CS 611
Advanced Programming Languages

Andrew Myers
Cornell University

Lecture 32
Type inference & ML polymorphism
10 Nov 00

Type inference

Simple typed language:

$e ::= x \mid b \mid \text{fn}\ x{:}\tau\ .\ e \mid e_1\ e_2 \mid e_1 \oplus e_2$
$\mid \text{if}\ e_0\ e_1\ e_2 \mid \text{let}\ x{=}e_1\ \text{in}\ e_2$
$\mid \text{rec}\ y{:}\tau_1{\to}\tau_2.\ \text{fn}\ x.e$

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

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

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

Typing rules

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

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

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

Problem: how does type checker construct proof? Guess τ ?

Example

```
let square = rec s.fn z. z*z in
  (fn f.fn x.fn y.
     if (f x y)
          (f (square x) y)
          (f x (f x y)))
```

What is the type of this program?

Example

```
let square = rec s.fn z. z*z in
  (fn f.fn x.fn y.
     if (f x y)
          (f (square x) y)
          (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

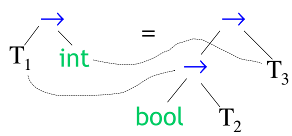
- Goal: reconstruct types even after erasure
- Idea: run ordinary type-checking algorithm, generate *type equations*

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

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

Unification

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



Robinson's algorithm (1965)

- *Unification* produces *weakest substitution* that equates two trees
  - $T_1 \rightarrow \text{int} = (\text{bool} \rightarrow T_2) \rightarrow T_3$ equated by any $T_1 \mapsto \text{bool} \rightarrow T_2$, $T_3 \mapsto \text{int}$, $T_2 \mapsto ?$
  - **Defn.** S_1 is weaker than S_2 if $S_2 = S_3 \circ S_1$ for S_3 a non-identity 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 $T \notin FTV[\![\tau]\!]$)

Rest of algorithm

Unify($T = \tau$, E) = **Unify**($E\{\tau/T\}$)$\circ[T \mapsto \tau]$
(if $T \notin FTV[\![\tau]\!]$)
Unify($\emptyset$) = $\emptyset$
Unify($B = B$, E) = **Unify**(E)
Unify($B_1 = B_2$, E) = ?
Unify($T = T$, E) = **Unify**(E)
Unify($\tau_1 \rightarrow \tau_2 = \tau_3 \rightarrow \tau_4$, E)
= **Unify**($\tau_1 = \tau_3$, $\tau_2 = \tau_4$, E)
Termination?

Type inference algorithm

- $\mathcal{R}(e, \Gamma, S) = \langle \tau, S' \rangle$ means "Reconstructing the type of e in type context Γ with respect to substitution S yields type τ, stronger substitution S′ *or*
"S′ is weakest subst. stronger 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 subst. stronger than S such that $S'(\Gamma) \vdash e : S'(\tau)$"
- **Unify**(E, S) = **Unify**(SE)$\circ S$

$\mathcal{R}(n, \Gamma, S) = \langle \text{int}, S \rangle$ $\quad$ $\mathcal{W}(\#t, \Gamma, S) = \langle \text{bool}, S \rangle$
$\mathcal{R}(x, \Gamma, S) = \langle \Gamma(x), S \rangle$
$\mathcal{R}(e_1\, e_2, \Gamma, S)$ = let $\langle T1, S1 \rangle = \mathcal{R}(e_1, \Gamma, S)$ in
let $\langle T2, S2 \rangle = \mathcal{R}(e_2, \Gamma, S1)$ in
$\langle T_f$, **Unify**($T2 \rightarrow T_f = T1, S2$)$\rangle$
$\mathcal{R}(\text{fn } x.e, \Gamma, S)$ = let $\langle T1, S1 \rangle = \mathcal{R}(e, \Gamma[x \mapsto T_f], S)$ in
$\langle T_f \rightarrow T1, S1 \rangle$

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

Example

$\mathcal{R}((\text{fn } x\ x)\ 1, \emptyset, \emptyset)$ =
let $\langle T1, S1 \rangle = \mathcal{R}(\text{fn } x\ x, \emptyset, \emptyset)$ in
let $\langle T2, S2 \rangle = \mathcal{R}(1, \emptyset, S1)$ in
$\langle T3$, **Unify**($T1 \rightarrow T3 = T2, S2$)$\rangle$

