

# Type Reconstruction

Klaus Ostermann  
Uni Marburg

Based on lecture notes by Andrew Myers

# Type Reconstruction

- A.k.a. type inference
- Advantage of type annotations:
  - Type checker is simple
- Disadvantage:
  - Writing type annotations all the time can be tiresome, since many annotations are obvious

# Places where we required type annotations

$$\frac{\Gamma, x:\tau \vdash e : \tau'}{\Gamma \vdash \lambda x:\tau. e : \tau \rightarrow \tau'}$$

$$\frac{\Gamma, y:\tau \rightarrow \tau', x:\tau \vdash e : \tau'}{\Gamma \vdash \mathbf{rec} \ y:\tau \rightarrow \tau'. \lambda x. e : \tau \rightarrow \tau'} \quad \frac{\Gamma \vdash e : \tau \quad \Gamma, x:\tau \vdash e' : \tau'}{\Gamma \vdash \mathbf{let} \ x:\tau = e \mathbf{in} \ e' : \tau'}$$

# Typing rules are easy to specify w/o type annotations

$$\frac{\Gamma, x:\tau \vdash e : \tau'}{\Gamma \vdash \lambda x. e : \tau \rightarrow \tau'}$$

$$\frac{\Gamma, y:\tau \rightarrow \tau', x:\tau \vdash e : \tau'}{\Gamma \vdash \mathbf{rec} \ y. \lambda x. e : \tau \rightarrow \tau'}$$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma, x:\tau \vdash e' : \tau'}{\Gamma \vdash \mathbf{let} \ x = e \ \mathbf{in} \ e' : \tau'}$$

- But how can they be implemented?
  - Problem: Need to guess  $\tau$

# Inferring types by hand...

```
let d =  $\lambda z. z + z$  in  
  ( $\lambda f. \lambda x. \lambda y.$   
    if (f x y) then  
      f (d x) y  
    else  
      f x (f x y))
```

Since + is an operation on ints,  $z: \text{Int}$

Hence  $d: \text{Int} \rightarrow \text{Int}$

Hence  $x: \text{Int}$

Hence  $f: \text{Int} \rightarrow ??? \rightarrow ???$

Since (f x y) is used as condition  
 $f: \text{Int} \rightarrow ??? \rightarrow \text{Bool}$

Hence (see else branch)  
 $f: \text{Int} \rightarrow \text{Bool} \rightarrow \text{Bool}$  and  $y: \text{Bool}$

Hence the type of the expression is  
 $(\text{Int} \rightarrow \text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Int} \rightarrow \text{Bool} \rightarrow \text{Bool}$

# Type equality constraints

$$\begin{array}{c}
 \frac{f:T2, x:T5 \vdash f : \text{int} \rightarrow T6 \quad f:T2, x:T5 \vdash 1 : \text{int}}{\quad} \\
 \frac{f:T2, x:T5 \vdash f \ 1 : T6}{\quad} \quad \frac{\quad}{y:T3 \vdash y : T4} (T3=T4) \\
 \frac{f:T2 \vdash \lambda x. f \ 1 : T1 \text{ } (=T5 \rightarrow T6)}{\quad} \quad \frac{\quad}{\vdash (\lambda y. y) : T2} (T2=T3 \rightarrow T4) \\
 \hline
 \vdash \lambda f. \lambda x. f \ 1 : T2 \rightarrow T1 \quad \vdash (\lambda y. y) : T2 \text{ } (T2=T3 \rightarrow T4) \\
 \hline
 \vdash (\lambda f. \lambda x. (f \ 1)) (\lambda y. y) : T1
 \end{array}$$

$$T2 = T3 \rightarrow T4, T3 = T4, T1 = T5 \rightarrow T6, T2 = \text{int} \rightarrow T6$$

# Type equality constraints

$$\overline{\Gamma, x:\tau \vdash x:\tau}$$

$$\overline{\Gamma \vdash b:B}$$

$$\frac{\Gamma \vdash e_0:\tau_0 \quad \Gamma \vdash e_1:\tau_1 \quad \tau_0 = \tau_1 \rightarrow T}{\Gamma \vdash e_0 \ e_1 : T}$$

