

TYPE INFERENCE (II)

SOLVING CONSTRAINTS (RECAP)

- Judgement form:
 - $G \dashv\vdash u \implies e : t, q$
 - u is untyped expression
 - $e : t$ is a term scheme
 - q is a set of constraints
- A **solution** to a system of type constraints is a **substitution S**
 - a **function** from *type variables* to *type schemes*
 - substitutions are defined on all type variables (a total function), but only some of the variables are actually changed:
 - $S(a) = a$ (for most variables a)
 - $S(a) = s$ (for some a and some type scheme s)
 - $\text{dom}(S) = \text{set of variables s.t. } S(a) \neq a$

SUBSTITUTIONS

- Given a substitution S , we can define a function S^* from type schemes (as opposed to type variables) to type schemes:
 - $S^*(\text{int}) = \text{int}$
 - $S^*(\text{bool}) = \text{bool}$
 - $S^*(s1 \rightarrow s2) = S^*(s1) \rightarrow S^*(s2)$
 - $S^*(a) = S(a)$
- For simplicity, next I will write $S(s)$ instead of $S^*(s)$
- s denotes type schemes, whereas a, b, c denote type variables
- This function **replaces all type variables in a type scheme.**
- There's no variable binding in the language of type scheme, hence no danger of **capturing!**

$\forall x. S(x)$

EXTENSIONS TO SUBSTITUTION

- Substitution can be extended pointwise to the typing context:

$$G := . \mid G, \underline{x : s}$$

$$S(.) = .$$

$$S(G, x:s) = S(G), x: S(s)$$

Similarly, substitution can be applied to the type annotations in an expression, e.g.:

$$S(x) = x$$

$$S(\underline{\lambda x:s.e}) = \underline{\lambda x:S(s)}.S(\underline{e})$$

$$S(\underline{\text{nil}[s]}) = \underline{\text{nil}[S(s)]}$$

COMPOSITION OF SUBSTITUTIONS

○ **Composition** ($U \circ S$) applies the substitution S and then applies the substitution U :

- $(U \circ S)(a) = U(S(a))$

○ We will need to compare substitutions

- $T \leq S$ if T is “more specific” than S

- $T \leq S$ if T is “less general” than S

- Formally: $T \leq S$ if and only if $T = U \circ S$ for some U

COMPOSITION OF SUBSTITUTIONS

○ Examples:

- example 1: any substitution is less general than the identity substitution I:
 - $S \leq I$ because $S = S \circ I$
- example 2:
 - $S(a) = \text{int}$, $S(b) = c \rightarrow c$
 - $T(a) = \text{int}$, $T(b) = c \rightarrow c$, $T(c) = \text{int}$
 - we conclude: $T \leq S$
 - if $T(a) = \text{int}$, $T(b) = \text{int} \rightarrow \text{bool}$ then T is unrelated to S (neither more nor less general)

$U \circ S$

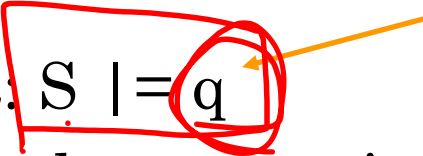
S

PRESERVATION OF TYPING UNDER TYPE SUBSTITUTION

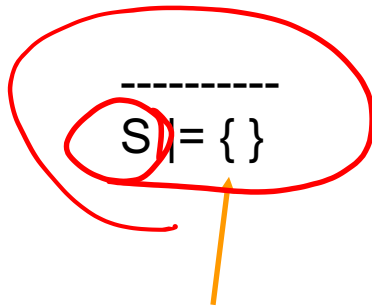
- Theorem: If S is any type substitution and $G \vdash e : s$, then $S(G) \vdash S(e) : S(s)$

Proof: straightforward induction on the typing derivations.

