CS 611
Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 26

Type reconstruction

1 Nov 04

Type reconstruction

Simple typed language:

$e ::= x \mid b \mid \lambda x{:}\tau\ .\ e \mid e_1\ e_2 \mid e_1 + e_2$
$\quad \mid$ if e_0 then e_1 else $e_2 \mid$ let $x{=}e_1$ in e_2
$\quad \mid$ rec $y{:}\tau_1{\to}\tau_2.(\lambda x.e)$

$\tau ::=$ unit $\mid$ bool $\mid$ int $\mid \tau_1{\to}\tau_2$

- Question: Do we really need to write type declarations?

$e ::= \ldots \mid \lambda x.e \mid \ldots \mid$ rec $y.(\lambda x.e)$

Typing rules

$e ::= x \mid b \mid \lambda x.e \mid e_1\ e_2 \mid e_1 \oplus e_2$
$\mid$ if e_0 then e_1 else $e_2 \mid$ let $x{=}e_1$ in $e_2 \mid$ rec $y.\lambda x.e$

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

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

Problem: how does type checker construct proof?

Guess τ, τ' ?

Example

```
let square = λz.z*z in
  (λf.λx.λy.
      if (f x y)
            then (f (square x) y)
            else (f x (f x y)))
```

What is the type of this program?

Manual type inference

```
let square = λz.z*z in
   (λf.λx.λy.
       if (f x y)
             then (f (square x) y)
             else (f x (f x y)))
```

z : int

s, square : int→int

f : $\tau_x \to \tau_y \to$bool

y : τ_y =bool

x : τ_x = int

Answer:

(int→bool→bool)→int→bool→bool

Type inference

- Goal: reconstruct types even after erasure
- Idea: run ordinary type-checking algorithm, generate *type equations* on *type variables*

$$\dfrac{\dfrac{\dfrac{f{:}T2, x{:}T5 \vdash f : int \to T6 \quad f{:}T2, x{:}T5 \vdash 1 : int}{f{:}T2, x{:}T5 \vdash f\ 1 : T6}}{f{:}T2 \vdash \lambda x.\ f\ 1 : T1\ (=T5 \to T6)} \quad \dfrac{}{y{:}T3 \vdash y : T4}\ (T3=T4)}{\vdash \lambda f.\ \lambda x.\ f\ 1 : T2 \to T1 \qquad \vdash (\lambda y.y) : T2 \quad (T2=T3 \to T4)}$$

$$\vdash (\lambda f.\lambda x.\ (f\ 1))\ (\lambda y.y) : T1$$

T2 = T3→T4, T3 = T4, T1 =T5→T6, T2 = int→T6

Typing rules

$$\frac{}{\Gamma \vdash n : \text{int}} \qquad \frac{x \in \text{dom}(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma, x{:}T_x \vdash e : \tau}{\Gamma \vdash \lambda x\,.\,e : T_x \to \tau}$$

$$\frac{\Gamma \vdash e_0 : \tau_0 \quad \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \tau_1 = \tau_2,\ \tau_0 = \text{bool}}{\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau_1}$$

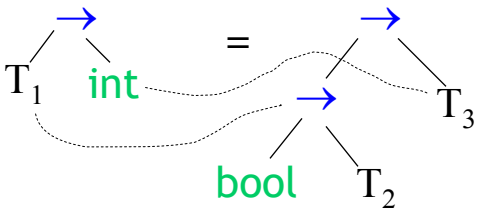
$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau_2} \qquad \frac{\Gamma, x{:}T_1, y{:}T_1 \to T_2 \vdash e : \tau \quad \tau = T_2}{\Gamma \vdash \text{rec } y.\lambda x.e : T_1 \to T_2}$$

$$\frac{\Gamma \vdash e_0 : \tau_0 \quad \Gamma \vdash e_1 : \tau_1 \quad \tau_0 = \tau_1 \to T_2}{\Gamma \vdash e_0\ e_1 : T_2}$$

Only type metavariables on RHS of premises

Unification

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



Robinson's algorithm (1965)

- *Unification* produces *weakest substitution* that equates two trees
  - $T_1 \to \text{int} = (\text{bool} \to T_2) \to T_3$ equated by any $T_1 \mapsto \text{bool} \to T_2$, $T_3 \mapsto \text{int}$, $T_2 \mapsto \tau$
  - **Defn.** S_1 is weaker than S_2 if $S_2 = S_3 \circ S_1$ for S_3 a non-trivial substitution
- **Unify**(E) where E is set of equations gives weakest equating substitution: define recursively

Unify$(T = \tau, E)$ = **Unify**$(E\{\tau/T\}) \circ [T \mapsto \tau]$
(if $T \notin \text{FTV}(\tau)$)

Rest of algorithm

Unify$(T = \tau, E)$ = **Unify**$(E\{\tau/T\}) \circ [T \mapsto \tau]$
(if $T \notin \text{FTV}[\![\tau]\!]$)

Unify$(\varnothing) = \varnothing$

Unify$(B = B, E)$ = **Unify**(E)

Unify$(B_1 = B_2, E)$ = ?

