ENE4014: Programming Languages

Lecture 16 — Let-Polymorphic Type System

Woosuk Lee
2024 Spring

Motivation

- Our type system is useful but it is not as expressive as we would like it to be. In particular, it does not support *polymorphism*[1]. For example, it rejects the following program:

  ```
  let f = proc (x) x in
    if (f (iszero (0))) then (f 11) else (f 22)
  ```

- Polymorphic functions are widely used in practice, so OCaml supports polymorphism:

  ```
  # let f = fun x -> x in
      if (f (0=0)) then (f 11) else (f 22);;
  - : int = 11
  ```

- Lets extend our type system to the let-polymorphic type system, the ML-style polymorphism.

---

[1] Polymorphism refers to the language mechanisms that allow a single part of a program to be used with different types in different contexts

What went wrong?

```
let f = proc (x) x in
  if (f (iszero (0))) then (f 11) else (f 22)
```

- We assign type $t \to t$ to `f`, generating the constraint that the argument and return types are the same.
- Intuitively, the program can be well typed because the all usages of `f` satisfy the required constraint:
  - In `(f (iszero 0))`, we can assign bool → bool to `f`.
  - In `(f 11)` and `(f 22)`, we can assign int → int to `f`.
- However, our type checking algorithm uses the same type variable t in both cases and generates the spurious constraint that bool $=$ int.
- Any idea to fix this problem?

A Simple Solution

Associate a *different* variable t with each use of `f`. This is easily accomplished by substituting the body of `f` for each occurrence of `f`. For example, convert the program

```
let f = proc (x) x in
  if (f (iszero (0))) then (f 11) else (f 22)
```

into the following before type-checking:

```
if ((proc (x) x) (iszero (0)))
then ((proc (x) x) 11)
else ((proc (x) x) 22)
```

which is accepted by our type system as we can generate different type variables for different copies of the procedure.

Typing Rule

Instead of the ordinary typing rule for `let`:

$$\frac{\Gamma \vdash E_1 : t_1 \qquad [x \mapsto t_1]\Gamma \vdash E_2 : t_2}{\Gamma \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 : t_2}$$

we used the new typing rule:

$$\frac{\Gamma \vdash [x \mapsto E_1]E_2 : t_2}{\Gamma \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 : t_2}$$

Here, $[x \mapsto E_1]E_2$ denotes an expression obtained by replacing each occurrence of x by E_1 in E_2.
The corresponding algorithm for generating type equation:

$$\mathcal{V}(\Gamma, \texttt{let}\ x = e_1\ \texttt{in}\ e_2, t) = \mathcal{V}(\Gamma, [x \mapsto e_1]e_2, t)$$

The ordinary unification algorithm does the rest.

Flaws

This simplistic method has some flaws that need to be addressed before we can use it in practice.

1. Unused definitions are not type-checked, so a program like

   ```
   let x = <unsafe code> in 5
   ```

   will pass the type-checker. (This can be easily fixed. See Exercise 1)

2. The method is not efficient if the body of `let` contains many occurrences of the bound variables:

   ```
   let a = <complex code> in
     let b = a + a in
       let c = b + b in
         let d = c + c in
           ...
   ```

   The typing rule can cause the type-checker to perform an amount of work that is exponential in the size of the original code.

Exercise 1

Fix the typing rule and $\mathcal{V}$ to repair the first problem.
We can fix the problem by adding a premise to the typing rule:

$$\frac{\Gamma \vdash [x \mapsto E_1]E_2 : t_2 \qquad \Gamma \vdash E_1 : t_1}{\Gamma \vdash \texttt{let}\ x = E_1\ \texttt{in}\ E_2 : t_2}$$

and a corresponding premise to the algorithm:

$$\mathcal{V}(\Gamma, \texttt{let}\ x = e_1\ \texttt{in}\ e_2, t) = \mathcal{V}(\Gamma, e_1, \alpha) \wedge \mathcal{V}(\Gamma, [x \mapsto e_1]e_2, t)\ (\text{new}\ \alpha)$$

Let-Polymorphic Type Checking Algorithm

To avoid the re-computation, practical implementations of languages with let-polymorphism use a more clever algorithm. In outline, the type-checking of

$$\texttt{let } x = e_1 \texttt{ in } e_2$$

proceeds as follows:

- We find the most general type t of e_1 by running the ordinary type-checking algorithm (i.e., compute $\mathcal{U}(\mathcal{V}(\Gamma, e_1, t))$ where Γ is the type environment embracing e_1).
- We *generalize* any variables remaining in the type, obtaining the *type scheme* $\forall \alpha_1 \ldots \alpha_n . t$, where $\alpha_1 \ldots \alpha_n$ appear in t.
- We extend the type environment to record the type scheme for the bound variable x, and start type-checking e_2
- Each time we encounter an occurrence of x, we generate fresh type variables $\beta_1 \ldots \beta_n$ and use them to instantiate the type scheme.

