 1$   
 $\quad \mathbf{else} \lambda_{\mathbf{G}}y. y * 2$ 
```

Should we have $\mathit{Nat} \xrightarrow{\{\mathbf{H}\}} (\mathit{Nat} \xrightarrow{\{\mathbf{F}\}} \mathit{Nat})$ or
 $\mathit{Nat} \xrightarrow{\{\mathbf{H}\}} (\mathit{Nat} \xrightarrow{\{\mathbf{G}\}} \mathit{Nat})$?



Subeffecting

$$\frac{\widehat{\Gamma}[x \mapsto \widehat{\tau}_1] \vdash_{\text{CFA}} t_1 : \widehat{\tau}_2}{\widehat{\Gamma} \vdash_{\text{CFA}} \lambda_{\pi} x. t_1 : \widehat{\tau}_1 \xrightarrow{\{\pi\} \cup \varphi} \widehat{\tau}_2} \text{ [cfa-lam]}$$



Subeffecting

$$\frac{\hat{\Gamma}[x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\} \cup \varphi} \hat{\tau}_2} \text{ [cfa-lam]}$$

$$\frac{\hat{\Gamma}[f \mapsto (\hat{\tau}_1 \xrightarrow{\{\pi\} \cup \varphi} \hat{\tau}_2)][x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \mu f. \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\} \cup \varphi} \hat{\tau}_2} \text{ [cfa-mu]}$$



Subeffecting: example

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z 3$  in  
 $h\ g + h\ f$ 
```

```
 $f : \text{Nat} \xrightarrow{\{\mathbf{F}, \mathbf{G}\}} \text{Nat}$   
 $g : \text{Nat} \xrightarrow{\{\mathbf{F}, \mathbf{G}\}} \text{Nat}$   
 $h : (\text{Nat} \xrightarrow{\{\mathbf{F}, \mathbf{G}\}} \text{Nat}) \xrightarrow{\{\mathbf{H}\}} \text{Nat}$ 
```



Subeffecting: example

$$\lambda_{\mathbf{H}}z. \text{if } z \equiv 0 \\ \text{then } \lambda_{\mathbf{F}}x. x + 1 \\ \text{else } \lambda_{\mathbf{G}}y. y * 2$$
$$\mathit{Nat} \xrightarrow{\{\mathbf{H}\}} (\mathit{Nat} \xrightarrow{\{\mathbf{F},\mathbf{G}\}} \mathit{Nat})$$


Inference algorithm: simple types

β	\in	$\widehat{\text{AnnVar}}$	annotation variables
$\hat{\tau}$	\in	$\widehat{\text{SimpleTy}}$	simple types
$\hat{\sigma}$	\in	$\widehat{\text{SimpleTyScheme}}$	simple type schemes
$\hat{\Gamma}$	\in	$\widehat{\text{SimpleTyEnv}}$	simple type environments
$\hat{\theta}$	\in	$\widehat{\text{TySubst}}$	hybrid type substitution
C	\in	$\widehat{\text{Constr}}$	constraint

$\hat{\tau}$	$::=$	$\alpha \mid \text{Nat} \mid \text{Bool} \mid \hat{\tau}_1 \xrightarrow{\beta} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau} \mid \forall \alpha. \hat{\sigma}_1$
$\hat{\Gamma}$	$::=$	$[] \mid \hat{\Gamma}_1[x \mapsto \hat{\sigma}]$
C	$::=$	$\emptyset \mid \{\beta \supseteq \varphi\} \mid C_1 \cup C_2$