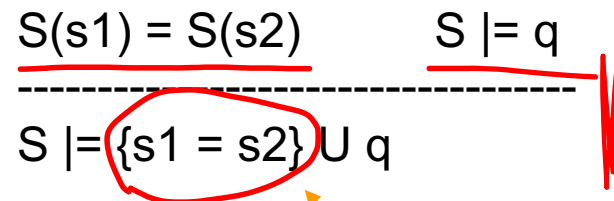
SOLVING A CONSTRAINT (FIRST ATTEMPT)

- Judgment format: $S \models q$
(S is a solution to the constraints q)


However this will not help you
Solve q to obtain S!

$$\frac{}{S \models \{\}}$$


any substitution is
a solution for the empty
set of constraints

$$\frac{S(s1) = S(s2) \quad S \models q}{S \models \{s1 = s2\} \cup q}$$


a solution to an equation
is a substitution that makes
left and right sides equal

MOST GENERAL SOLUTIONS

- S is the principal (most general) solution of a set of constraints q if
 - S \models q (S is a solution)
 - if T \models q then T \leq S (S is the most general one)
- Lemma: If q has a solution, then it has a most general one
- We care about principal solutions since they will give us the most general types for terms (polymorphism!)

reuse

EXAMPLES

○ Example 1

- $q = \{a=int, b=a\}$
- principal solution S :
 - $S(a) = S(b) = int$
 - $S(c) = c$ (for all c other than a, b)

EXAMPLES

○ Example 2

- $q = \{a=\text{int}, b=a, b=\text{bool}\}$
- principal solution S:
 - does not exist (there is no solution to q)

PRINCIPAL SOLUTIONS

- principal solutions give rise to most general *reconstruction* of typing information for a term:

- $\text{fun } f(x:a):a = x$
 - is a most general reconstruction

- $\text{fun } f(x:\text{int}):\text{int} = x$
 - is not

$$S(a) = \text{int}$$

$$\text{S} \circ \text{I}$$

✓ UNIFICATION

$$\boxed{S = \theta}$$

- **Unification:** An algorithm that provides the principal solution to a set of constraints (if one exists)
 - If one exists, it will be principal

UNIFICATION

- **Unification:** Unification systematically simplifies a set of constraints, yielding a substitution
- During simplification, we maintain (S, q)
 - S is the solution so far
 - q are the constraints left to simplify
 - Starting state of unification process: (I, q)
 - Final state of unification process: $(S, \{\})$

identity
substitution
is most
general

UNIFICATION MACHINE

$$e_1 \rightarrow e_2$$

- We can specify unification as a transition system:
 - $(S, q) \rightarrow (S', q')$
- Base types & simple variables:

I

$$\text{----- (u-int)} \\ (S, \{\text{int}=\text{int}\} \cup q) \rightarrow (S, q)$$

$$\text{----- (u-eq)} \\ (S, \{a=a\} \cup q) \rightarrow (S, q)$$

$$\text{----- (u-bool)} \\ (S, \{\text{bool}=\text{bool}\} \cup q) \rightarrow (S, q)$$

UNIFICATION MACHINE

- Functions:

$$\begin{array}{l} \text{-----} \quad \text{(u-fun)} \\ (S, \{s11 \rightarrow s12 = s21 \rightarrow s22\} \cup q) \rightarrow \\ (S, \{s11 = s21, s12 = s22\} \cup q) \end{array}$$

- Variable definitions

$$\begin{array}{l} \text{-----} \quad \text{(a not in FV(s)) (u-var1)} \\ (S, \{a=s\} \cup q) \rightarrow ([a=s] \circ S, q[s/a]) \end{array}$$

AV

$$\begin{array}{l} \text{-----} \quad \text{(a not in FV(s)) (u-var2)} \\ (S, \{s=a\} \cup q) \rightarrow ([a=s] \circ S, q[s/a]) \end{array}$$

OCCURS CHECK

- What is the solution to $\{a = a \rightarrow a\}$?
 - There is none!
 - The occurs check detects this situation

----- (a not in FV(s))
 $(S, \{a=s\} \cup q) \rightarrow ([a=s] \circ S, q[s/a])$

occurs check

IRREDUCIBLE STATES

- Recall: final states have the form $(S, \{\})$
- Stuck states (S, q) are such that every equation in q has the form:
 - $\text{int} = \text{bool}$
 - $s1 \rightarrow s2 = s$ (s not function type)
 - $a = s$ (s contains a)
 - or is symmetric to one of the above
- Stuck states arise when constraints are unsolvable

TERMINATION

- We want unification to terminate (to give us a type reconstruction **algorithm**)
- In other words, we want to show that there is no infinite sequence of states
 - $(S1, q1) \rightarrow (S2, q2) \rightarrow \dots$
- **Theorem:** unification algorithm always terminates.

