Part I: Written Problems

Problem 1: (25 points).

Give full type derivations for the following using the LFP+ typing rules given in class (and page 3).

(a) \( \vdash 6 + (\text{if } \ell > 0 \text{ then } 5 \text{ else } 11) : \text{int} \)
(b) \( x : \sigma, f : (\sigma \rightarrow \tau) \vdash (f x) : \tau \)
(c) \( \vdash \lambda x. \lambda y. (y x) : \sigma \rightarrow (\sigma \rightarrow \tau) \rightarrow \tau \)
(d) \( \vdash \lambda y. \lambda x. (y x) : (\text{bool} \rightarrow \text{int}) \rightarrow \text{bool} \rightarrow \text{int} \)
(e) \( \vdash \lambda p. \lambda f. \lambda x. (p (f x)) : (\text{int} \rightarrow \text{bool}) \rightarrow (\text{int} \rightarrow \text{int}) \rightarrow \text{int} \rightarrow \text{bool} \)

Problem 2: (32 points).

(a) Come up with a workable typing rule for the \texttt{let} construct. I.e., something along the lines of
\[
\begin{align*}
?? & \vdash e_1 : ?? \\
?? & \vdash e_2 : ?? \\
\Gamma & \vdash \text{let } x = e_1 \text{ in } e_2 : \tau
\end{align*}
\]
(b) Give a full type derivation for:
\( x : \text{int} \vdash \text{let } p = \lambda y. (x + y) \text{ in } (\text{let } x = 3 \text{ in } (p x)) : \text{int} \)
(c) Give a full type derivation for:
\( y : \text{int} \vdash \text{let } p = \lambda x. \lambda y. (x > y) \text{ in } (\text{let } x = y \text{ in } (p x)) : \text{int} \rightarrow \text{bool} \)
(d) On page 66 Pitts [Pit02] defines the \texttt{letrec} construct. Come up with a workable type for the \texttt{letrec}.

Part II: Programming Problems

Read the Input and Output chapter of [Lip11] before working these problems. For testing for these problems all you need to do is run around four sample runs with a boundary cases included (e.g., an empty input list for problem 3).

In the following the text in \textit{green italics} is what is typed by the user.

Problem 3: (15 points = 10 pts correctness + 5 pts testing).

Write a function
\[
\text{showHisto} :: \text{IO } ()
\]
that repeatedly reads Ints (one per line) until finding a negative value and then outputs a histogram of the codeInts values in the order they were entered. (You may find the \texttt{replicate} function from \texttt{Data.List} handy.) For example:

\[
\begin{align*}
*\text{Main}> & \text{showHisto} \\
10 & \\
3 & \\
7 & \\
-1 & \\
********** & \\
*** & \\
*******
\end{align*}
\]

Suggestion: Break down the problem with helper functions.
Problem 4: (10 points = 6 pts correctness + 4 pts testing).

Write a function

\[
\text{ask} :: \text{String} \rightarrow \text{IO Char}
\]

that writes its string argument as a user prompt, reads the entire next line of input, and returns (as in return) the character of the first line of that input. For example,

```
*Main> ask "Do you like hakarl?"
Do you like hakarl? no!!!
'n'
```

Note: I don’t think anyone likes hákarl.

Problem 5: (18 points = 12 pts correctness + 6 pts testing).

Write a function

\[
\text{game} :: \text{IO ()}
\]

that plays binary-search number-guessing game. Here is a sample play.

```
*Main> game
Think of a number between 1 and 100
Is your number 50? lower
Is your number 25? higher
Is your number 37? higher
Is your number 43? lower
Is your number 40? higher
Is your number 41? higher
Is your number 42? yes
I win!
```

If the user cheats (e.g., answers lower for 42 in the previous interaction), you may have your program say something snarky.

