CMSC330 Fall 2019 - Midterm 2

First and Last Name (PRINT): ____________________________

9-Digit University ID: ____________________________

Instructions:
- Do not start this test until you are told to do so!
- You have 75 minutes to take this midterm.
- This exam has a total of 100 points, so allocate 45 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.
- Write your 9-Digit UID at the top of EVERY PAGE.

| 1. PL Concepts           | / 15 
|-------------------------|------
| 2. Finite Automata      | / 30 
| 3. CFGs and Parsing     | / 30 
| 4. Operational Semantics| / 10 
| 5. Lambda Calculus      | / 15 
| **Total**               | / 100 

Please write and sign the University Honor Code below: I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

.............................................................
.............................................................
.............................................................

Signature: ____________________________
1. [15pts] PL Concepts

1 (7pts) **Circle your answers.** Each T/F question is 1 point.

<table>
<thead>
<tr>
<th>T</th>
<th>F</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A regular expression can express all palindromes with letters A-Z, and shorter than 10 letters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Static analysis, such as type checking, occurs before parsing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>There are multiple paths by which the same string can be accepted in a DFA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Calling a grammar ambiguous is equivalent to saying a string may have multiple different leftmost derivations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Using lookahead in our parser is an example of predictive parsing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operational semantics are analogous to interpreting a program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Regular expressions are more powerful than DFAs (i.e., they can express more languages than DFAs can)</td>
</tr>
</tbody>
</table>

2 (1pts) The step below is an example of...

\[(\lambda x \cdot x y) \ (\lambda z \cdot a z) \ (\lambda z \cdot a z) y\]

A. \(\alpha\)-conversion

B. \(\beta\)-reduction
3 (3pts) What is the output of the following OCaml code? (That is, what is printed)

```ocaml
let x = ref 0 in
let y = x in
y := 1;
print_int !x;
print_int !y
```

**OUTPUT:**

```

```

4 (4pts) What is printed by the following OCaml program when the parameters are passed by call-by-name and call-by-value?

```ocaml
let f x y =
  if x > 5 then (y,y) else (10,10);;
f 10 (print_string "hello"; 2);;
```

**Call-by-name:**

```

```

**Call-by-value:**

```

```
2. [30pts] Finite Automata

1 (6pts) Which of the following strings are accepted by this NFA? *Circle all that apply.*

A. abcab
B. abca
C. abccc
D. aacaccaca

2 (8pts) Construct an NFA that accepts the same language as the following regular expression. There are many answers, any equivalent NFA will be accepted.

\[(a+|b^*)c?\]
3  (6pts)  Answer the following questions about this NFA:

\[ 0 \xleftarrow{\varepsilon} 1 \xrightarrow{a} 0 \xrightarrow{b} 2 \xrightarrow{a} 1 \]

- \[ \text{e-closure}\{0\} = \]
- \[ \text{e-closure}(\text{move}\{0, 1\}, a) = \]

4  (10pts)  Give a DFA equivalent to the NFA above. Any equivalent DFA will be accepted, but your answer should be clear. You may give steps for partial credit.
3. [30pts] CFGs and Parsing

1. (5pts) Write a CFG that generates the following language:

   \[a^x b^y c^x y, \text{ where } x, y \geq 0\]

2. (5pts) The following CFG is ambiguous. Rewrite it so that it is not ambiguous. There are many answers, any CFG which is equivalent and is not ambiguous will be accepted. (Note: here, the terminals are: +, *, (, ), a, and b.)

   \[E \rightarrow E + E \mid E * E \mid (E) \mid a \mid b\]

3. (4pts) List the FIRST SETS for each nonterminal in the following grammar (lowercase letters are terminals):

   \[S \rightarrow aB \mid Bb \mid Sc\]
   \[B \rightarrow dB \mid d\]
4  (6pts)  Indicate if each of the following grammars can be parsed by a recursive descent parser. If the answer is no, give a very brief explanation why.