Inference algorithm

$$\begin{aligned} \mathit{generalise}_{\text{CFA}} & : \widehat{\text{SimpleTyEnv}} \times \widehat{\text{SimpleTy}} \rightarrow \widehat{\text{SimpleTyScheme}} \\ \mathit{instantiate}_{\text{CFA}} & : \widehat{\text{SimpleTyScheme}} \rightarrow \widehat{\text{SimpleTy}} \\ \mathcal{U}_{\text{CFA}} & : \widehat{\text{SimpleTy}} \times \widehat{\text{SimpleTy}} \rightarrow \widehat{\text{TySubst}} \\ \mathcal{W}_{\text{CFA}} & : \widehat{\text{SimpleTyEnv}} \times \text{Tm} \rightarrow \widehat{\text{SimpleTy}} \times \widehat{\text{TySubst}} \times \text{Constr} \end{aligned}$$


Inference algorithm: constants

$$\mathcal{W}_{\text{CFA}}(\hat{\Gamma}, n) = (\text{Nat}, \text{id}, \emptyset)$$

$$\mathcal{W}_{\text{CFA}}(\hat{\Gamma}, \text{false}) = (\text{Bool}, \text{id}, \emptyset)$$

$$\mathcal{W}_{\text{CFA}}(\hat{\Gamma}, \text{true}) = (\text{Bool}, \text{id}, \emptyset)$$



Inference algorithm: variables

$$\mathcal{W}_{\text{CFA}}(\hat{\Gamma}, x) = (\text{instantiate}_{\text{CFA}}(\hat{\Gamma}(x)), \text{id}, \emptyset)$$



Inference algorithm: functions

$$\begin{aligned} \mathcal{W}_{\text{CFA}}(\widehat{\Gamma}, \lambda_{\pi} x. t_1) = & \text{let } \alpha_1 \text{ be fresh} \\ & (\widehat{\tau}_2, \widehat{\theta}, C_1) = \mathcal{W}_{\text{CFA}}(\widehat{\Gamma}[x \mapsto \alpha_1], t_1) \\ & \beta \text{ be fresh} \\ \text{in } ((\widehat{\theta} \alpha_1) \xrightarrow{\beta} \widehat{\tau}_2, & \widehat{\theta}, C_1 \cup \{\beta \supseteq \{\pi\}\}) \end{aligned}$$

- ▶ Introduce fresh variables for annotations.
- ▶ Invariant: only variables as annotations in types (aka simple types).
- ▶ Put concrete information about the variables into C .
- ▶ Solve constraints later to obtain actual sets.
- ▶ Simplifies unification substantially.



Changes to unification

Only the case for function changes:

$$\dots$$
$$\mathcal{U}_{\text{UL}} (\tau_1 \xrightarrow{\beta_1} \tau_2, \tau_3 \xrightarrow{\beta_2} \tau_4) = \theta_2 \circ \theta_1 \circ \theta_0$$

where

$$\theta_0 = [\beta_1 \mapsto \beta_2]$$

$$\theta_1 = \mathcal{U}_{\text{UL}} (\theta_0 \tau_1, \theta_0 \tau_3)$$

$$\theta_2 = \mathcal{U}_{\text{UL}} (\theta_1 (\theta_0 \tau_2), \theta_1 (\theta_0 \tau_4))$$

...

No need to recurse on annotations: just map one variable to the other.



Inference algorithm: recursive functions

$$\begin{aligned} \mathcal{W}_{\text{CFA}}(\widehat{\Gamma}, \mu f. \lambda_{\pi} x. t_1) = \\ \text{let } \alpha_1, \alpha_2, \beta \text{ be fresh} \\ (\widehat{\tau}_2, \widehat{\theta}_1, C_1) = \mathcal{W}_{\text{CFA}}(\widehat{\Gamma}[f \mapsto (\alpha_1 \xrightarrow{\beta} \alpha_2)] [x \mapsto \alpha_1], t_1) \\ \widehat{\theta}_2 = \mathcal{U}_{\text{CFA}}(\widehat{\tau}_2, \widehat{\theta}_1 \alpha_2) \\ \text{in } (\widehat{\theta}_2 (\widehat{\theta}_1 \alpha_1) \xrightarrow{\widehat{\theta}_2 (\widehat{\theta}_1 \beta)} \widehat{\theta}_2 \widehat{\tau}_2, \widehat{\theta}_2 \circ \widehat{\theta}_1, \\ (\widehat{\theta}_2 C_1) \cup \{\widehat{\theta}_2 (\widehat{\theta}_1 \beta) \supseteq \{\pi\}\}) \end{aligned}$$

Remember: $\widehat{\theta}_1$ and $\widehat{\theta}_2$ can only rename annotation variables.