Big, obvious hints:

- Helper functions can be handy.
- The functions getline, putStrLn, and ask (from the previous problem) may also prove handy.
- Since this is the “binary-search number-guessing game”, binary search may be involved.
- The strings “lower”, “higher”, and “yes” all begin with different letters.

Administrivia

This assignment is a solo effort: NO TEAMS!

- Turn in problems 1 and 2 in the CIS 352 submissions box. If you trade ideas with another student, document this in your cover sheet.
- For the remaining problems, turn them in via Blackboard. Include:
  (i) the source files,
  (ii) the transcripts of test runs, and
  (iii) your cover sheet.

References


[Pit02] Andrew Pitts, Lecture notes on semantics of programming languages: For part
IB of the Cambridge CS tripos, Tech. report, University of Cambridge, 2002,
LFP⁺ typing rules

- **int:** \( \Gamma \vdash n : \text{int} \quad (n \in \mathbb{Z}) \)
- **bool:** \( \Gamma \vdash b : \text{bool} \quad (b \in \mathbb{B}) \)
- **loc:** \( \Gamma \vdash \ell : \text{loc} \quad (\ell \in \mathbb{L}) \)
- **iop:** \( \Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_1 \ iop \ e_2 : \text{int} \)
- **cop:** \( \Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad \Gamma \vdash e_1 \ cop \ e_2 : \text{bool} \)

\( iop \in \{+,-,*\} \quad \text{cop} \in \{=,\neq,<,>,\leq,\geq\} \)

\[ \begin{align*}
\Gamma \vdash e_0 : \text{bool} & \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau \\
\Gamma \vdash e : \text{loc} & \quad \Gamma \vdash e : \text{int} \\
\Gamma \vdash e_1 : \text{loc} \quad \Gamma \vdash e_2 : \text{int} & \quad \Gamma \vdash e_1 \leftarrow e_2 : \text{cmd} \\
\Gamma \vdash e_1 : \text{cmd} \quad \Gamma \vdash e_2 : \text{cmd} & \quad \Gamma \vdash e_1 \cop e_2 : \text{cmd} \\
\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{cmd} & \quad \Gamma \vdash \text{while } e_1 \text{ do } e_2 : \text{cmd} \\
\end{align*} \]

\[ \begin{align*}
\Gamma \vdash \text{skip} : \text{cmd} & \quad \Gamma \vdash \text{skip} \\
\Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau & \quad \Gamma \vdash \text{if } e_0 \text{ then } e_1 \text{ else } e_2 : \tau \\
\Gamma \vdash e : \text{int} & \quad \Gamma \vdash e : \text{int} \\
\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} & \quad \Gamma \vdash e_1 \cop e_2 : \text{bool} \\
\end{align*} \]

Notes:
- We use \( \sigma \) and \( \tau \) as meta-variables over types.
- \( \Gamma, x : \tau \equiv \Gamma \cup \{x : \tau\} \)
- \( \Gamma \vdash e : \tau \equiv \emptyset \vdash e : \tau \).

A sample derivation

\[ \begin{align*}
\text{var:} & \quad x : \text{int}, y : \text{int} \vdash x : \text{int} \\
+ : & \quad x : \text{int}, y : \text{int} \vdash x + y : \text{int} \\
\ast : & \quad x : \text{int}, y : \text{int} \vdash (x \ast y) : \text{int} \\
\text{fn:} & \quad x : \text{int}, y : \text{int} \vdash \lambda y. (x + (3 \ast y)) : \text{int} \\
\text{fn:} & \quad \lambda x. \lambda y. (x + (3 \ast y)) : \text{int} \rightarrow \text{int} \\
\text{fn:} & \quad \lambda x. \lambda y. (x + (3 \ast y)) : \text{int} \rightarrow \text{int} \\
\text{var:} & \quad x, y : \text{int} \vdash x + (3 \ast y) : \text{int} \\
\text{var:} & \quad x, y : \text{int} \vdash x + (3 \ast y) : \text{int} \\
\end{align*} \]