Simple Type Inference

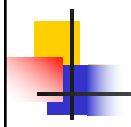
1

Announcements

- Graded HW4!
- Graded HW5!
- HW6 up!
- Working on paper list and presentation schedule

2

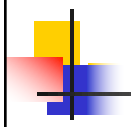
2



Outline

- Simple type inference
  - Expressions, types and type environment (last time)
  - Goal and intuition (last time)
  - Equality constraints
  - Substitution
  - Robinson's unification
  - Type inference strategies
    - Algorithm V (Strategy One) and
    - Algorithm V (Strategy Two)

Algorithm W



Type Inference

The task of type inference is to

Reject bad programs with a decent error message

Elaborate good programs

Expressions

Language of the Simply Typed Lambda calculus:

$$E ::= c \mid x \mid \lambda x.E \mid E_1\ E_2 \mid$$
$$E_1 + E_2 \mid E_1 = E_2 \mid$$
$$\textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3 \mid$$
$$\textbf{let } x = E_1 \textbf{ in } E_2$$

letrec

$\Gamma, x{:}\sigma \vdash E_1 : \sigma$

For the purposes of type inference, there are no types in syntax

The type of each subexpression is derived by simple type inference

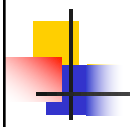
Types

Types (as known as simple types or monotypes):

$$\tau ::= b \mid \tau_1 \rightarrow \tau_2 \mid t$$

t is a type variable (tyvar)

b is a base type
Assume **Int** and **Bool**



Type Environment

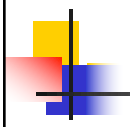
Type environment Gamma maps identifiers (variables) to types:

Gamma ::= Identifiers -> Types

For example, we can only type subexpression

```
(f x)
```

in a type environment that binds identifies `f` and `x` to types. E.g., in `Gamma = [f :: t -> t, x :: t]`



Goal and Intuition

Given `\x -> \y -> x`

Deduce `\x -> \y -> x :: t1 -> t2 -> t1`

1. Construct parse tree for expression. Associate a fresh tyvar to each identifier and each subexpression

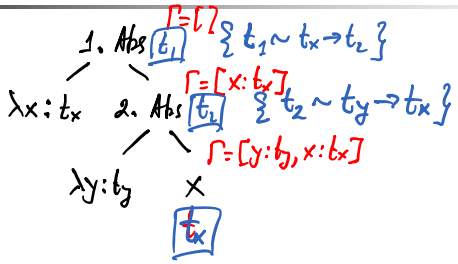
2. Generate equality constraints (based on typing rules)

3. Solve equality constraints using unification

4. Deduce type for expression

```
\x -> \y -> x
```

$$\frac{\Gamma, x:\sigma \vdash E : \tau}{\Gamma \vdash \lambda x:\sigma . E : \sigma \to \tau}$$



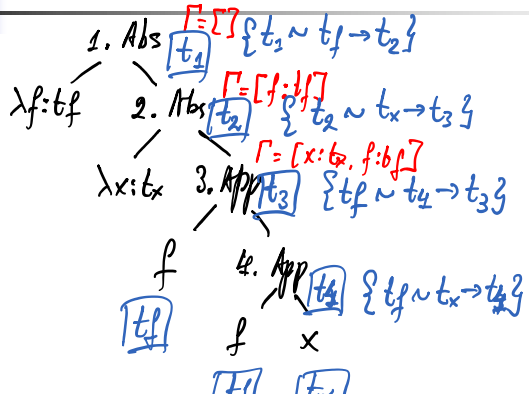
Constraints: $\{t_1 \sim t_x \to t_2,\ t_2 \sim t_y \to t_x\}$

Thus: $t_1 \sim t_x \to t_y \to t_x$

9

```
\f -> \x -> f (f x)
```

$$\frac{\Gamma \vdash E_1 : \sigma \to \tau \quad E_2 : \sigma}{\Gamma \vdash E_1\ E_2 : \tau}$$



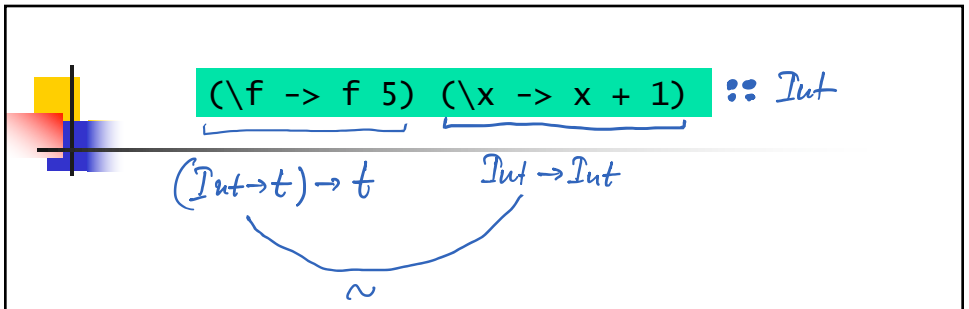
Elaborate t1: $\sigma(t_x \to t_x) \to t_x \to t_x$

