Type Inference

Gul Agha
CS 322
Fall 2001
Type Inference - Example

letrec
  ? even ( ? odd, ? x) =
    if zero? (x) then 0
    else (odd sub1x)
in letrec ? odd ( ? x) =
  if zero? (x) then 0
  else (even odd sub1(x))
in odd (13)
Introduce Type Variables

- A type variable to represent every unknown type
  - Serial number to uniquely identify it
  - Container – vector of length 1
    - a write once variable
      - () nothing is known about this type
      - full the type it is known to be
(define-datatype type type?
  (atomic-type
    (name symbol?))
  (proc-type
    (arg-types (list-of type?))
    (result-type type??))
  (tvar-type
    (serial-number integer?)
    (container vector?)))
Unifying Types

• Check if two types can be made equal
  ➢ if so make them equal
• For type inference
  ➢ walk through program
  ➢ make deductions about types of variables based on use
Inferring Type of Application

• Create a type variable for result type
• Check if:
  • Operator is a procedure
  • Accepts arguments of the same type as the operands
  • Produces a result that is the same type as the application
• *Make deductions in the process*
Inferring Type of Application

(define type-of-application
  (lambda (rator-type actual-types rator rands exp)
    (let ((result-type (fresh-tvar)))
      (check-equal-type!
        rator-type
        (proc-type actual-types result-type)
        exp)
      result-type)))
Example: Conditional Expressions

if $e_0$ then $e_1$ else $e_2$
(type of expression $<<e_0>>\ tenv$)
  = bool
(type of expression $<<e_1>>\ tenv$)
  = (type of expression $<<e_2>>\ tenv$)
  = (type of expression
     $<<\text{if } e_0\text{ then } e_1\text{ else } e_2>>\ tenv$)
Typing Applications Rule

\[
\text{(type of expression } \langle\langle \text{rator} \rangle\rangle \ tenv \text{)} = \left( \text{(type of expression } \langle\langle \text{rand}_1 \rangle\rangle \ tenv \text{)} \right.
\]
\[
\times \ldots \times
\]
\[
\left( \text{(type of expression } \langle\langle \text{rand}_n \rangle\rangle \ tenv \text{)} \right)
\]
\[
\rightarrow
\]
\[
\text{(type of expression } \langle\langle \text{rator rand}_1 \ldots \text{rand}_n \rangle\rangle \ tenv \text{)}
\]

Procedure Definitions Rule

\[
\text{(type of expression}
\langle \langle \text{proc } (x_1 \ldots x_n) \text{ exp} \rangle \text{ tenv} \rangle =
\langle \langle \text{proc } x_1 \text{ tenv_{body}} \rangle \text{ tenv} \rangle
\]
\[
\ast \ldots \ast
\]
\[
\langle \langle \text{proc } x_n \text{ tenv_{body}} \rangle \text{ tenv} \rangle
\rightarrow
\langle \langle \text{exp} \rangle \text{ tenv_{body}} \rangle
\]
Type Inference for Procedures

• Introduce type variable for
  ➢ each bound variable
  ➢ each application

• Write an equation for each compound expression using the abstraction and application rules
Creating Type Variables

\[ \text{proc}(f, x) \ (f + (1, x) \ \text{zero?}(x)) \]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( tf )</td>
</tr>
<tr>
<td>( x )</td>
<td>( tx )</td>
</tr>
<tr>
<td>( f +(1,x) \ \text{zero?} (x) )</td>
<td>( t1 )</td>
</tr>
<tr>
<td>(+ (1,x))</td>
<td>( t2 )</td>
</tr>
<tr>
<td>( \text{zero?} (x) )</td>
<td>( t3 )</td>
</tr>
</tbody>
</table>
Infer Types

• By the procedure rule,
  type of entire expression:
  \( \text{proc}(f, x) \ (f \ +(1,x) \ \text{zero}? \ (x)) \)
  \( = \ (tf \ast tx \rightarrow t1) \)

• write type equations for each application
# Type Equations

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>((f + (1,x) \text{zero?}(x)))</td>
<td>(tf = (t2 * t3 \rightarrow t1))</td>
</tr>
<tr>
<td>+ (1,x)</td>
<td>((\text{int} * \text{int} \rightarrow \text{int}))</td>
</tr>
<tr>
<td></td>
<td>(= (\text{int} * tx \rightarrow t2))</td>
</tr>
<tr>
<td>zero? (x)</td>
<td>((\text{int} \rightarrow \text{bool}) = (tx \rightarrow t3))</td>
</tr>
</tbody>
</table>
## Type Equations

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>((f + (1, x) \text{zero?}(x)))</td>
<td>(tf = (t2 \times t3 \rightarrow t1))</td>
</tr>
</tbody>
</table>
| \(+ (1, x)\)        | \((\text{int} \times \text{int} \rightarrow \text{int})\)  
                          | \(= (\text{int} \times tx \rightarrow t2)\) |
| \text{zero?}(x)     | \((\text{int} \rightarrow \text{bool}) = (tx \rightarrow t3)\) |
Solutions

tf = (int * bool \rightarrow t1)

But we don’t yet have a solution for t1
type of entire expression:

\[ \text{proc}(f, x) \ (f \ +(1,x) \ \text{zero}\? \ (x)) \]
\[ = \ (tf \ * \ tx \ \rightarrow \ t1) \]
\[ = \ ((\text{int} \ * \ \text{bool} \ \rightarrow \ t1) \ * \ \text{int} \ \rightarrow \ t1) \]

*for any choice of* \( t1 \)

polymorphic types with

universal quantification
**Variant Example (Contradiction)**

```
proc(f, x) (f cons(1, x) zero? (x))
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f cons(1, x) zero? (x))</td>
<td>tf = (t2 * t3 → t1)</td>
</tr>
<tr>
<td>cons (1, x)</td>
<td>(int * (list int) → (list int) = (int * tx → t2)</td>
</tr>
<tr>
<td>zero? (x)</td>
<td>(int → bool) = (tx → t3)</td>
</tr>
</tbody>
</table>
How to Solve Equations

• Recursively traverse the type structures to be equated:
  ➢ encounter a type variable containing a type, recur on that type.
  ➢ encounter a type variable that is empty, fill the type variable with the other type.
Checker versus Inferencer

Both recursively walk through the program:

- *Checker* always computes the type of its subexpressions.

- *Inferencer* **notes** how each symbol is used and makes **deductions** about the types:
  - notes are equations.
  - deduction is a walk through the equations making substitutions.
Check if $t_1$ and $t_2$ can be equal:

1. Are $t_1$ and $t_2$ the same value
   - Succeed and return unspecified value.
2. $t_1$ is a type variable, check if $t_1$ and $t_2$ are equal.
3. *Mutatis mutandis* for $t_2$
4. If $t_1$ and $t_2$ are both atomic types:
   ➢ *Do they have the same name?*

5. If $t_1$ and $t_2$ are both procedure types:
   ➢ *Do they have the same number of arguments?*
     - yes → recur on the arguments
     - no → report error

6. Otherwise error!!
Termination

t1 = (int \rightarrow t1)

If t1 is filled in, we get a cycle on the argument:
- Infinite loop!!
- Add occurs check …