Problem 1 (Fun, but too much for a quiz)

BACKGROUND. Fractran is a Turing-complete programming language invented by the mathematician John Conway. (See: \url{http://en.wikipedia.org/wiki/FRACTRAN}) A Fractran program is an ordered list of positive fractions together with an initial positive integer input \(n\). The program is run by updating the integer \(n\) as follows:

1. For the first fraction \(f\) in the list for which \(n \cdot f\) is an integer, replace \(n\) by \(n \cdot f\).
2. Repeat rule 1 until no fraction in the list produces an integer when multiplied by \(n\), then halt.

For example, the Fractran program

\[
\left\lfloor \frac{455}{33}, \frac{11}{13}, \frac{1}{11}, \frac{3}{7}, \frac{11}{2}, \frac{1}{3} \right\rfloor
\]

turns out the map any input of the form \(2^a 3^b\) to the output \(5^a 2^b\). E.g.,

\[
2^2 3^2 = 36 \sim 36 \cdot \frac{11}{2} = 198 \sim 198 \cdot \frac{455}{33} = 2730 \sim \\
\cdots \sim 5625 \cdot \frac{1}{3} = 1875 \sim 1875 \cdot \frac{1}{3} = 625 = 5^2 2^2.
\]

YOUR PROBLEM. Represent fractions as pairs of Integers and Fractran programs as lists of pairs of

Integers. So the example Fractran program above would be

\[
[(455, 33), (11, 13), (1, 11), (3, 7), (11, 2), (1, 3)].
\]

Write a Haskell functions

\[
\text{run} :: [(\text{Integer}, \text{Integer})] \rightarrow \text{Integer} \rightarrow \text{Integer}
\]

such that \((\text{run} \ p \ n)\) is the result of running Fractran program \(p\) on input \(n\). (You may assume each pair \((a, b)\) in the program list has \(a, b > 0\).) Recall that \((\text{quotRem} \ a \ b)\) returns the pair \((q, r)\) where \(q\) is the quotient of \(a \div b\) and \(r\) is the remainder of \(a \div b\).

Question 2. Consider:

\[
\rho_0 = \{ a \mapsto 10 \} \\
e = \text{let } g = \lambda z. (z - a) \\
\text{in } \text{let } h = \lambda a. (p \cdot (a + 10)) \\
\text{in } \text{let } a = 1 \\
\text{in } \text{let } \ell_1 : e
\]

What is the value of \(e \rho_0\) with call-by-value under

(a) lexical scoping?
(b) dynamic scoping?

Question 3. Consider:

\[
\rho_0 = \{ a \mapsto 4 \} \\
e = \text{let } g = \lambda z. (z - a) \\
\text{in let } a = 7 \\
\text{in } (g \ 10)
\]

What is printed by \(e \rho_0\) under lexical scoping with

(a) call-by-value?
(b) call-by-name?

Question 4. Assume all locations start with contents

0. Consider:

\[
\ell_0 = 0 \\
e = \text{let } f = \lambda x. (\ell_0 : = !\ell_0 + 1; \\
\text{return} (x \cdot 10 + !\ell_0)) \\
\text{in let } g = \lambda y. (y + y) \\
\text{in } \{ \ell_1 : = (g \ 5); \\
\text{print} (!\ell_0, \ell_1) \}
\]

What is printed by \(e \rho_0\) under lexical scoping with

(a) call-by-value?
(b) call-by-name?

Question 5. Assume all locations start with contents

0. Consider:

\[
\ell_0 = 0 \\
e = \text{let } f = \lambda y. (\ell_1 : = y; \\
\ell_0 : = 20; \\
\ell_1 : = !\ell_1 + y; \\
\ell_0 : = 300; \\
\text{return} y
\]

What does the above print under lexical scoping with

(a) call-by-value?
(b) call-by-name?
An answer for 1

trace :: Integer -> [Frac] -> [Integer]
-- trace inp program = the sequence of
-- states of the program's run
trace m prg = step m prg
where
  step m [] = [m]
  step m ((a,b):fs)
    | r==0 = m:(step q prg)
    | otherwise = step m fs
      where
        (q,r) = quotRem (a*m) b

run :: Integer -> [Frac] -> Integer
-- run inp program = the final state (and
-- output) of the program,
run m prg = last(run m prg)

