TYPE INFERENCE (II)
Example (Recap)

fun map (f, l) =
    if null (l) then
        nil
    else
        cons (f (hd l), map (f, tl l))
fun map (f, l) = 
  if null (l) then
    nil
  else
    cons (f (hd l), map (f, tl l))

library functions
argument type is ‘a list

library function
argument type is (‘a * ‘a list)
result type is ‘a list

result type is ‘a
result type is ‘a list
**Step 1: Add Type Schemes**

```ocaml
fun map (f : a, l : b) : c =
    if null (l) then
        nil
    else
        cons (f (hd l), map (f, tl l)))
```
**Step 2: Generate Constraints**

fun map (f : a, l : b) : c = 
if null (l) then 
nil 
else 
cons (f (hd l), map (f, tl l)))
**Step 3: Solve Constraints**

- Constraint solution provides all possible solutions to type scheme annotations on terms

```
final constraints
b = b' list
b = b'' list
b = b''' list
a = a
...
```

```
solution
a = b → c'
b = b' list
c = c' list
```

```
map (f : b' → c' 
x : b' list)
: c' list
=
...
```
**Step 4: Generate Types**

- Generate types from type schemes
  - Option 1: pick an instance of the most general type when we have completed type inference on the entire program
    - map : ((int \to int) \times int list) \to int list
  - Option 2: generate polymorphic types for program parts and continue (polymorphic) type inference
    - map : \forall (a,b) ((a \to b) \times a list) \to b list
**Solving 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 almost all variables $a$)
    - $S(a) = s$ (for some $a$ and some type scheme $s$)
  - $\text{dom}(S) = \text{set of variables s.t. } S(a) \neq a$
Given a substitution $S$, we can define a function $S^*$ from type schemes (as opposed to type variables) to type schemes:

- $S^*(\text{int}) = \text{int}$
- $S^*(s_1 \rightarrow s_2) = S^*(s_1) \rightarrow S^*(s_2)$
- $S^*(a) = S(a)$

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.
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)
SOLVING A CONSTRAINT

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

\[
\begin{align*}
\text{S(s1) = S(s2)} & \quad \text{S \models q} \\
\hline
\text{S \models \{\}} & \quad \text{S \models \{s1 = s2\} \cup q}
\end{align*}
\]

any substitution is a solution for the empty set of constraints

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!)
EXAMPLES

Example 1

- \( q = \{a=\text{int}, \ b=a\} \)
- principal solution \( S \):
  - \( S(a) = S(b) = \text{int} \)
  - \( S(c) = c \) (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 give rise to most general reconstruction of typing information for a term:

- fun f(x:a):a = x
  - is a most general reconstruction

- fun f(x:int):int = x
  - is not
Unification

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

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

\[
\begin{align*}
(S, \{\text{int} = \text{int}\} \cup q) & \rightarrow (S, q) \\
(S, \{\text{bool} = \text{bool}\} \cup q) & \rightarrow (S, q) \\
(S, \{a = a\} \cup q) & \rightarrow (S, q)
\end{align*}
\]
UNIFICATION MACHINE

Functions:

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

Variable definitions

\[(S,\{a=s\} \cup q) \rightarrow ([a=s] \circ S, q[s/a])\]

\[(S,\{s=a\} \cup q) \rightarrow ([a=s] \circ S, q[s/a])\]
**Occurs Check**

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

\[
\begin{align*}
\text{--------------------------------------------} \\
& (a \text{ not in } \text{FV}(s)) \\
& (S,\{s=a\} \cup q) \rightarrow ([a=s] \circ S, q[s/a])
\end{align*}
\]
IRREDUCIBLE STATES

- Recall: final states have the form $(S, \emptyset)$
- Stuck states $(S, q)$ are such that every equation in $q$ has the form:
  - `int = bool`
  - $s_1 \rightarrow s_2 = s$ (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
  - \((S_1, q_1) \rightarrow (S_2, q_2) \rightarrow \ldots\)
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 (i.e., fewer occurrences of `int`, `bool`, or `→`)
    - in other words, $q$ is simpler than $q'$
  - This is a lexicographic ordering
    - 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
**TERMINATION**

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

\[
\begin{align*}
(S, \{\text{int} = \text{int}\} \cup q) & \rightarrow (S, q) \\
(S, \{\text{bool} = \text{bool}\} \cup q) & \rightarrow (S, q) \\
(S, \{a = a\} \cup q) & \rightarrow (S, q)
\end{align*}
\]

\[
\begin{align*}
(S, \{s_{11} \rightarrow s_{12} = s_{21} \rightarrow s_{22}\} \cup q) & \rightarrow (S, \{s_{11} = s_{21}, s_{12} = s_{22}\} \cup q) \\
(a \text{ not in } \text{FV}(s)) & \rightarrow (S, \{a = s\} \cup q) \rightarrow ([a = s] \circ S, q[s/a])
\end{align*}
\]
**Correctness**

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

  \[(I, q_1) \rightarrow (S_2, q_2) \rightarrow (S_3, q_3) \rightarrow \ldots \rightarrow (S, \{\})\]

  solves the initial constraints \(q_1\)

- 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.
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 solution.
    - It is \(S\):
      - \(S <= S\)
      - \(S |= {}\)
  
  every substitution is a solution to the empty set of constraints
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
    - int ≠ bool
    - int ≠ t1 -> 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 \(-\rightarrow\) (exercise!)

- In the forward direction, this is the preservation theorem for the unification machine!
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 \( q \).
... 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.
Type inference algorithm.
- Given a context $G$, and untyped term $u$:
  - Find $e, t, q$ such that $G |- u \Rightarrow e : t, q$
  - 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.