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Yes</th>
<th>No</th>
<th>If no, why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow S \cdot S \mid N$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N \rightarrow 1 \mid 2 \mid 3 \mid (S)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S \rightarrow aS \mid B$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$B \rightarrow bB \mid b$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S \rightarrow Sb \mid A$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A \rightarrow aAc \mid c$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5  (10pts)  Complete the OCaml implementation for a recursive-descent parser of the following context-free grammar. The implementation of `match_tok` and `lookahead` are given below:

```ocaml
let tok_list = ref [];;
let match_tok x = match !tok_list with
  | h :: t when x = h -> tok_list := t
  | _ -> raise (ParseError "bad match");;
let lookahead () = match !tok_list with
  | [] -> None
  | h :: t -> Some h
```

NOTE: this parser takes the imperative approach. Also notice that the tokens are simply strings. So the token list for the string "abcdc" would look like `["a"; "b"; "c"; "d"; "c"]`. You are not creating an AST. If the input is invalid, throw a `ParseError`.

**Write your implementation on the next page. The CFG is repeated on the next page for your reference.**
let rec parse_S () =
    if lookahead () = Some "b" then
        match_tok "b";
        parse_S ()
    else (* fill in below *)

and rec parse_T () = (* fill in below *)

and rec parse_R () =
    if lookahead () = None then
        ()
    else (* fill in below *)

<table>
<thead>
<tr>
<th>S</th>
<th>→ bS</th>
<th>cT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>→ Ra</td>
<td>RbR</td>
</tr>
<tr>
<td>R</td>
<td>→ dR</td>
<td>ε</td>
</tr>
</tbody>
</table>
4. [10pts] Operational Semantics

1 (2pts) Below is an incorrect rule for an if-then-else construct when the condition is true. Identify the mistake, and explain how to fix it. Here, the expression if a then b else c is encoded as if-then-else a b c.

\[
\begin{array}{c}
A; e_1 \rightarrow true \\
A; e_3 \rightarrow v
\end{array} \quad \text{IFTHENELSE-TRUE} \\
\begin{array}{c}
A; \text{if-then-else } e_1 e_2 e_3 \rightarrow v
\end{array}
\]

2 (3pts) Describe what the operator myst does, or give its name.

\[
\begin{array}{c}
A; e_1 \rightarrow true \\
A; e_2 \rightarrow true
\end{array} \quad A; e_1 \rightarrow false \quad A; e_2 \rightarrow false
\]

\[
\begin{array}{c}
A; e_1 \rightarrow true \\
A; e_2 \rightarrow true
\end{array} \quad A; e_1 \rightarrow false \quad A; e_2 \rightarrow false
\]

\[
\begin{array}{c}
A; e_1 \rightarrow false \\
A; e_2 \rightarrow true
\end{array} \quad A; e_1 \rightarrow false \quad A; e_2 \rightarrow false
\]

\[
\begin{array}{c}
A; e_1 \rightarrow false \\
A; e_2 \rightarrow false
\end{array} \quad A; e_1 \rightarrow false \quad A; e_2 \rightarrow false
\]
3 (5pts) Using the following rules, show that:

\[
A; \text{let } x = 3 \text{ in let } x = 2 \text{ in } x + x \rightarrow 4
\]

\[
\begin{align*}
A; n & \rightarrow n \\
A(x) = v & \rightarrow A; x \rightarrow v
\end{align*}
\]

\[
\begin{align*}
A; e_1 & \rightarrow v_1 \\
A; e_1 \rightarrow n_1
\end{align*}
\]

\[
\begin{align*}
A; x: v_1; e_2 & \rightarrow v_2 \\
A; \text{let } x = e_1 \text{ in } e_2 & \rightarrow v_2
\end{align*}
\]

\[
\begin{align*}
A; e_1 & \rightarrow n_1 \\
A; e_2 & \rightarrow n_2
\end{align*}
\]

\[
\begin{align*}
n_3 \text{ is } n_1 + n_2 & \rightarrow A; e_1 + e_2 \rightarrow n_3
\end{align*}
\]
5. [15pts] Lambda Calculus

1 (8pts) Reduce the expressions as far as possible by showing the intermediate β-reductions and α-conversions. Make sure to show each step for full credit!

\[(\lambda x. \lambda y. x \ y) \ (\lambda y. y) \ x\]

\[(\lambda x. \lambda y. x \ y \ y) \ (\lambda m. m) \ n\]
2 (7pts) Reduce the following expression to $\beta$-normal form using both call-by-name and call-by-value. Show each step, including any $\beta$-reductions and $\alpha$-conversions. If there is infinite reduction, write “infinite reduction.”

$$(\lambda y. x) ((\lambda x. x x x) (\lambda z. z z z))$$

Call-by-name:


Call-by-value:


