HW8 – Type Inference and Polymorphism

CS 476, Fall 2019
Due Nov. 25 at 2 PM

1 Instructions

This assignment is to be completed by hand (or in LaTeX if you know how to use it). Submit your answers as a PDF file via Gradescope. If you don’t have easy access to a scanner, you can use the one in SEO 1120, the main CS office – the staff will be happy to help you. As always, please don’t hesitate to ask for help on Piazza (https://piazza.com/class/jushbmjmi113yb).

2 Type Inference Rules

The rules for type inference for a simple OCaml-like language are as follows.

\[
\begin{align*}
&\Gamma \vdash n : \text{int} \mid \{} \quad (n \text{ is an integer literal}) \\
&\Gamma \vdash b : \text{bool} \mid \{} \quad (b \text{ is a boolean literal}) \\
&\Gamma \vdash x : \tau \mid \{} \\
&\Gamma \vdash e_1 : \tau_1 \mid C_1 \quad \Gamma \vdash e_2 : \tau_2 \mid C_2 \\
&\Gamma \vdash e_1 \land e_2 : \text{int} \mid \{\tau_1 = \text{int}, \tau_2 = \text{int}\} \cup C_1 \cup C_2 \\
&\Gamma \vdash e_1 \land e_2 : \text{bool} \mid \{\tau_1 = \text{bool}, \tau_2 = \text{bool}\} \cup C_1 \cup C_2 \\
&\Gamma \vdash x_1 \mapsto e \mid \tau_2 \mid C \quad \tau_1 \text{ fresh} \\
&\Gamma \vdash \text{fun } x_1 \mapsto e : \tau_1 \rightarrow \tau_2 \mid C \\
&\Gamma \vdash \text{fun } (x_1 \mapsto e_1, x_2 \mapsto e_2) : \tau \mid \{\tau_1 = \tau_2 \rightarrow \tau\} \cup C_1 \cup C_2 \quad \tau \text{ fresh} \\
&\Gamma \vdash x = e : \tau \mid \{} \\
&\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau \mid C_1 \cup C_2 \\
&\Gamma \vdash e_1 : \tau_1 \mid C_1 \quad \text{fv}(\tau_1) - \text{fv}(\Gamma) = a_1, \ldots, a_n \\
&\Gamma \vdash e_2 : \tau \mid C_2 \\
&\Gamma \vdash x_1 \mapsto e_1, \ldots, x_n \mapsto e_n : \tau_1 \mid \{} \\
&\Gamma \vdash x : [a_1 \mapsto b_1, \ldots, a_n \mapsto b_n] \tau \mid \{} \\
\end{align*}
\]

where \([a_1 \mapsto b_1, \ldots, a_n \mapsto b_n] \tau\) means “replace the bound variables \([a_1, \ldots, a_n]\) with the fresh type variables \([b_1, \ldots, b_n]\).”

When we add polymorphism, we add the following additional rules.

\[
\begin{align*}
&\Gamma \vdash e_1 : \tau_1 \mid C_1 \\
&\text{fv}(\tau_1) - \text{fv}(\Gamma) = \alpha_1, \ldots, \alpha_n \\
&\Gamma \vdash \forall \alpha_1 \ldots \alpha_n. \tau_1 \mid e_2 : \tau \mid C_2 \\
&\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : \tau \mid C_1 \cup C_2 \\
&\Gamma \vdash x : [\alpha_1 \mapsto b_1, \ldots, \alpha_n \mapsto b_n] \tau \mid \{} \\
&\end{align*}
\]
3 Problems

1. (3 points) What is the type of \( \text{fun } x \rightarrow \text{fun } y : \text{int } \rightarrow (x \, y) + 1 \)?

2. (7 points) Write the proof tree for the judgment
   \[ \{ \} \vdash \text{fun } x \rightarrow \text{fun } y : \text{int } \rightarrow (x \, y) + 1 : \tau \mid C, \] where \( \tau \) and \( C \) start as blank and are filled in as you complete the proof tree.
3. Consider the program

\[ \text{let } g = (\text{fun } x \to (x = x)) \text{ in } g \ 5 \&\& g \ \text{true}. \]

(a) (5 points) Write a proof tree for the judgment \( \{ \} \vdash (\text{fun } x \to (x = x)) : \tau \mid C \). Again, \( \tau \) and \( C \) should start blank and be filled in as you complete the proof tree.

(b) (3 points) Given your answer to part a, what is the polymorphic type (polytype) of \( g \) in the program?
(c) (7 points) Write the proof tree for the judgment

\[
\{\} \vdash \text{let } g = (\lambda x. x = x) \text{ in } g \ 5 \ \&\& \ g \ \text{true} : \tau \ | \ C.
\]

Once again, \(\tau\) and \(C\) should start blank and be filled in as you complete the proof tree. You can use \(P_1\) to stand for your proof tree from part a.

Hint 1: In the type context for the body of the let (i.e., \(g \ 5 \ \&\& \ g \ \text{true}\)), the type of \(g\) should be your answer to part b.

Hint 2: When you reach a variable, make sure to use the polymorphic variable rule from the end of section 2.