Constraints: example

```
let f = λFx. x + 1 in
let g = λGy. y * 2 in
let h = λHz. z 3 in
h g + h f
```



Constraints: example

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z \ 3$  in  
 $h \ g + h \ f$ 
```

```
 $f$  :  $\mathit{Nat} \xrightarrow{\beta_1} \mathit{Nat}$   
 $g$  :  $\mathit{Nat} \xrightarrow{\beta_2} \mathit{Nat}$   
 $h$  :  $(\mathit{Nat} \xrightarrow{\beta_3} \mathit{Nat}) \xrightarrow{\{\mathbf{H}\}} \mathit{Nat}$ 
```



Constraints: example

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z \ 3$  in  
 $h \ g + h \ f$ 
```

$f : \text{Nat} \xrightarrow{\beta_1} \text{Nat}$

$g : \text{Nat} \xrightarrow{\beta_2} \text{Nat}$

$h : (\text{Nat} \xrightarrow{\beta_3} \text{Nat}) \xrightarrow{\{\mathbf{H}\}} \text{Nat}$

$\widehat{\theta}(\beta_1) = \beta_3$

$\widehat{\theta}(\beta_2) = \beta_3$



Constraints: example

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z \ 3$  in  
 $h \ g + h \ f$ 
```

$f : \text{Nat} \xrightarrow{\beta_1} \text{Nat}$
 $g : \text{Nat} \xrightarrow{\beta_2} \text{Nat}$
 $h : (\text{Nat} \xrightarrow{\beta_3} \text{Nat}) \xrightarrow{\{\mathbf{H}\}} \text{Nat}$

$$\widehat{\theta}(\beta_1) = \beta_3$$

$$\widehat{\theta}(\beta_2) = \beta_3$$

$$C = \{\beta_1 \supseteq \{\mathbf{F}\}, \beta_2 \supseteq \{\mathbf{G}\}\}$$



Constraints: example

```
let f = λFx. x + 1 in
let g = λGy. y * 2 in
let h = λHz. z 3 in
h g + h f
```

$$f : \text{Nat} \xrightarrow{\beta_1} \text{Nat}$$

$$g : \text{Nat} \xrightarrow{\beta_2} \text{Nat}$$

$$h : (\text{Nat} \xrightarrow{\beta_3} \text{Nat}) \xrightarrow{\{H\}} \text{Nat}$$

$$\hat{\theta}(\beta_1) = \beta_3$$

$$\hat{\theta}(\beta_2) = \beta_3$$

$$C = \{\beta_1 \supseteq \{F\}, \beta_2 \supseteq \{G\}\}$$

$$\hat{\theta}C = \{\beta_3 \supseteq \{F\}, \beta_3 \supseteq \{G\}\}$$



Constraints: example

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z \ 3$  in  
 $h \ g + h \ f$ 
```

$$f : \text{Nat} \xrightarrow{\beta_1} \text{Nat}$$

$$g : \text{Nat} \xrightarrow{\beta_2} \text{Nat}$$

$$h : (\text{Nat} \xrightarrow{\beta_3} \text{Nat}) \xrightarrow{\{\mathbf{H}\}} \text{Nat}$$

$$\widehat{\theta}(\beta_1) = \beta_3$$

$$\widehat{\theta}(\beta_2) = \beta_3$$

$$C = \{\beta_1 \supseteq \{\mathbf{F}\}, \beta_2 \supseteq \{\mathbf{G}\}\}$$

$$\widehat{\theta} C = \{\beta_3 \supseteq \{\mathbf{F}\}, \beta_3 \supseteq \{\mathbf{G}\}\}$$

Least solution: $\beta_3 = \{\mathbf{F}, \mathbf{G}\}$.