$\mathcal{R}(\text{fn } x\ x, \emptyset, \emptyset)$ = let $\langle T1, S1 \rangle = \mathcal{R}(x, \Gamma[x \mapsto T4], \emptyset)$ in
$\langle T4 \rightarrow T1, S1 \rangle$
$= \langle T4 \rightarrow T4, \emptyset \rangle$

= let $\langle T2, S2 \rangle = \mathcal{R}(1, \emptyset, \emptyset)$ in
$\langle T3$, **Unify**($T2 \rightarrow T3 = T4 \rightarrow T4, \emptyset$)$\rangle$
= $\langle T3$, **Unify**($\text{int} \rightarrow T3 = T4 \rightarrow T4, \emptyset$)$\rangle$
= $\langle T3$, **Unify**($\text{int} = T4$, $T3 = T4, \emptyset$)$\rangle$
= $\langle T3$, **Unify**($T3 = \text{int}$, $[T4 \mapsto \text{int}]$)$\rangle$
= $\langle T3, [T3 \mapsto \text{int}, T4 \mapsto \text{int}] \rangle$

Polymorphism

$\mathcal{R}$(fn x x, ∅, ∅) = let ⟨T1, S1⟩ = $\mathcal{R}$(x, Γ[x↦T4], ∅) in ⟨T4→T1, S1⟩
= ⟨T4→T4, ∅⟩

- Reconstruction algorithm doesn't solve type fully… opportunity!
- fn x x can have type T4→T4 for any T4!
  - polymorphic (= "many shape") term
  - be nice to reuse same expression multiple places in program with different types:

let id = (fn x x) in … (f id) … (g x id) … id

Polymorphic types

- Type expression may have some unsolved type identifiers after type reconstruction
- Type T4→T4 is a *type schema* that can be instantiated with any T4 to make a type
- Idea: allow "let" to bind identifiers to polymorphic terms
  - type context Γ maps variable either to
    - type τ or
    - type schema $\forall T_1,\ldots,T_n.\ \tau$ where $FTV(\tau) \subseteq \{T_1,\ldots,T_n\}$
- Can still do type inference! (ML)

Typing rules

$\Gamma \in Var \to \sigma$

$\sigma ::= \tau \mid \forall T_1,\ldots,T_n.\ \tau$

$\Delta = \{T_1,\ldots,T_n\}$ (Δ : set of legal type variables)

$\Delta \vdash \tau$ (judgment: τ is well formed)

$\Delta\ ;\Gamma \vdash e : \tau$

$$\frac{}{\Delta\ ;\Gamma, x:\tau \vdash x:\tau} \qquad \frac{\Delta\ ;\Gamma, x:\tau \vdash e:\tau' \qquad \Delta \vdash \tau, \tau'}{\Delta\ ;\Gamma \vdash \text{fn}\ x\ e : \tau\to\tau'}$$

$$\frac{\Delta \vdash \tau_i \quad {}^{i\in 1..n}}{\Delta\ ;\Gamma, x:(\forall T_1,\ldots,T_n.\ \tau) \vdash x : \tau\{\tau_i/T_i\ {}^{i\in 1..n}\}}$$

More typing rules

$$\frac{\Delta\ ;\Gamma \vdash e_1 : \tau\to\tau' \quad \Delta\ ;\Gamma \vdash e_2 : \tau\to\tau' \quad \Delta \vdash \tau, \tau'}{\Delta\ ;\Gamma \vdash e_1\ e_2 : \tau'}$$

$$\frac{\begin{array}{ll}\Delta\cup\{T_1,\ldots,T_n\}\ ;\Gamma \vdash e_1 : \tau & \Delta\cup\{T_1,\ldots,T_n\}\vdash\tau \\ \Delta\ ;\Gamma, x:\forall T_1,\ldots,T_n.\tau \vdash e_2 : \tau' & \Delta\vdash\tau'\end{array}}{\Delta\ ;\Gamma \vdash \text{let}\ x = e_1\ \text{in}\ e_2 : \tau'}$$

$$\frac{T_1\ ;\emptyset\vdash (\text{fn x x}) : T_1\to T_1 \qquad \dfrac{\emptyset;\ \text{id} : \forall T_1.T_1\to T_1 \vdash \text{id} : \text{int}\to\text{int}}{\emptyset;\ \text{id} : \forall T_1.T_1\to T_1 \vdash \text{id 2} : \text{int}}}{\emptyset;\emptyset\vdash \text{let id = (fn x x) in id 2 : int}}$$

Algorithm $\mathcal{W}$ (Milner)

- Infers types in language with let-bound type schemas! (& letrec)
  - Built into ML
- $\mathcal{W}$(e, Γ, S) = ⟨τ, S'⟩ gives type, subst S' as before, (but Γ can map vars to type schemas)

$\mathcal{W}$(x, Γ, S) = case Γ(x) of
τ ⇒ ⟨τ, S⟩
| $\forall T_1,\ldots,T_n.\tau \Rightarrow \langle\tau\{T_{fi}/T_i\}, S\rangle$

$\mathcal{W}$(letrec $x = e_1$ in e_2, Γ, S) =
let Γ' = Γ[x↦T_f] in let ⟨T1, S1⟩ = $\mathcal{W}$(e_1, Γ', S) in
let S2=**Unify**({T_f = T1}, S1) in
let Γ'' = Γ[x↦Generic(T1, Γ, S2)] in
$\mathcal{W}$(e_2, Γ'', S2)

Generic(τ,Γ,S) = $\forall T_1,\ldots,T_n.S\tau$ where $\{T_1,\ldots,T_n\}$ = FTV(Sτ) – FTV(SΓ)