Constraints $t_1 \sim t_f \to t_2,\ t_2 \sim t_x \to t_3 \Rightarrow t_1 \sim t_f \to t_x \to t_3$

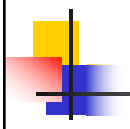
$t_f \sim t_4 \to t_3$

$t_f \sim t_x \to t_4$ $\Rightarrow \{t_x \sim t_2 \sim t_4\}$ (tx)

10



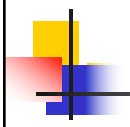

```
let f = \x -> x in f 1
```



Equality Constraints

Two key concepts

- Equality
  - What does it mean for two types to be equal?
  - Structural equality
- Unification
  - Can two types be made equal by choosing appropriate substitutions for their type variables?
  - Robinson's unification algorithm



What does it mean for two types τ_a and τ_b to be equal?

Structural equality

Suppose $\tau_a = t_1 \rightarrow t_2$

$\tau_b = t_3 \rightarrow t_4$

Structural equality entails

$\tau_a \sim \tau_b$ means $t_1 \rightarrow t_2 \sim t_3 \rightarrow t_4$ iff $t_1 \sim t_3$ and $t_2 \sim t_4$

Can two types be made equal by choosing appropriate substitutions for their type variables?

Robinson's unification algorithm

Suppose τ_a = **Int**→t_1 = Int → Bool

τ_b = t_2→**Bool** = Int → Bool

Can we unify τ_a and τ_b? Yes, if **Bool**/t_1 and **Int**/t_2

Suppose τ_a = **Int** →t_1

τ_b = **Bool**→**Bool**

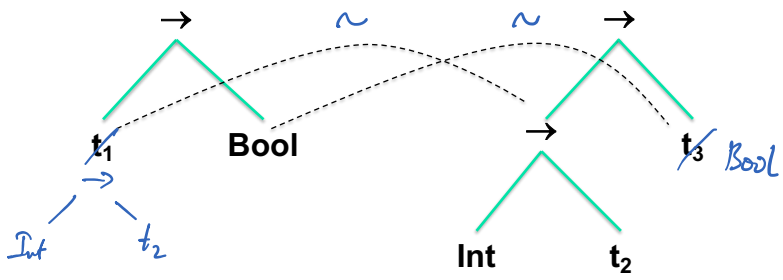
Can we unify τ_a and τ_b? No.

15

Example

τ_a τ_b

$t_1 \rightarrow \textbf{Bool} \sim (\textbf{Int} \rightarrow t_2) \rightarrow t_3$



Yes, if **Int**→t_2/t_1 and **Bool**/t_3

16

Substitution

Language of types

τ ::= **b** // base type: **Int** and **Bool**

| **t** // type variable (tyvar)

| $\tau_1 \rightarrow \tau_2$ // function type

A substitution is a map

S : Type Variable → Type

S = [τ_1/t_1, … τ_n/t_n] // substitute type τ_i for tyvar t_i

A substitution instance τ' = S τ

S = [t_0→Bool / t_1] $\tau = t_1 \rightarrow t_1$ then

S τ = S($t_1 \rightarrow t_1$) = ($t_1 \rightarrow t_1$)[t_0→Bool / t_1] = (t_0→Bool) → (t_0→Bool)

Exercises

Substitutions can be composed

S_1 = [t_0→Bool/t_1]

S_2 = [Int/t_0]

$\tau = t_1 \rightarrow t_1$

$S_2 S_1$ = [Int/t_0] ∘ [t_0→Bool /t_1]

same as:

S = [t_0→Bool/t_1 , Int/t_0]

$S_2 S_1 \tau = S_2 (S_1 (t_1 \rightarrow t_1)) = ?$

$((t_1 \rightarrow t_1)[t_0 \rightarrow Bool / t_1])[Int/t_0] =$

$((t_0 \rightarrow Bool) \rightarrow t_0 \rightarrow Bool)[Int/t_0] =$

$(Int \rightarrow Bool) \rightarrow Int \rightarrow Bool$

Substitutions can be composed

$S_1 = [\ t_x/t_1\ ]$

$S_2 = [\ t_x/t_2\ ]$

$\tau = t_2 \to t_1$

$S_2\, S_1\, \tau = ?$

$((t_2 \to t_1)[t_x/t_1])[t_x/t_2] =$

$t_x \to t_x$

Substitutions can be composed

$S_1 = [\ t_1/t_2\ ]$

$S_2 = [\ t_3/t_1\ ]$

$S_3 = [\ t_4 \to Int/t_3\ ]$

$t_1 \to t_2 \to t_3$ same as

$t_1 \to (t_2 \to t_3)$

$=$

$(t_1 \to t_2) \to t_3$

$\tau = t_1 \to t_2$

$S_3\, S_2\, S_1\, \tau = ?$

$(((t_1 \to t_2)[t_1/t_2])[t_3/t_1])[t_4 \to Int/t_3]$

$((t_1 \to t_1)[t_3/t_1])[t_4 \to Int/t_3]$

$(t_3 \to t_3)[t_4 \to Int/t_3] = (t_4 \to Int) \to (t_4 \to Int)$

Principal Unifier

A unifier is a substitution that unifies (i.e., makes equal) a set of constraints

A principal unifier is a most general unifier of a set of constraints

$\{ (t_1 \to t_1) \to t_1 \to t_1 \sim t_2 \to t_3 \}$

$[t_1 \to t_1 \,/\, t_2,\ t_1 \to t_1 \,/\, t_3]$

$\{ t_1 \to t_1 \sim t_2 \to t_3 \}$

$[t_1/t_2,\ t_1/t_3]$ u_1

$[t_2/t_1,\ t_2/t_3]$ u_2

$[Int/t_1,\ Int/t_2,\ Int/t_3]$

$[Int \to Int / t_1 \ldots$

$[t_5 \to t_6/t_1,\ t_5 \to t_6/t_2,$
$t_5 \to t_6/t_3]$

Exercise

A principal unifier is the most general unifier of a set of constraints

Find principal unifiers (when they exist) for

$\{ Int \to Int \sim t_1 \to t_2 \}$ $[Int/t_1,\ Int/t_2]$

$\{ Int \sim Int \to t_2 \}$ DNE

$\{ t_1 \sim Int \to t_2 \}$ $[Int \to t_2 / t_1]$

$\{ t_1 \sim Int,\ t_2 \sim t_1 \to t_1 \}$ $[Int/t_1,\ Int \to Int/t_2]$

$\{ t_1 \to t_2 \sim t_2 \to t_3,\ t_3 \sim t_4 \to t_5 \}$ $[t_1/t_2,\ t_1/t_3,\ t_4 \to t_5/t_1]$

Unification

- **Unify**: tries to unify τ_1 and τ_2 and returns a **principal unifier for** $\tau_1 \sim \tau_2$ if unification is successful

def **Unify**(τ_1,τ_2) =

This is the occurs check!

case (τ_1,τ_2)

- (τ_1,$\mathbf{t_2}$) = [τ_1/$\mathbf{t_2}$] provided $\mathbf{t_2}$ does not occur in τ_1
- ($\mathbf{t_1}$,τ_2) = [τ_2/$\mathbf{t_1}$] provided $\mathbf{t_1}$ does not occur in τ_2
- ($\mathbf{b_1}$,$\mathbf{b_2}$) = if (eq? $\mathbf{b_1}$ $\mathbf{b_2}$) then [ ] else ***fail***
- ($\tau_{11}\rightarrow\tau_{12}$, $\tau_{21}\rightarrow\tau_{22}$) = let $\mathbf{S_1}$ = **Unify**(τ_{11},τ_{21})
  $\mathbf{S_2}$ = **Unify**($\mathbf{S_1}\,\tau_{12}$, $\mathbf{S_1}\,\tau_{22}$)
  in $\mathbf{S_2}\,\mathbf{S_1}$ // compose substitutions

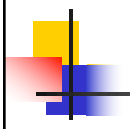
otherwise = ***fail***

Exercise

Unify (**Int**→**Int**, $\mathbf{t_1}\rightarrow\mathbf{t_2}$) yields ?

Unify (**Int**, **Int**→$\mathbf{t_2}$) yields ?

Unify ($\mathbf{t_1}$, **Int**→$\mathbf{t_2}$) yields ?



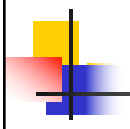
Unify Set of Constraints C

Robinson's algorithm unifies (i.e., solves) a single constraint $\tau_1 \sim \tau_2$.

What if we have a set of constraints?

Intuition:

1. Pick a constraint $\tau_1 \sim \tau_2$ from the set
2. Solve $\tau_1 \sim \tau_2$ either failing or succeeding getting subst **S**
   - If fail, then done, constraints cannot be unified
   - If success, then first apply **S** on remaining constraints as S carries structure that must be taken into account



Unify Set of Constraints C

UnifySet: tries to unify **C** and returns a **principal unifier for C** if unification is successful

```
def UnifySet (C) =
  if C is Empty Set then [] // Empty subsitution
  else let
         C = { τ1 ~ τ2 } U C'
         S = Unify (τ1,τ2) // Unify returns a substitution S
       in
         UnifySet ( S(C') ) S
         // Compose the substitutions
```

$(\tau_1 \to \tau_2 \;:\; C')$

$S \mathbin{++} \text{UnifySet}\,(S(C'))$

Exercise

UnifySet { $t_1 \sim \mathbf{Int}$, $t_2 \sim t_1 \to t_1$ } yields ?