Unify$(T = T, E)$ = **Unify**(E)

Unify$(\tau_1 \to \tau_2 = \tau_3 \to \tau_4, E)$
= **Unify**$(\tau_1 = \tau_3, \tau_2 = \tau_4, E)$

Termination? *Degree* = (#vars, size(eqns))

Type inference algorithm

- $\mathcal{R}(e, \Gamma, S) = \langle \tau, S' \rangle$ means "Reconstructing the type of e in typing context Γ with respect to substitution S yields type τ, identical or stronger substitution S′ *or*

"S′ is weakest substitution no weaker than than S such that $S'(\Gamma) \vdash e : S'(\tau)$"

Define: **Unify**(E, S) = **Unify**$(SE) \circ S$

– solve substituted equations E and fold in new substitutions

Inductive defn of inference

- $\mathcal{R}(e, \Gamma, S) = \langle \tau, S' \rangle \Leftrightarrow$ "S′ is weakest substitution stronger than (or same as) S such that $S'(\Gamma) \vdash e : S'(\tau)$"
- **Unify**(E, S) = **Unify**$(SE) \circ S$

$\mathcal{R}(n, \Gamma, S) = \langle \text{int}, S \rangle$ $\quad\quad$ $\mathcal{R}(\text{true}, \Gamma, S) = \langle \text{bool}, S \rangle$

$\mathcal{R}(x, \Gamma, S) = \langle \Gamma(x), S \rangle$

$\mathcal{R}(e_1\, e_2, \Gamma, S) = \text{let } \langle T1, S1 \rangle = \mathcal{R}(e_1, \Gamma, S) \text{ in}$
$\quad\quad \text{let } \langle T2, S2 \rangle = \mathcal{R}(e_2, \Gamma, S1) \text{ in}$
$\quad\quad \langle T_f, \textbf{Unify}(T1 = T2 \rightarrow T_f, S2) \rangle$

$\mathcal{R}(\lambda x.e, \Gamma, S) = \text{let } \langle T1, S1 \rangle = \mathcal{R}(e, \Gamma[x \mapsto T_f], S) \text{ in}$
$\quad\quad \langle T_f \rightarrow T1, S1 \rangle$

where T_f is "fresh" (not mentioned anywhere in e, Γ, S)

Example

$\mathcal{R}((\lambda x.x)\ 1, \emptyset, \emptyset) =$

let $\langle T1, S1\rangle = \mathcal{R}(\lambda x.x, \emptyset, \emptyset)$ in

let $\langle T2, S2\rangle = \mathcal{R}(1, \emptyset, S1)$ in

$\langle T3, \textbf{Unify}(T1 \to T3 = T2, S2)\rangle$

$\mathcal{R}(\lambda x.x, \emptyset, \emptyset)$ = let $\langle T1, S1\rangle = \mathcal{R}(x, \Gamma[x \mapsto T4], \emptyset)$ in
$\langle T4 \to T1, S1\rangle$
$= \langle T4 \to T4, \emptyset\rangle$

= let $\langle T2, S2\rangle = \mathcal{R}(1, \emptyset, \emptyset)$ in
$\langle T3, \textbf{Unify}(T2 \to T3 = T4 \to T4, \emptyset)\rangle$

= $\langle T3, \textbf{Unify}(\text{int} \to T3 = T4 \to T4, \emptyset)\rangle$

= $\langle T3, \textbf{Unify}(\text{int} = T4, T3 = T4, \emptyset)\rangle$

= $\langle T3, \textbf{Unify}(T3 = \text{int}, [T4 \mapsto \text{int}])\rangle$

= $\langle T3, [T3 \mapsto \text{int}, T4 \mapsto \text{int}])\rangle$

Implementation

- Can implement with imperative update:

```
datatype type = Int | Bool | Arrow of type * type
              | TypeVar of type option ref
fun freshTypeVar() =
  TypeVar(ref NONE)
fun resolve(t: type) = case t of
    TypeVar(ref (SOME t')) => t'
  | _ => t
fun unify(t1: type, t2: type) : unit =
  case (resolve t1, resolve t2) of
    (TypeVar(r as ref NONE), t2) => r := t2
  | (t1, TypeVar(r as ref NONE)) => r := t1
  | (Arrow(t1,t2), Arrow(t3,t4)) =>
     unify(t1,t3); unify(t2,t4)
  | (Int, Int) => () | (Bool,Bool) => ()
  | _ => raise Fail "Can't unify types"
```

Polymorphism

$\mathcal{R}(\lambda x.x, \varnothing, \varnothing)$ = let $\langle T1, S1\rangle = \mathcal{R}(x, \Gamma[x \mapsto T4], \varnothing)$ in
$\langle T4 \to T1, S1\rangle$
$= \langle T4 \to T4, \varnothing\rangle$

- Reconstruction algorithm doesn't solve type fully… opportunity!
- $\lambda x.x$ can have type $T4 \to T4$ for any T4
  - polymorphic (= "many shape") term
  - Could reuse same expression multiple places in program, with different types:

```
let id = (λx.x) in … (f id) … (g x id) … id
```