An answer for Question 2a

<table>
<thead>
<tr>
<th>ENVIRONMENT</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ₀:</td>
<td>a → 4</td>
</tr>
<tr>
<td>ρ₁:</td>
<td>g → λz.(z-a) ρ₀</td>
</tr>
<tr>
<td>ρ₂:</td>
<td>a → 7</td>
</tr>
<tr>
<td>ρ₃:</td>
<td>z → 10</td>
</tr>
</tbody>
</table>

An answer for Quiz 5a, Question 2b

<table>
<thead>
<tr>
<th>ENVIRONMENT</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ₀:</td>
<td>a → 4</td>
</tr>
<tr>
<td>ρ₁:</td>
<td>g → λz.(z-a)</td>
</tr>
<tr>
<td>ρ₂:</td>
<td>a → 7</td>
</tr>
<tr>
<td>ρ₃:</td>
<td>z → 10</td>
</tr>
</tbody>
</table>

An answer for Question 3(a)

<table>
<thead>
<tr>
<th>ENVIRONMENT</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ₀:</td>
<td>a → 10</td>
</tr>
<tr>
<td>ρ₁:</td>
<td>p → λb.(b-a) ρ₀</td>
</tr>
<tr>
<td>ρ₂:</td>
<td>h → λa.(p(a*10)) ρ₁</td>
</tr>
<tr>
<td>ρ₃:</td>
<td>a → 1</td>
</tr>
<tr>
<td>ρ₄:</td>
<td>a → 4</td>
</tr>
<tr>
<td>ρ₅:</td>
<td>b → 40</td>
</tr>
</tbody>
</table>

An answer for Quiz 5a, Question 2b

<table>
<thead>
<tr>
<th>ENVIRONMENT</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ₀:</td>
<td>a → 4</td>
</tr>
<tr>
<td>ρ₁:</td>
<td>g → λz.(z-a)</td>
</tr>
<tr>
<td>ρ₂:</td>
<td>a → 7</td>
</tr>
<tr>
<td>ρ₃:</td>
<td>z → 10</td>
</tr>
</tbody>
</table>

An answer for Question 3(b)

<table>
<thead>
<tr>
<th>ENVIRONMENT</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ₀:</td>
<td>a → 10</td>
</tr>
<tr>
<td>ρ₁:</td>
<td>p → λb.(b-a)</td>
</tr>
<tr>
<td>ρ₂:</td>
<td>h → λa.(p(a*10))</td>
</tr>
<tr>
<td>ρ₃:</td>
<td>a → 1</td>
</tr>
<tr>
<td>ρ₄:</td>
<td>a → 4</td>
</tr>
<tr>
<td>ρ₅:</td>
<td>b → 40</td>
</tr>
</tbody>
</table>

The call to h
ρ₄: a → 4  (p(a*10))
ρ₄(a)*10 = 40
The call to p
ρ₅: b → 40  (b-a)
= ρ₅(b) - ρ₄(a)
= 36, the answer
An answer for Question 4(a)

\[
\begin{align*}
\rho_0 &: \emptyset \\
\rho_1 &: f \rightarrow \lambda x. \{ \ldots \} \rho_0 \\
\rho_2 &: g \rightarrow (\lambda y. (y + y)) \rho_1 \\
\end{align*}
\]

The call to \( f \) \( \rho_4 \)
\[
\begin{align*}
\ell_0 &: = \ldots \\
x &: = 5 \\
x \times 10 + \ell_0 &= 51 \\
\end{align*}
\]
The to \( g \) \( \rho_5 \)
\[
\begin{align*}
\ell_1 &: = \ldots \\
y &: = 51 \\
(y + y) &= 102
\end{align*}
\]
Thus: \( 1, 102 \) is printed.

An answer for Question 4(b)

\[
\begin{align*}
\rho_0 &: \emptyset \\
\rho_1 &: f \rightarrow \lambda x. \{ \ldots \} \rho_0 \\
\rho_2 &: g \rightarrow (\lambda y. (y + y)) \rho_1 \\
\end{align*}
\]

The call to \( f \) \( \rho_4 \)
\[
\begin{align*}
\ell_0 &: = \ldots \\
x &: = 5 \\
x \times 10 + \ell_0 &= 51 \\
\end{align*}
\]
The to \( g \) \( \rho_5 \)
\[
\begin{align*}
\ell_1 &: = \ldots \\
y &: = 51 \\
(f \ 5) \rho_4 &= 102 \\
\end{align*}
\]
Thus: \( 2, 103 \) is printed.