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{f}_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ \mathit{true})}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
$t_f = \forall t_x.\ t_x \rightarrow \text{int}$	
$t_1 = \text{int}$	
$t_2 = \text{int}$	
$t_3 = \text{int}$	
$t_f = \text{int} \rightarrow t_2$	
$t_f = \text{bool} \rightarrow t_3$	
$t_0 = t_1$	

$$\mathcal{U}(\mathcal{V}(\emptyset, \texttt{proc}\ (x)\ 1, t_4)) = t_x \rightarrow \text{int}.$$

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{f}_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ \mathit{true})}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \rightarrow \text{int}$
	$t_1 = \text{int}$
	$t_2 = \text{int}$
	$t_3 = \text{int}$
$t_f = \text{int} \rightarrow t_2$	
$t_f = \text{bool} \rightarrow t_3$	
$t_0 = t_1$	

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{f}_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ true)}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to \text{int}$
	$t_1 = \text{int}$
	$t_2 = \text{int}$
	$t_3 = \text{int}$
$\beta_1 \to \text{int} = \text{int} \to t_2$	
$t_f = \text{bool} \to t_3$	
$t_0 = t_1$	

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{f}_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ true)}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \rightarrow \mathsf{int}$
	$t_1 = \mathsf{int}$
	$t_2 = \mathsf{int}$
	$t_3 = \mathsf{int}$
$\beta_1 = \mathsf{int}$	
$\mathsf{int} = t_2$	
$t_f = \mathsf{bool} \rightarrow t_3$	
$t_0 = t_1$	

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{f}_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ true)}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to \text{int}$
	$t_1 = \text{int}$
	$t_2 = \text{int}$
	$t_3 = \text{int}$
	$\beta_1 = \text{int}$
	$t_2 = \text{int}$
$t_f = \text{bool} \to t_3$	
$t_0 = t_1$	

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{f}_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ \mathit{true})}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \rightarrow \text{int}$
	$t_1 = \text{int}$
	$t_2 = \text{int}$
	$t_3 = \text{int}$
	$\beta_1 = \text{int}$
	$t_2 = \text{int}$
$\beta_2 \rightarrow \text{int} = \text{bool} \rightarrow t_3$	
$t_0 = t_1$	

Example 1

$$\underbrace{\texttt{let}\ (\underbrace{\ f\ }_{t_f}) = \underbrace{\texttt{proc}\ (\underbrace{\ x\ }_{t_x})\ 1}_{t_4}\ \texttt{in}\ \underbrace{\underbrace{(f\ 1)}_{t_2} + \underbrace{(f\ true)}_{t_3}}_{t_1}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to$ int
	$t_1 =$ int
	$t_2 =$ int
	$t_3 =$ int
	$\beta_1 =$ int
	$t_2 =$ int
	$\beta_2 =$ bool
	$t_3 =$ int
	$t_0 =$ int

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
$t_f = \forall t_x.\ t_x \to t_x$	
$t_2 = \text{bool}$	
$t_3 = \text{int}$	
$t_4 = \text{int} \to t_3$	
$t_f = \text{bool} \to t_2$	
$t_f = t_f \to t_4$	
$t_0 = t_3$	

$$\mathcal{U}(\mathcal{V}(\emptyset, \texttt{proc}\ (x)\ x, t_1)) = t_x \to t_x.$$

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \to \text{int}$
$t_f = \text{bool} \to t_2$	
$t_f = t_f \to t_4$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ \mathit{true})}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \to \text{int}$
$\beta_1 \to \beta_1 = \text{bool} \to \text{bool}$	
$t_f = t_f \to t_4$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \to \text{int}$
	$\beta_1 = \text{bool}$
$t_f = t_f \to t_4$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \rightarrow t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \rightarrow \text{int}$
	$\beta_1 = \text{bool}$
$\beta_2 \rightarrow \beta_2 = (\beta_3 \rightarrow \beta_3) \rightarrow t_4$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \rightarrow t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \rightarrow \text{int}$
	$\beta_1 = \text{bool}$
$\beta_2 = \beta_3 \rightarrow \beta_3$	
$\beta_2 = \text{int} \rightarrow \text{int}$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\, t_x \to t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \to \text{int}$
	$\beta_1 = \text{bool}$
	$\beta_2 = \beta_3 \to \beta_3$
$\beta_3 \to \beta_3 = \text{int} \to \text{int}$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \to \text{int}$
	$\beta_1 = \text{bool}$
	$\beta_2 = \beta_3 \to \beta_3$
$\beta_3 = \text{int}$	
$t_0 = t_3$	

Example 2

$$\texttt{let}\ \underbrace{f}_{t_f} = \underbrace{\texttt{proc}\ (\underbrace{x}_{t_x})\ x}_{t_1}\ \texttt{in}\ \underbrace{\texttt{if}\ \underbrace{(f\ true)}_{t_2}\ \texttt{then}\ 1\ \texttt{else}\ \underbrace{(\underbrace{(f\ f)}_{t_4}\ 2)}_{t_3}}_{t_0}$$

Equations	Substitution
	$t_f = \forall t_x.\ t_x \to t_x$
	$t_2 = \text{bool}$
	$t_3 = \text{int}$
	$t_4 = \text{int} \to \text{int}$
	$\beta_1 = \text{bool}$
	$\beta_2 = \text{int} \to \text{int}$
	$\beta_3 = \text{int}$
	$t_0 = \text{int}$

Summary

- We extended our type system (called *simple type system*) to *let-polymorphic type system*, the core of ML type system.
- The extension is conservative:

$$\boldsymbol{\Gamma \vdash_{simple} E : T \implies \Gamma \vdash_{poly} E : T}$$

Let-polymorphic type system accepts all programs acceptable by the simple type system.