TERMINATION

- We associate an ordering with constraints
 - q < q' if and only if
 - q contains fewer variables than q'
 - q contains the same number of variables as q' but fewer type constructors (ie: fewer occurrences of int, bool, or "→")
 - in other words, q is simpler than q'
 - This is a **lexicographic ordering** on (nv, nc)
 - nv: Number of variables
 - nc: Number of constructors
 - There is no infinite decreasing sequence of constraints
 - To prove termination, we must demonstrate that every step of the algorithm reduces the size of q according to this ordering



21 > 16

26 > 21

TERMINATION

- Lemma: Every step reduces the size of q
 - Proof: By observation on the definition of the reduction relation.

 $(S, \{\text{int}=\text{int}\} \cup q) \rightarrow (S, q)$

 $(S, \{s_{11} \rightarrow s_{12} = s_{21} \rightarrow s_{22}\} \cup q) \rightarrow$
 $(S, \{s_{11} = s_{21}, s_{12} = s_{22}\} \cup q)$

 $(S, \{\text{bool}=\text{bool}\} \cup q) \rightarrow (S, q)$

----- (a not in FV(s))
 $(S, \{a=s\} \cup q) \rightarrow$
 $([a=s] \circ S, q[s/a])$

 $(S, \{a=a\} \cup q) \rightarrow (S, q)$

----- (a not in FV(s))
 $(S, \{s=a\} \cup q) \rightarrow$
 $([a=s] \circ S, q[s/a])$

CORRECTNESS

- we know the algorithm terminates
- we want to prove that a series of steps:

$(I, q1) \rightarrow (S2, q2) \rightarrow (S3, q3) \rightarrow \dots \rightarrow (S, \{\})$

solves the initial constraints $q1$


- We'll do that by induction on the length of the sequence, but we'll need to define the **invariants** that are preserved from step to step



COMPLETE SOLUTIONS

- A **complete solution** for (S, q) is a substitution T such that
 1. $T \leq S$
 2. $T \models q$
 - intuition: T extends S and solves q
- A **principal solution** T for (S, q) is complete for (S, q) and
 3. for all T' such that 1. and 2. hold, $T' \leq T$
 - intuition: T is the most general solution (it's the least restrictive)

PROPERTIES OF SOLUTIONS

- Lemma 1: Every final state $(S, \{\})$ has a complete and principal solution, which is S .
- To show that S is a complete solution:
 - $S \leq S$
 - $S \models \{\}$  every substitution is a solution to the empty set of constraints
- Proof: by induction on the length of the unification sequence.
 - Case 0 steps: $S \models \{\}$ is always true for any S , including I . $S \leq I$ for any S .
 - Hypothesis: for k steps from (S', q) , final state $(S, \{\})$ has a complete solution S , i.e. $S \leq S'$, $S \models q$.

- Case k+1 steps:
 - There are 6 subcases, one for each unification rule.
 - Cases int, bool, fun and equal are trivial since S' remains the same after the first step, then remaining k steps is true due to hypothesis.
 - Case (u-var1) and (u-var2):
 - if $([a=s] \circ S, q[s/a])$ has a final solution, i.e. $S \models q[s/a]$ (by IH)
 - then $[a=s] \circ S \models \{a=s\} \cup q$ (proved)

 $(S, \{int=int\} \cup q) \rightarrow (S, q)$

 $(S, \{s11 \rightarrow s12 = s21 \rightarrow s22\} \cup q) \rightarrow$
 $(S, \{s11 = s21, s12 = s22\} \cup q)$