UnifySet { $t_1 \to t_2 \sim t_2 \to t_3$, $t_3 \sim t_4 \to t_5$ } yields ?

UnifySet { $t_f \sim t_2 \to t_1$, $t_f \sim t_x \to t_2$ } yields ?

UnifySet { $t_2 \sim t_4 \to t_1$, $t_2 \sim t_f \to t_3$, $t_4 \sim t_x \to \mathbf{Int}$, $t_f \sim \mathbf{Int} \to t_3$, $t_x \sim \mathbf{Int}$ } yields ?

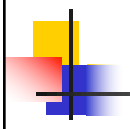
27

27

Outline

- Simple type inference
  - Expressions, types and type environment
  - Goal and intuition
  - Equality constraints
  - Substitution
  - Robinson's unification
  - Type inference strategies
    - Algorithm V (Strategy One) and
    - Algorithm V (Strategy Two)

28



Type Inference Strategies

Strategy One aka constraint-based typing (Haskell)

Traverse expression's parse tree and generate constraints. Solve constraints offline producing substitution map S. Finally, apply S on expression tyvar to infer the principal type of expression

Strategy Two (Classical Hindley Milner)

Generate and solve constraints on-the-fly while traversing parse tree. Build and apply substitution map incrementally

Constraint Generation Strategy One

def V(Γ, E) = case E of

c -> ({}, TypeOf(c))

x -> if (x NOT in Dom(Γ)) then *fail*
else ({}, Γ(x))

\x -> E_1 -> let (C_{E1},T_{E1}) = V(Γ+{x:t_x},E_1) – t_x is fresh tyvar
in (C_{E1},t_x→T_{E1})

def V(Γ, E) = case E of

…

$E_1\ E_2$ -> let $(C_{E1},T_{E1}) = V(\Gamma,E_1)$

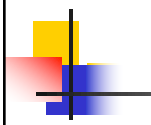
$(C_{E2},T_{E2}) = V(\Gamma,E_2)$

in $(C_{E1} + C_{E2} + \{T_{E1} \sim T_{E2} \rightarrow t\}, t)$ -- t is fresh tyvar

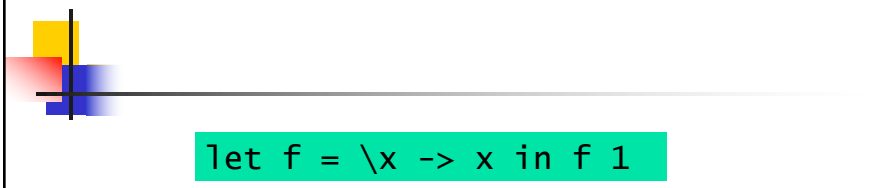
let x = E_1 in E_2 -> let $(C_{E1},T_{E1}) = V(\Gamma+\{x:t_x\},E_1)$

$(C_{E2},T_{E2}) = V(\Gamma+\{x:T_{E1}\},E_2)$

in $(C_{E1} + C_{E2} + \{t_x \sim T_{E1}\}, T_{E2})$



```
(\f -> f 5) (\x -> x + 1)
```



```
let f = \x -> x in f 1
```


On-the-fly Generation and Resolution

Strategy Two

def V(Γ, E) = case E of

c -> ([], TypeOf(c))

x -> if (x NOT in Dom(Γ)) then *fail*
else ([], T_E)

\x -> E_1 -> let (S_{E1},T_{E1}) = V(Γ+{x:t_x},E_1)
in (S_{E1}, S_{E1}(t_x)→T_{E1})

def V(Γ, E) = case E of

$E_1\ E_2$ -> let $(S_{E1},T_{E1}) = V(\Gamma,E_1)$

$(S_{E2},T_{E2}) = V(S_{E1}(\Gamma),E_2)$

$S = Unify(S_{E2}(T_{E1}),T_{E2}\rightarrow t)$

in $(S\ S_{E2}\ S_{E1}, S(t))$ // $S\ S_{E2}\ S_{E1}$

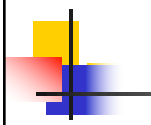
let $x = E_1$ in E_2 -> let $(S_{E1},T_{E1}) = V(\Gamma+\{x:t_x\},E_1)$

$S = Unify(S_{E1}(t_x),T_{E1})$

$(S_{E2},T_{E2}) = V(S\ S_{E1}(\Gamma)+\{x:S(T_{E1})\},E_2)$

in $(S_{E2}\ S\ S_{E1}, T_{E2})$

35



```
(\f -> f 5) (\x -> x + 1)
```


36