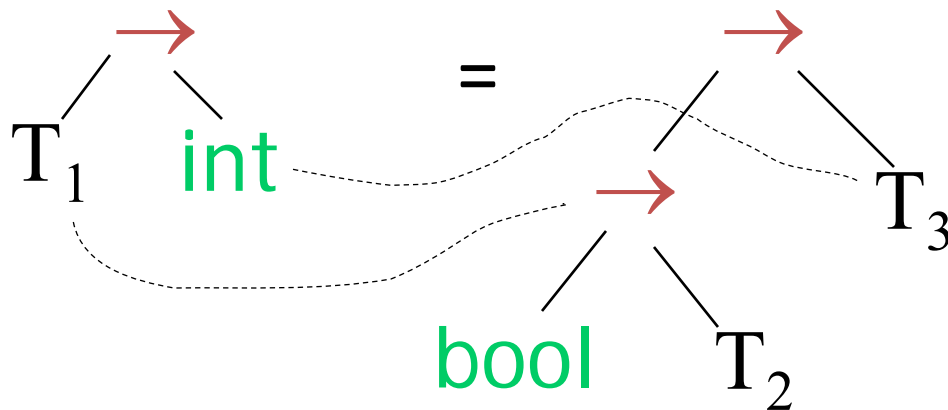
$$\frac{\Gamma, x:T \vdash e:\tau'}{\Gamma \vdash \lambda x. e : T \rightarrow \tau'}$$

$$\frac{\Gamma \vdash e_1:\tau_1 \quad \Gamma, x:\tau_1 \vdash e_2:\tau_2}{\Gamma \vdash \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2}$$

$$\frac{\Gamma, x:T_1, y:T_1 \rightarrow T_2 \vdash e:\tau' \quad \tau' = T_2}{\Gamma \vdash \mathbf{rec} \ y. \lambda x. e : T_1 \rightarrow T_2}$$

# Unification

- How to solve equations?
- Idea: given equation  $\tau_1 = \tau_2$ , *unify* type expressions to solve for variables in both
- Example:  $T_1 \rightarrow \text{int} = (\text{bool} \rightarrow T_2) \rightarrow T_3$
- Result: *substitution*  $T_1 := \text{bool} \rightarrow T_2, T_3 \mapsto \text{int}$





# Substitution and Unifier

- A substitution  $S$  maps type variables  $T$  to types  $\tau$
- We write  $S(\tau)$  for the result of applying all substitutions in  $S$  on  $\tau$
- A substitution  $S$  is a unifier of  $\tau$  and  $\tau'$ , if  $S(\tau) = S(\tau')$
- To preserve maximal polymorphism we want the *weakest* unifier
- $S_1$  is weaker than  $S_2$ , if there is a non-trivial substitution  $S_3$  such that  $S_2 = S_3 \cdot S_1$  whereby  $S_3 \cdot S_1(\tau) = S_3(S_1(\tau))$

# Robinson's Algorithm

$$\text{unify}(\emptyset) = \emptyset$$

$$\text{unify}(\{B = B\} \cup E) = \text{unify}(E)$$

$$\text{unify}(\{B_1 = B_2\} \cup E) = \text{error} \quad (\text{if } B_1 \neq B_2)$$

$$\text{unify}(\{T = T\} \cup E) = \text{unify}(E) \quad (\text{if } T \text{ is not mentioned in } \tau)$$

$$\text{unify}(\{T = \tau\} \cup E) = \text{unify}(\{\tau = T\} \cup E) = \text{unify}(E\{\tau/T\}) \circ \{T \mapsto \tau\}$$

$$\text{unify}(\{\tau_1 \rightarrow \tau_2 = \tau'_1 \rightarrow \tau'_2\} \cup E) = \text{unify}(\{\tau_1 = \tau'_1, \tau_2 = \tau'_2\} \cup E)$$

$$\text{unify}(\{\tau = \tau'\} \cup E) = \text{error} \quad (\text{if no previous rules match})$$

Often called "Occurs Check"



This definition is well-founded, but this is not obvious.

Either the number of variables in the equations becomes smaller, or it stays equal.

In the latter case, the total size of the equations or the number of arrows becomes smaller.

# Complexity of Unification

```
let b = true in
let f0 = λx. x+1 in
let f1 = λx. if b then f0 else λy.x y in
let f2 = λx. if b then f1 else λy.x y in
⋮
let fn = λx. if b then fn-1 else λy.x y in
0
```

If  $f_n : \tau_n$ , then  $\tau_0 = \text{Int} \rightarrow \text{Int}$  and  $\tau_{n+1} = \tau_n \rightarrow \tau_n$

Hence type inference can take exponential time

With a better representations of types (DAGs instead of trees) the complexity can be improved to approximately  $O(n^2)$

# Curry-Howard Isomorphism

- There is a deep connection between type systems and (intuitionistic/constructive) logic
- A proof of a proposition in constructive logic is a construction of an object that witnesses the proposition
- The Curry-Howard isomorphism says that proofs are the same as terms/programs

# Constructive vs classical proofs

- Not every proof in classical logic is also valid in intuitionistic logic

**Theorem** There exist irrational numbers  $a$  and  $b$  such that  $a^b$  is rational.

*Proof.* Either  $\sqrt{2}^{\sqrt{2}}$  is rational or not. If it is, take  $a = b = \sqrt{2}$  and we are done. If it is not, take  $a = \sqrt{2}^{\sqrt{2}}$  and  $b = \sqrt{2}$ ; then  $a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2$ , and again we are done.  $\square$

- Law of excluded middle is not valid in intuitionistic logic: It is not constructive!

