CMSC 330, Fall 2018 — Midterm 2

NAME ______________________________

TEACHING ASSISTANT

Kameron Aaron Danny Chris Michael P. Justin Cameron B. Derek Kyle Hasan Shriraj Cameron M. Alex Michael S. Pei-Jo

INSTRUCTIONS

- Do not start this exam until you are told to do so.
- You have 75 minutes for this exam.
- This is a closed book exam. No notes or other aids are allowed.
- For partial credit, show all your work and clearly indicate your answers.

HONOR PLEDGE

Please copy and sign the honor pledge: "I pledge on my honor that I have not given or received any unauthorized assistance on this examination."

____

____

____

____

Section	Points
Programming Language Concepts	10
Finite Automata	23
Context-Free Grammars	18
Parsing	18
Operational Semantics	11
Lambda Calculus	13
Imperative OCaml	7
Total	100

1 Programming Language Concepts

In the following questions, circle the correct answer.

1. [1 pts] (T / F) The input to a lexer is source code and its output is an abstract syntax tree.
2. [1 pts] (T / F) Any language that can be expressed by a context-free grammar can be expressed by a regular expression.
3. [1 pts] (T / F) OCaml is Turing-complete.
4. [1 pts] (T / F) Converting a DFA to an NFA always requires exponential time.
5. [1 pts] (T / F) Recursive descent parsing requires the target grammar to be right recursive.
6. [1 pts] (T / F) The SmallC parser in P4A used recursive descent.
7. [1 pts] (T / F) The call-by-name and call-by-value reduction strategies can produce different normal forms for the same λ expression.
8. [1 pts] (T / F / Decline to Answer) I voted last Tuesday. (All answers are acceptable.)
9. [1 pts] What language feature does the fixed-point combinator implement?

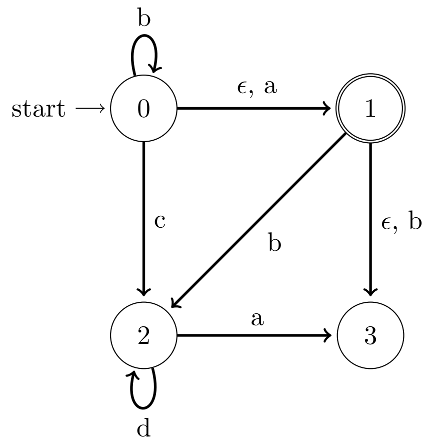
   (a) Booleans (b) Integers (c) Recursion (d) Closures

10. [1 pts] What is wrong with this definition of an NFA?

```
type ('q, 's) nfa = {
    qs : 'q list;
    sigma : 's list;
    delta : ('q, 's) transition list;
    q0 : 'q list;
    fs : 'q list;
}
```

   (a) Allows states with multiple transitions on the same character.
   (b) Allows ε-transitions.
   (c) Allows multiple final states.
   (d) Allows multiple start states.

2 Finite Automata



1. Use the NFA shown above to answer the following questions.

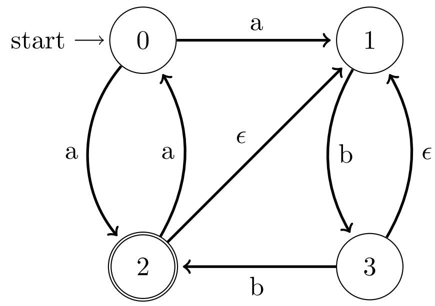
   - [2 pts] ε-closure$(\{0\}) = \{$ $\}$
   - [2 pts] move$(\{1\}, b) = \{$ $\}$

2. [1 pts] (T / F) Every NFA is also a DFA.

3. [1 pts] (T / F) Every DFA is also an NFA.

4. [5 pts] Draw an NFA that corresponds to the following regular expression: $((a^{\star}b) \mid (ab))^{\star}$

5. [7 pts] Convert the following NFA into an equivalent DFA.



6. [5 pts] Circle all of the strings that will be accepted by the above **NFA**. (**Note**: Not the DFA you generated)

(a) abbaa (b) aaaa (c) abbaabb (d) abbbbaab (e) aaaaa

3 Context-Free Grammars

1. [6 pts] Write a CFG that is equivalent to the regular expression $(wp)^+g^\star$

2. [6 pts] Create a CFG that generates all strings of the form $a^xb^ya^z$, where $y = x + z$ and $x, y, z \geq 0$.