An answer for Question 5(a)

\[
\begin{align*}
\rho_0 &: \emptyset \\
\rho_1 &: f \rightarrow \lambda y. \{ \ldots \} \rho_0 \\
\rho_2 &: y \rightarrow 10 \rightarrow \rho_0 \\
\end{align*}
\]

The call to \( p \) \( \rho_2 \)
\[
\begin{align*}
\ell_0 &: = y; \ldots \\
\ell_0 &= 1 \\
10 \times \ell_0 &= 10 \\
\end{align*}
\]
So, in the call to \( p \), the body is equivalent to:
\[
\begin{align*}
\ell_1 &: = 10; \quad \ell_0 &: = 20; \quad \ell_1 &: = !\ell_1 + 10; \\
\ell_0 &: = 300; \quad \text{return} \ 10
\end{align*}
\]
Thus: \( 300, 20, 10 \) is printed.

An answer for Question 5(a)

\[
\begin{align*}
\rho_0 &: \emptyset \\
\rho_1 &: f \rightarrow \lambda y. \{ \ldots \} \rho_0 \\
\rho_2 &: y \rightarrow (10 + \ell_0) \rightarrow \rho_0 \\
\end{align*}
\]

The call to \( p \) \( \rho_2 \)
\[
\begin{align*}
\ell_0 &: = y; \ldots \\
\ell_0 &= 1 \\
10 + \ell_0 &= 10 \\
\end{align*}
\]
So, in the call to \( p \), the body is equivalent to:
\[
\begin{align*}
\ell_1 &: = 10 + \ell_0; \quad \ell_0 &: = 20; \quad \ell_1 &: = !\ell_1 + 10 + \ell_0; \\
\ell_0 &: = 300; \quad \text{return} \ 10 + \ell_0
\end{align*}
\]
Thus: \( 300, 210, 3000 \) is printed.
Reference Page

The version of LFP we’ll be working with has two new commands:
1. The \texttt{return} command works pretty much as it does in C.

\[
\text{Return: } \frac{\rho \vdash (e, s) \Downarrow (v, s')} {\rho \vdash \texttt{return } e, s \Downarrow (v, s')}
\]

2. The \texttt{print} command does what you expect.

\[
\text{Print: } \frac { \{ \rho \vdash (e_i, s_i) \Downarrow (v_i, s_i+1) \}_{i=1,...,n} } {\rho \vdash (\texttt{print } (e_1, \ldots, e_n, s_1) \Downarrow (\texttt{skip}, s_{n+1})} \quad (\text{the values of } e_i, \\
\text{are printed})
\]

The big-step operational semantics rules for (i) Call-by-value, lexical-scoping, (ii) Call-by-value, dynamic-scoping, (iii) Call-by-name, lexical-scoping, and (iv) Call-by-name, dynamic-scoping are given below. I’ve put closures in boxes. Also, the hat in \(\hat{\rho}\) is just a distinctive decoration.

**Call-by-value, lexical-scoping**

\[
\rho \vdash (e_1, s) \Downarrow_V \langle \lambda x.e_1 \rangle s'
\]

\[
\rho \vdash (e_2, s') \Downarrow_V \langle v_2, s'' \rangle
\]

\[
\rho \vdash ((e_1 e_2), s) \Downarrow_V \langle v, s''' \rangle
\]

**Call-by-value, dynamic-scoping**

\[
\rho \vdash (e_1, s) \Downarrow_V \langle \lambda x.e_1 \rangle s'
\]

\[
\rho \vdash (e_2, s') \Downarrow_V \langle v_2, s'' \rangle
\]

\[
\rho \vdash (e_1 e_2), s \Downarrow_V \langle v, s''' \rangle
\]

**Call-by-name, lexical-scoping**

\[
\rho \vdash (e_1, s) \Downarrow_N \langle \lambda x.e_1 \rangle s'
\]

\[
\rho \vdash (e_2, s') \Downarrow_N \langle v, s'' \rangle
\]

**Call-by-name, dynamic-scoping**

\[
\rho \vdash (e_1, s) \Downarrow_N \langle \lambda x.e_1 \rangle s'
\]

\[
\rho \vdash (e_2, s') \Downarrow_N \langle v, s'' \rangle
\]

\[
\rho \vdash (e_1 e_2), s \Downarrow_N \langle v, s''' \rangle
\]

\[
\rho \vdash (e, s) \Downarrow_N \langle v, s' \rangle \quad (e = \text{lookup}(\rho, x))
\]

\[
\rho \vdash (x, s) \Downarrow_N \langle v, s' \rangle \quad (e = \text{lookup}(\rho, x))
\]

\[
\rho \vdash (\lambda x.e), s \Downarrow_N \langle \lambda x.e \rangle s
\]