Poisoning

Naive use of subeffecting is fatal for the precision of your analysis:

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. \mathbf{if } z \equiv 0 \mathbf{ then } f \mathbf{ else } g$  in  
 $f$ 
```

$$Nat \xrightarrow{\{\mathbf{F}, \mathbf{G}\}} Nat$$


Separate rule for subeffecting

$$\frac{\widehat{\Gamma} \vdash_{\text{CFA}} t : \widehat{\tau}_1 \xrightarrow{\varphi} \widehat{\tau}_2}{\widehat{\Gamma} \vdash_{\text{CFA}} t : \widehat{\tau}_1 \xrightarrow{\varphi \cup \varphi'} \widehat{\tau}_2} \text{ [cfa-sub]}$$



Separate rule for subeffecting

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\tau}_1 \xrightarrow{\varphi \cup \varphi'} \hat{\tau}_2} \text{ [cfa-sub]}$$

We can remove the subeffecting from the lambda rule:

$$\frac{\hat{\Gamma}[x \mapsto \hat{\tau}_1] \vdash_{\text{CFA}} t_1 : \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} \lambda_{\pi} x. t_1 : \hat{\tau}_1 \xrightarrow{\{\pi\}} \hat{\tau}_2} \text{ [cfa-lam]}$$



Separate compilation?

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z 3$  in  
 $h\ g + h\ f$ 
```

```
 $f : \text{Nat} \xrightarrow{\{\mathbf{F}\}} \text{Nat}$   
 $g : \text{Nat} \xrightarrow{\{\mathbf{G}\}} \text{Nat}$   
 $h : (\text{Nat} \xrightarrow{\{\mathbf{F},\mathbf{G}\}} \text{Nat}) \xrightarrow{\{\mathbf{H}\}} \text{Nat}$ 
```



Separate compilation?

```
let  $f = \lambda_{\mathbf{F}}x. x + 1$  in  
let  $g = \lambda_{\mathbf{G}}y. y * 2$  in  
let  $h = \lambda_{\mathbf{H}}z. z 3$  in  
 $h\ g + h\ f$ 
```

```
 $f : \text{Nat} \xrightarrow{\{\mathbf{F}\}} \text{Nat}$   
 $g : \text{Nat} \xrightarrow{\{\mathbf{G}\}} \text{Nat}$   
 $h : (\text{Nat} \xrightarrow{\{\mathbf{F},\mathbf{G}\}} \text{Nat}) \xrightarrow{\{\mathbf{H}\}} \text{Nat}$ 
```

- ☞ We need to analyse the whole program to accurately determine the domain of h .



Subeffecting and subtyping

- ▶ We have now seen subeffecting at work.
- ▶ The main ideas of all of these are:
 - ▶ compute types and annotations independent of context,
 - ▶ allow to weaken the outcomes whenever convenient.
- ▶ Weakening provides a form of context-sensitiveness.
- ▶ In (shape conformant) subtyping we may also weaken annotations deeper in the type.



Polyvariance



Example: parity analysis

- ▶ The natural number 1 can be analysed to have type $Nat\{O\}$.
- ▶ A function like *double* on naturals should work for all naturals: $Nat\{O,E\} \rightarrow Nat\{E\}$.
- ▶ The type of 1 can then be weakened to $Nat\{O,E\}$ as it is passed into *double*, without influencing the type and other uses of 1.

```
let one = 1 in
let double = λGy. y * 2 in
one * double one
```



Limitations to subeffecting and subtyping

- ▶ Weakening prevents certain forms of poisoning,
- ▶ but it does not help propagate analysis information.
- ▶ For *id* on naturals we expect the type $\text{Nat}^{O,E} \rightarrow \text{Nat}^{O,E}$.
- ▶ However, we also know that *O* inputs leads to *O* outputs, and similar for *E*.
- ▶ Our annotated types cannot represent this information.
- ▶ Is it acceptable that *id* 1 and 1 give different analyses?