3. [6 pts] Given the following grammar, where S and A denote non-terminals, give a right-most and left-most derivation of $((100, 33), 30)$. Show all steps of your derivation.

$$S \rightarrow A \mid (S, S)$$

$$A \rightarrow 100 \mid 33 \mid 30$$

4 Parsing

1. [3 pts] Convert the following to a right-recursive grammar.

$$S \to S + S \mid A$$

$$A \to A * A \mid B$$

$$B \to n \mid (S)$$

2. [5 pts] What are the first sets of the non-terminals in the following grammar?

$$S \to bc \mid cA$$

$$A \to cAd \mid B$$

$$B \to wS \mid \varepsilon$$

3. [10 pts] Finish the definition of a recursive descent parser for the grammar below. You need not build an AST, assume all methods return unit. Note that `match_tok` takes a string.

$$S \to Abc \mid A$$

$$A \to cAd \mid e$$

```
let lookahead () : string =
  match !tok_list with
  | [] ->  raise (ParseError "no tokens")
  | h::t -> h

let match_tok (a : string) : unit =
  match !tok_list with
  | h::t when a = h -> tok_list := t
  | _ -> raise (ParseError "bad match")

let rec parse_S () : unit =
```

```
and parse_A () : unit =
```

5 Operational Semantics

$$\frac{}{A;\ \textbf{false} \Rightarrow \textbf{false}} \qquad \frac{}{A;\ \textbf{true} \Rightarrow \textbf{true}}$$

$$\frac{}{A;\ n \Rightarrow n} \qquad \frac{A(x) = v}{A;\ x \Rightarrow v}$$

$$\frac{A; e_1 \Rightarrow v_1 \qquad A, x : v_1;\ e_2 \Rightarrow v_2}{A;\ \textbf{let}\ x = e_1 \textbf{in}\ e_2 \Rightarrow v_2} \qquad \frac{A;\ e_1 \Rightarrow n_1 \qquad A;\ e_2 \Rightarrow n_2 \qquad n_3\ is\ n_1 + n_2}{A;\ e_1 + e_2 \Rightarrow n_3}$$

$$\frac{A;\ e_1 \Rightarrow \textbf{true} \qquad A;\ e_2 \Rightarrow v}{A;\ \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \Rightarrow v} \qquad \frac{A;\ e_1 \Rightarrow \textbf{false} \qquad A;\ e_3 \Rightarrow v}{A;\ \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \Rightarrow v}$$

Use the above rules to fill in the given constructions.

1. [6 pts]

$$\frac{A;\ \boxed{} \qquad \dfrac{A;\ \boxed{} \qquad A;\ \boxed{} \qquad \boxed{}}{A;\ \boxed{}}}{A;\ \textbf{if}\ \boxed{}\ \textbf{then}\ 10 + 3\ \textbf{else}\ 5 + 2 \Rightarrow 7}$$

2. [5 pts]

$$\frac{\boxed{};\ 4 \Rightarrow 4 \qquad \dfrac{\boxed{};\ 5 \Rightarrow 5 \qquad \dfrac{\boxed{}(x) \Rightarrow 5}{\boxed{};\ x \Rightarrow 5}}{\boxed{}\ \textbf{let}\ x = 5\ \textbf{in}\ x \Rightarrow 5}}{A;\ \textbf{let}\ x = 4\ \textbf{in}\ \textbf{let}\ x = 5\ \textbf{in}\ x \Rightarrow 5}$$

6 Lambda Calculus

1. [2 pts] Circle all of the free variables in the following λ expression. (A variable is **free** if it is not bound by a λ abstraction.)

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

2. [2 pts] Circle all of the following where the λ expressions are α-equivalent.

   (a) $((\lambda a.\ (\lambda y.\ y\ a)\ y)$ and $(\lambda x.\ x\ y)$

   (b) $(\lambda x.\ (\lambda y.\ x\ y))$ and $(\lambda y.\ (\lambda x.\ y\ x))$

3. Reduce each λ expression to β-normal form (to be eligible for partial credit, show each reduction step). If already in normal form, write "normal form." If it reduces infinitely, write "reduces infinitely."

   (a) [2 pts] x (λa. λb. b a) x (λy. y) — Hint: application is left-associative.

   (b) [2 pts] $((\lambda x.\ x\ x)(\lambda y.\ y\ y))$

   (c) [2 pts] $((\lambda a.\ \lambda b.\ a\ b\ c)\ x\ y)$

4. [3 pts] Write an OCaml expression that has the same semantics as the following λ expression.

$$(\lambda a.\ \lambda b.\ a\ b)\ (\lambda x.\ x\ x)\ y$$

7 Imperative OCaml

1. [7 pts] Given the `mut_lst` variable, which is `'a ref list`, implement the `add` and `contains` functions which should add a given element to `mut_lst` and check if the `mut_lst` contains a specified element, respectively. You may add helpers and change the functions to be recursive.

```
let mut_lst = ref []
```

```
let add (ele : 'a) : unit =
```

```
let contains (ele : 'a) : bool =
```