Practice Midterm Exam II

CAS CS 320: Concepts of Programming Languages

Name:

BUID:

- ▷ You will have approximately 75 minutes to complete this exam. Make sure to read every question, some are easier than others.
- ▷ Do not remove any pages from the exam.
- ▷ Make very clear what your final solution for each problem is (e.g., by surrounding it in a box). We reserve the right to mark off points if we cannot tell what your final solution is.
- ▷ You must show your work on all problems unless otherwise specified. A solution without work will be considered incorrect (and will be investigated for potential academic misconduct).
- ▷ Unless stated otherwise, you should only need the rules provided **in the problem statement** for any derivation.
- ▷ We will not look at any work on the pages marked "*This page is intentionally left blank.*" You should use these pages for scratch work.

1 Higher-Order Programming

Implement the function `val func_max : (int -> int) list -> int -> int` so that `func fs x` is given by

$$\max(\max_{f \in \texttt{fs}} \{f(\texttt{x})\}, 0)$$

You should accomplish this by a single call to `fold_left`. That is, you should give definitions for `op` and `base` below.

```
let op : (int -> int) -> (int -> int) -> int -> int = assert false (* TODO *)
let base : int -> int = assert false (* TODO *)

let func_max (fs : (int -> int) list) : int -> int = List.fold_left op base fs

let _ = assert (func_max [(+) 1; fun x -> x * x] 1 = 2)
let _ = assert (func_max [(+) 1; fun x -> x * x] (-2) = 4)
```

2 Formal Grammar

```
<e>  ::= <e> <op> <e1> | <e1>
<e1> ::= <e1> <e2> | <e2>
<e2> ::= - <e2> | <n> | <v> | ( <e>)
<op> ::= + | -
<n>  ::= 0 | 1 | 2
<v>  ::= x | y | z | f
```

A. Demonstrate that the sentence `f x + y (z 2)` is recognized by the above grammar by providing a parse tree for it.

B. Demonstrate that the above grammar is ambiguous by determining a sentence which has multiple parse trees. Include both parse trees in your solution.

3 Small-Step Semantics

Recall the following small-step semantic rules for the λ-calculus:

$$\frac{e_1 \longrightarrow e_1'}{e_1\ e_2 \longrightarrow e_1'\ e_2}\ \textsc{appLeft} \qquad \frac{e_2 \longrightarrow e_2'}{(\lambda x.e)\ e_2 \longrightarrow (\lambda x.e)\ e_2'}\ \textsc{appRight} \qquad \frac{}{(\lambda x.e)(\lambda y.e') \longrightarrow [\lambda y.e'/x]e}\ \textsc{beta}$$

A. Determine the expression e_0 such that $(\lambda x.xxx)(\lambda x.xxx) \longrightarrow e_0$. Show your work.

B. Based on the expression e_0 you determined, does $(\lambda x.xxx)(\lambda x.xxx)$ evaluation evaluate to a value in finitely many steps (i.e., does it terminate)? Answer yes/no with a brief justification.

4 Substitution

Recall the following definition of substitution for expressions of the λ-calculus.

$$[v/y]x = \begin{cases} v & x = y \\ x & \text{otherwise} \end{cases}$$

$$[v/y](\lambda x.e) = \begin{cases} \lambda x.e & x = y \\ \lambda z.[v/y][z/x]e & x \in FV(v), z \text{ is fresh} \\ \lambda x.[v/y]e & \text{otherwise} \end{cases}$$

$$[v/y](e_1 e_2) = ([v/y]e_1)([v/y]e_2)$$

Perform the following substitutions. In the case of capture-avoidance, you are free to choose whatever variable names you want which maintain α-equivalence. Show your work.

A. $[\lambda x.y/z](\lambda x.\lambda y.z)$

B. $[\lambda x.x/f](\lambda y.\lambda x.f(yx))$

C. $[\lambda z.x/y][\lambda x.y/z](\lambda y.\lambda x.zy)$