Polyvariance

- ▶ We consider only let-polyvariance.
- ▶ Exactly analogous to let-polymorphism, but for annotations.
- ▶ For *id* we then derive the type $\forall\beta. \text{Nat}^\beta \rightarrow \text{Nat}^\beta$.
- ▶ For *id* 1 we can choose $\beta = \{O\}$ so that *id* 1 has annotation $\{O\}$.
- ▶ Allows us to propagate properties through functions that are property-agnostic.
- ▶ Polyvariant analyses with subtyping are current state of the art.
- ▶ But it depends somewhat on the analysis.



Annotated polyvariant types

$\varphi \in \mathbf{Ann}$ annotations

$\varphi ::= \beta \mid \emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$



Annotated polyvariant types

φ	\in	\mathbf{Ann}	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types

φ	$::=$	β	$ $	\emptyset	$ $	$\{\pi\}$	$ $	$\varphi_1 \cup \varphi_2$
$\hat{\tau}$	$::=$	α	$ $	<i>Nat</i>	$ $	<i>Bool</i>	$ $	$\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$



Annotated polyvariant types

φ	\in	Ann	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types
$\hat{\sigma}$	\in	$\widehat{\mathbf{TyScheme}}$	annotated type schemes

φ	$::=$	$\beta \mid \emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$
$\hat{\tau}$	$::=$	$\alpha \mid Nat \mid Bool \mid \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau} \mid \forall \alpha. \hat{\sigma}_1 \mid \forall \beta. \hat{\sigma}_1$



Annotated polyvariant types

φ	\in	Ann	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types
$\hat{\sigma}$	\in	$\widehat{\mathbf{TyScheme}}$	annotated type schemes
$\hat{\Gamma}$	\in	$\widehat{\mathbf{TyEnv}}$	annotated type environments

φ	$::=$	$\beta \mid \emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$
$\hat{\tau}$	$::=$	$\alpha \mid \mathit{Nat} \mid \mathit{Bool} \mid \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau} \mid \forall \alpha. \hat{\sigma}_1 \mid \forall \beta. \hat{\sigma}_1$
$\hat{\Gamma}$	$::=$	$[] \mid \hat{\Gamma}_1[x \mapsto \hat{\sigma}]$



Annotated polyvariant types

φ	\in	Ann	annotations
$\hat{\tau}$	\in	$\widehat{\mathbf{Ty}}$	annotated types
$\hat{\sigma}$	\in	$\widehat{\mathbf{TyScheme}}$	annotated type schemes
$\hat{\Gamma}$	\in	$\widehat{\mathbf{TyEnv}}$	annotated type environments

φ	$::=$	$\beta \mid \emptyset \mid \{\pi\} \mid \varphi_1 \cup \varphi_2$
$\hat{\tau}$	$::=$	$\alpha \mid \mathit{Nat} \mid \mathit{Bool} \mid \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2$
$\hat{\sigma}$	$::=$	$\hat{\tau} \mid \forall \alpha. \hat{\sigma}_1 \mid \forall \beta. \hat{\sigma}_1$
$\hat{\Gamma}$	$::=$	$[] \mid \hat{\Gamma}_1[x \mapsto \hat{\sigma}]$

$\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\sigma}$ control-flow analysis



Is this enough?

```
let f = λFx. True in
let g = λGk. if f 0 then k else (λHy. False) in
g f
```

A (mono)type for $g f$ is $v1 \xrightarrow{\{F\} \cup \{H\}} Bool$.

$\{H\}$ is contributed by the else-part, $\{F\}$ comes from the parameter passed to g .

But what is the type of g that can lead to such type?



