Midterm 2 (Practice Exam)
CMSC 430
Introduction to Compilers
Spring 2015
April 21, 2015

Instructions

This exam contains 7 pages, including this one. Make sure you have all the pages. Write your name on the top of this page before starting the exam.

Write your answers on the exam sheets. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.

If you have a question, raise your hand. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn’t need to do this at all, so be careful when making assumptions.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>
Question 1. Short Answer (15 points).

a. (5 points) Briefly explain what a virtual method table (or vtable) is and what it’s used for.

   Answer: It is a collection of methods for a particular class. Each instance of a class has a pointer to the virtual method table for that class. When one of an object’s instance methods is invoked, it is resolved by looking it up in the virtual method table.

b. (5 points) Briefly describe the role of an environment in an interpreter or compiler.

   Answer: An environment maps variables to their meaning (values). It is an efficient implementation of substitution.
c. (5 points) Briefly explain the goal of defunctionalization.

**Answer:** Defunctionalization is a program transformation that eliminates the need for function values in a program.
Question 2. Program transformations (15 points).

a. (15 points) Apply defunctionalization to this program:

```plaintext
let rec bk n k = match n with
  | 0 -> k 0
  | 1 -> k 1
  | n -> bk (n-1) (fun fn1 -> bk (n-2) (fun fn2 -> k (fn1 + fn2))
let b n = bk n (fun n -> n)
```

**Answer:**

```
type k = K0
  | K1 of int * k
  | K2 of int * k

let rec bk n k = match n with
  | 0 -> apply k 0
  | 1 -> apply k 1
  | n -> bk (n-1) (K1 (n, k))
and apply k n = match k with
  | K0 -> n
  | K1 (m, k) -> bk (m-2) (K2 (n, k))
  | K2 (m, k) -> apply k (m+n)
let b n = bk n K0
```
Question 3. Type Systems (25 points).

a. (8 points) Assume that \textit{int} < \textit{float}. Write down every type \( t \) such that \( t \leq \textit{int} \rightarrow \textit{float} \rightarrow \textit{float} \), following standard subtyping rules.

\textbf{Answer:} \\
\( \textit{int} \rightarrow \textit{float} \rightarrow \textit{int} \) \\
\( \textit{int} \rightarrow \textit{float} \rightarrow \textit{float} \) \\
\( \textit{float} \rightarrow \textit{float} \rightarrow \textit{int} \) \\
\( \textit{float} \rightarrow \textit{float} \rightarrow \textit{float} \)

b. (2 points) Assume that \textit{int} < \textit{float}. Write down every type \( t \) such that \( t \leq \textit{int \ ref} \rightarrow \textit{float \ ref} \), following standard subtyping rules.

\textbf{Answer:} \\
\( \textit{int \ ref} \rightarrow \textit{float \ ref} \)
c. (5 points) Recall the simply typed lambda calculus:

\[
\begin{align*}
e & ::= n \mid x \mid \lambda x: t. e \mid e \ e \\
t & ::= \text{int} \mid t \rightarrow t \\
A & ::= \emptyset \mid x : t, A
\end{align*}
\]

Draw a derivation that the following type judgment holds, where \( A = + : \text{int} \rightarrow \text{int} \rightarrow \text{int} \). (You can draw the derivation upward from the judgment, and you can write \( i \) instead of \( \text{int} \) to save time):

**Answer:**

\[
\begin{array}{c}
\text{INT} \quad \text{VAR} \quad \text{LAM} \quad \text{APP} \\
A \vdash n : \text{int} & A \vdash x : A(x) & A \vdash (\lambda x : t. e) : t ightarrow t' & A \vdash e_1 : t \rightarrow t' \quad A \vdash e_2 : t \\
\hline
\hline
A \vdash (\lambda x : t. e) : t \rightarrow t' \\
A \vdash e_1 \\
A \vdash e_2 \\
A \vdash e_1 e_2 : t'
\end{array}
\]

\[
\begin{array}{c}
x : \text{int}, A \vdash + : \text{int} \rightarrow \text{int} \rightarrow \text{int} & x : \text{int}, A \vdash x : \text{int} \\
\hline
\hline
A \vdash (\lambda x : \text{int}. +) : \text{int} \rightarrow \text{int} \rightarrow \text{int} & A \vdash 1 : \text{int} \\
\hline
\hline
A \vdash (\lambda x : \text{int}. +) 1 : \text{int} \rightarrow \text{int}
\end{array}
\]

\[
A \vdash (\lambda x : \text{int}. +) 1 : \text{int} \rightarrow \text{int}
\]
d. (10 points) Perform type inference on the following program by listing the types that OCaml will infer for the blanks:

```ocaml
let rec mumble (f : ...) (xs : ...) (ys : ...) : ... =
  match xs, ys with
  | [], [] → []
  | x::xs, y::ys →
    ((f (x+1))-1,(f (y-1))+1):(mumble f xs ys)
```

Answer:

```ocaml
let rec mumble (f : (int → int)) (xs : int list) (ys : int list) : (int * int) list = ...
```