# Intuitionistic logic

- Syntax of formulas:

$$\phi ::= \top \mid \perp \mid P \mid \phi_1 \Rightarrow \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg \phi.$$

- With second-order quantification:

$$\phi ::= \dots \mid \forall P. \phi.$$

# Natural Deduction

- Calculus developed by Gentzen to define proof rules of a logic
- Operators (so-called connectives) typically have introduction and elimination rules
- We will see that the deduction rules in natural deduction style correspond exactly to the typing rules of System F with sums and products
  - Terms are a linear notation of proofs!

# Proof- and Typing Rules Side-by-Side

*intuitionistic logic*

$\lambda^{\rightarrow}$  or System F type system

(axiom)  $\Gamma, \phi \vdash \phi$

$\Gamma, x : \tau \vdash x : \tau$

( $\rightarrow$ -intro) 
$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \Rightarrow \psi}$$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\lambda x : \sigma. e) : \sigma \rightarrow \tau}$$

( $\rightarrow$ -elim) 
$$\frac{\Gamma \vdash \phi_1 \Rightarrow \phi_2 \quad \Gamma \vdash \phi_1}{\Gamma \vdash \phi_2}$$

$$\frac{\Gamma \vdash e_0 : \sigma \rightarrow \tau \quad \Gamma \vdash e_1 : \sigma}{\Gamma \vdash (e_0 \ e_1) : \tau}$$

( $\wedge$ -intro) 
$$\frac{\Gamma \vdash \phi \quad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}$$

$$\frac{\Gamma \vdash e_1 : \sigma \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1, e_2) : \sigma * \tau}$$

( $\wedge$ -elim) 
$$\frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \quad \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}$$

$$\frac{\Gamma \vdash e : \sigma * \tau}{\Gamma \vdash \#1 e : \sigma} \quad \frac{\Gamma \vdash e : \sigma * \tau}{\Gamma \vdash \#2 e : \tau}$$



# Proof- and Typing Rules Side-by-Side

*intuitionistic logic*

$\lambda^{\rightarrow}$  or System F type system

( $\vee$ -intro)	$\frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \quad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$	$\frac{\Gamma \vdash e : \sigma}{\Gamma \vdash \mathbf{inl}_{\sigma+\tau} : e\sigma + \tau} \quad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathbf{inr}_{\sigma+\tau} : e\sigma + \tau}$
( $\vee$ -elim)	$\frac{\Gamma \vdash \phi \vee \psi \quad \Gamma \vdash \phi \rightarrow \chi \quad \Gamma \vdash \psi \rightarrow \chi}{\Gamma \vdash \chi}$	$\frac{\Gamma \vdash e : \sigma + \tau \quad \Gamma \vdash e_1 : \sigma \rightarrow \rho \quad \Gamma \vdash e_2 : \tau \rightarrow \rho}{\Gamma \vdash \mathbf{case } e_0 \mathbf{ of } e_1 \mid e_2 : \rho}$
( $\forall$ -intro)	$\frac{\Gamma, P \vdash \phi}{\Gamma \vdash \forall P. \phi}$	$\frac{\Delta, \alpha; \Gamma \vdash e : \tau \quad \alpha \notin FV(\Gamma)}{\Delta; \Gamma \vdash (\Lambda \alpha. e) : \forall \alpha. \tau}$
( $\forall$ -elim)	$\frac{\Gamma \vdash \forall P. \phi}{\Gamma \vdash \phi\{\psi/P\}}$	$\frac{\Delta; \Gamma \vdash e : \forall \alpha. \tau \quad \Delta \vdash \sigma}{\Delta; \Gamma \vdash (e \sigma) : \tau\{\sigma/\alpha\}}$

# The Curry-Howard Isomorphism

- A.k.a as “Propositions as Types”

<i>type theory</i>		<i>logic</i>	
$\tau$	type	$\phi$	proposition
$\tau$	inhabited type	$\phi$	theorem
$e$	well-typed program	$\pi$	proof
$\rightarrow$	function space	$\rightarrow$	implication
$*$	product	$\wedge$	conjunction
$+$	sum	$\vee$	disjunction
$\forall$	type quantifier	$\forall$	2nd order quantifier
$B$	inhabited type	$\top$	truth
void	uninhabited type	$\perp$	falsity

# Logical Interpretation of Program Transformations

- Reduction = Proof Normalization
  - Existence of normal form can be formalized as *Cut Elimination Theorem* (Gentzen's "Hauptsatz")
  - Typically presented using sequent calculus rather than natural deduction
- CPS Transformation = Double Negation