Is this enough?

```
let f = λFx. True in
let g = λGk. if 0 then k else (λHy. False) in
g f
```

A (mono)type for $g f$ is $v1 \xrightarrow{\{F\} \cup \{H\}} Bool$.

$\{H\}$ is contributed by the else-part, $\{F\}$ comes from the parameter passed to g .

But what is the type of g that can lead to such type?

$$g : \forall a. \forall \beta. (a \xrightarrow{\beta} Bool) \xrightarrow{G} (a \xrightarrow{\beta \cup \{H\}} Bool)$$

But how can we manipulate such annotations correctly?

👉 Add a few rules



Polyvariant type system: generalisation

Introduction for type variables:

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\sigma} \quad \alpha \notin \text{ftv}(\Gamma)}{\hat{\Gamma} \vdash_{\text{CFA}} t : \forall \alpha. \hat{\sigma}} \quad [\text{cfa-gen}]$$

Introduction for annotation variables:

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\sigma} \quad \beta \notin \text{fav}(\Gamma)}{\hat{\Gamma} \vdash_{\text{CFA}} t : \forall \beta. \hat{\sigma}} \quad [\text{cfa-ann-gen}]$$

Here $\text{fav}(\Gamma)$ computes the free annotation variables in Γ .



Polyvariant type system: instantiation

Elimination for type variables:

$$\frac{\widehat{\Gamma} \vdash_{\text{CFA}} t : \forall \alpha. \widehat{\sigma}}{\widehat{\Gamma} \vdash_{\text{CFA}} t : [\alpha \mapsto \widehat{\tau}] \widehat{\sigma}} \quad [\text{cfa-inst}]$$

Elimination for annotation variables:

$$\frac{\widehat{\Gamma} \vdash_{\text{CFA}} t : \forall \beta. \widehat{\sigma}}{\widehat{\Gamma} \vdash_{\text{CFA}} t : [\beta \mapsto \varphi] \widehat{\sigma}} \quad [\text{cfa-ann-inst}]$$



Polyvariant type system: subeffecting again

To align the types of the then-part and else-part, and to match arguments to function types, we still need subeffecting.

Recap:

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2}{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\tau}_1 \xrightarrow{\varphi \cup \varphi'} \hat{\tau}_2} \text{ [cfa-sub]}$$

then-part: β can be weakened to $\beta \cup \{\mathbf{H}\}$.

else-part: $\{\mathbf{H}\}$ can be weakened to $\{\mathbf{H}\} \cup \beta$.

But these are **not** the same!



When are two annotations equal?

The type system has no way of knowing, so we have to tell it when.

$$\frac{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 \quad \varphi \equiv \varphi'}{\hat{\Gamma} \vdash_{\text{CFA}} t : \hat{\tau}_1 \xrightarrow{\varphi'} \hat{\tau}_1} \text{ [cfa-eq]}$$

In other words: you may replace equals by equals.

☞ $\{\mathbf{H}\} \cup \beta$ by $\beta \cup \{\mathbf{H}\}$

Problem now becomes to define/axiomatize equality for these annotations.



Equality of annotations axiomatized (1)

$$\frac{}{\varphi \equiv \varphi} [q-refl]$$

$$\frac{\varphi' \equiv \varphi}{\varphi \equiv \varphi'} [q-symm]$$

$$\frac{\varphi \equiv \varphi'' \quad \varphi'' \equiv \varphi'}{\varphi \equiv \varphi'} [q-trans]$$

$$\frac{\varphi_1 \equiv \varphi'_1 \quad \varphi_2 \equiv \varphi'_2}{\varphi_1 \cup \varphi_2 \equiv \varphi'_1 \cup \varphi'_2} [q-join]$$