 $(S, \{bool=bool\} \cup q) \rightarrow (S, q)$

----- (a not in FV(s))
 $(S, \{a=s\} \cup q) \rightarrow ([a=s] \circ S, q[s/a])$

 $(S, \{a=a\} \cup q) \rightarrow (S, q)$

----- (a not in FV(s))
 $(S, \{s=a\} \cup q) \rightarrow ([a=s] \circ S, q[s/a])$

PROPERTIES OF SOLUTIONS

- Lemma 2: No stuck state has a complete solution (or any solution at all)
 - it is impossible for a substitution to make the necessary equations equal
 - $\text{int} \neq \text{bool}$
 - $\text{int} \neq t1 \rightarrow t2$
 - ...

PROPERTIES OF SOLUTIONS

○ Lemma 3

- If $(S, q) \rightarrow (S', q')$ then
 - T is complete for (S, q) iff T is complete for (S', q')
 - T is principal for (S, q) iff T is principal for (S', q')
- Proof: by induction on the derivation of unification step \rightarrow

- In the forward direction, this is the preservation theorem for the unification machine!

SUMMARY: UNIFICATION

- By termination, $(I, q) \rightarrow^* (S, q')$ where (S, q') is irreducible. Moreover:

If $q' = \{ \}$ then:

- (S, q') is final (by definition)
- S is a principal solution for q
 - Consider any T such that T is a solution to q .
 - Now notice, S is principal for (S, q') (by lemma 1)
 - S is principal for (I, q) (by lemma 3)
 - Since S is principal for (I, q) , we know $T \leq S$ and therefore S is a principal solution for q .

SUMMARY: UNIFICATION (CONT.)

- ... Moreover:
 - If q' is not $\{\}$ (and $(I, q) \rightarrow^* (S, q')$ where (S, q') is irreducible) then:
 - (S, q') is stuck. Consequently, (S, q') has no complete solution. By lemma 3, even (I, q) has no complete solution and therefore q has no solution at all.

SUMMARY: TYPE INFERENCE

- Type inference algorithm.
 - Given a context G , and untyped term u :
 - Find e, t, q such that $G \vdash u \implies e : t, q$
 - Find principal solution S of q via unification
 - if no solution exists, there is no reconstruction
 - Apply S to e , i.e., our solution is $S(e)$
 - $S(e)$ contains schematic type variables a, b, c , etc. that may be instantiated with any type
 - Since S is principal, $S(e)$ characterizes all reconstructions.

LET POLYMORPHISM

- Generalized from the type inference algorithm
- A.k.a ML-style or Hindley Milner-style polymorphism
- Basis of “generic libraries”:
 - Trees, lists, arrays, hashtables, streams, ...
- `let id = \x. x in`
(`id 25`, `id true`)
 - `id` can't be both `int → int` and `bool → bool`, due to:

$$\frac{G \vdash e1 : t1 \quad G, x:t1 \vdash e2 : t2}{G \vdash \text{let } x=e1 \text{ in } e2 : t2} \quad [\text{t-let}]$$

LET POLYMORPHISM

- Instead:

$$\frac{G \vdash e2[e1/x] : t2 \quad G \vdash e1 : t1}{G \vdash \text{let } x=e1 \text{ in } e2 : t2} \quad [\text{t-letPoly}]$$

- Or using the constraint generation rule:

$G \dashv\vdash u2[u1/x] \implies e2[e1/x] : t2, q2$

$G \dashv\vdash u1 \implies e1 : t1, q1$

 $G \dashv\vdash \text{let } x = u1 \text{ in } u2 \implies \text{let } x = e1 \text{ in } e2: t2, q1 \cup q2$

CAVEAT WITH LET POLYMORPHISM

- If the body (e2) contains many let bindings
- Every occurrence of a let binding in e2 causes a type check of right-hand-side e1
- e1 itself can contain many let binding as well
- Time complexity **exponential** to the size of the expression!
- Practical implementation uses a smarter but equivalent algorithm:
 - Amortized linear time
 - Worse-case still exponential
 - see Pierce Ch. 22.