Equality of annotations axiomatized (2)

$$\frac{}{\{\} \cup \varphi \equiv \varphi} \text{ [q-unit]}$$

$$\frac{}{\varphi \cup \varphi \equiv \varphi} \text{ [q-idem]}$$

$$\frac{}{\varphi_1 \cup \varphi_2 \equiv \varphi_2 \cup \varphi_1} \text{ [q-comm]}$$

$$\frac{}{\varphi_1 \cup (\varphi_2 \cup \varphi_3) \equiv (\varphi_1 \cup \varphi_2) \cup \varphi_3} \text{ [q-ass]}$$



This combination of axioms often occurs:


- ▶ Unit
- ▶ Commutativity
- ▶ Associativity
- ▶ Idempotency

👉 Modulo UCAI



What about the algorithm?

- ▶ We still perform generalization in the let.
- ▶ And instantiation in the variable case.
- ▶ Recall:
 - ▶ The algorithm unifies types and identifies annotation variables.
 - ▶ It collects constraints on the latter.
- ▶ After algorithm \mathcal{W}_{CFA} , we solve the constraints to obtain annotation variables.
- ▶ In the monovariant setting this was fine: correctness did not depend on the context.
- ▶ In a polyvariant setting, the context plays a role

 Constraints on annotations must be propagated along.



Some variations

- ▶ Idea 1: simply store all constraints in the type.
 - ▶ During instantiation refresh type and annotations variables in the type, and the constraint set (consistently).
 - ▶ Includes also trivial and irrelevant constraints.
 - ▶ Some say: simple duplication is not feasible.
- ▶ Idea 2: simplify constraints as much as possible before storing them.
 - ▶ Simplification can take many forms.
 - ▶ Takes place as part of generalisation.
 - ▶ Type schemes store constraints sets: rather like qualified types.



Simplification

- ▶ Simplification = intermediate constraint solving.
- ▶ In both cases, annotations left unconstrained can be defaulted to the best possible.
- ▶ However, annotation variables that occur in the type to be generalized must be left unharmed.
- ▶ Why? Annotation variables provide flexibility for propagation.
 - ☞ Defaulting throws that flexibility away.



Example (to illustrate)

- ▶ Assume \mathcal{W}_{CFA} returns type $(v1 \xrightarrow{\beta_1} v1) \xrightarrow{\beta_2} (v1 \xrightarrow{\beta_3} v1)$ and constraint set $\{\beta_2 \supseteq \{\mathbf{G}\}, \beta_3 \supseteq \beta_4, \beta_4 \supseteq \beta_1, \beta_5 \supseteq \{\mathbf{H}\}, \beta_3 \supseteq \beta\}$
- ▶ And that β occurs free in $\hat{\Gamma}$.
- ▶ β_5 is not relevant, so it can be omitted (set to $\{\mathbf{H}\}$).
 - ▶ It does not occur in the type, or the context
- ▶ β_4 is not relevant either, but removing it implies we must add $\beta_3 \supseteq \beta_1$.
- ▶ Neither $\beta_2 \supseteq \{\mathbf{G}\}$ and $\beta_3 \supseteq \beta$ may be touched.
- ▶ Remember the invariant to keep unification simple: only annotation variables in types.



Constrained types and type schemes

Introduce an additional layer of types (a la qualified types):

$$\begin{aligned}\hat{\tau} &::= \alpha \mid \mathit{Nat} \mid \mathit{Bool} \mid \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 \\ \hat{\rho} &::= \hat{\tau} \mid c \Rightarrow \hat{\rho} \\ \hat{\sigma} &::= \hat{\rho} \mid \forall \alpha. \hat{\sigma}_1 \mid \forall \beta. \hat{\sigma}_1\end{aligned}$$



Generalisation and instantiation

- ▶ Instantiation provides fresh variables for universally quantified variables.
- ▶ Generalisation invokes the simplifier.
- ▶ Simplification can be performed by a worklist algorithm, that leaves certain (which?) variables untouched.
 - ☞ Considers them to be constants
- ▶ Type signature compartmentalizes a local definition: we do not care what happens inside.

