CMSC330 – Organization of Programming Languages
Fall 2022
Exam 2

CMSC330 Course Staff
University of Maryland
Department of Computer Science

November 17th, 2022

- Do not remove the staple from your exam packet.
- Do not any individual sheets from the exam packet
- Write your name and UID in the header of each page.
- Refrain from bending or folding the exam in any place except near the staple, this helps us when scanning your exams.
- **Read *all* questions carefully before starting.**

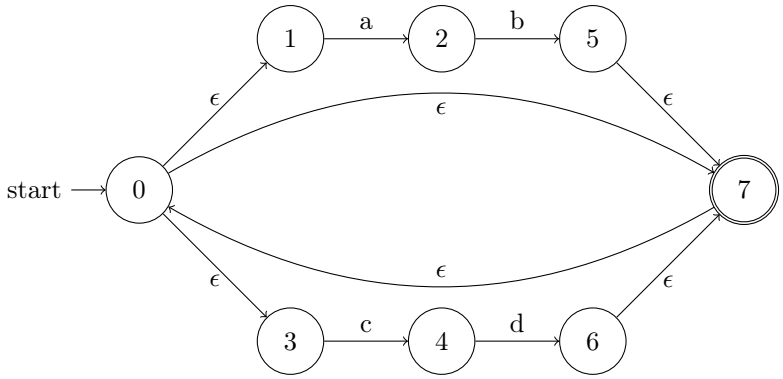
NAME: _____

UID: _____

Question	Points
1	10
2	10
3	10
4	6
5	15
6	10
Total:	61

Name: UID:

1. Non-deterministic Finite Automata
   (a) [8 points] Convert the following NFA into a DFA:



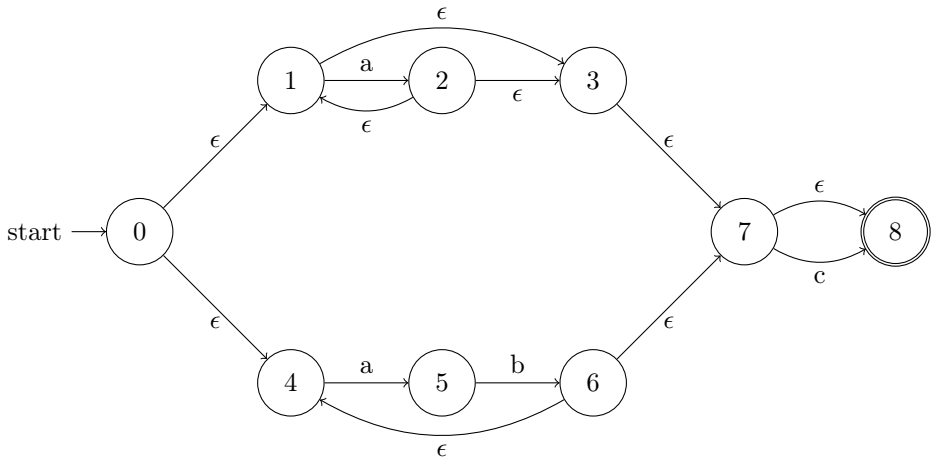
Scratch space for Question 1:

Your answer here (There is scratch space on the previous page):

(b) [2 points] What is an equivalent Regular Expression for the NFA in Question 1(a)?

Name: UID:

2. Consider the following NFA:



(a) [6 points] Which of the following strings does the NFA accept? Mark all that apply.

- ◯ "aaaaac"
- ◯ <empty string>
- ◯ "abbc"
- ◯ "ababac"
- ◯ "ababab"
- ◯ "ababc"

(b) [4 points] What is an equivalent Regular Expression for the Question 2 NFA?

Scratch space to use as you please:

Name: UID:

3. Context Free Grammars

(a) [4 points] Write a Regular Expression that describes the same set of strings as the following grammar, if it cannot be converted to a regex, explain why:

$S \rightarrow$ aSU | T
$T \rightarrow$ bTU | ϵ
$U \rightarrow$ cU | c

(b) [6 points] Consider the following grammar:

$S \rightarrow$ bSc | AS | c | ϵ
$A \rightarrow$ bA | ϵ

i. Provide the derivation of an example string of your own invention (i.e. you cannot use the string from the next part) that shows that the grammar is ambiguous.

ii. Given the string "bbc", either draw two ASTs or show two derivations that prove the grammar is ambiguous:

Name: UID:

4. Parsing

(a) [3 points] Can the following grammar be parsed by a recursive-decent parser? Select the option that is the most accurate.

$$\begin{aligned} \text{S} &\rightarrow \quad \text{bS} \mid \text{C} \\ \text{C} &\rightarrow \quad \text{cC} \mid \epsilon \end{aligned}$$

- ◯ Yes
- ◯ No, because the grammar is ambiguous
- ◯ No, because the grammar is left recursive
- ◯ No, because the grammar is ambiguous *and* left recursive

(b) [3 points] You are using a programming language with the grammar provided below. In your editor, you write the program "`1 2 + *`" and try to run the program. At which stage of the implementation will there be an error, if any?

$$\begin{aligned} \text{S} &\rightarrow \quad \text{M} + \text{S} \mid \text{M - S} \mid \text{M} \\ \text{M} &\rightarrow \quad \text{N * S} \mid \text{N / S} \mid \text{N} \\ \text{N} &\rightarrow \quad \text{n} \end{aligned}$$

- ◯ Lexer
- ◯ Parser
- ◯ Interpreter
- ◯ There will be no error, that is a valid program.

5. Semantics

Take note of the following Operational Semantics for a simple language with `let`-bindings, conditionals, addition, and comparison of numbers. Each semantic rule is labelled with a name to its left (e.g. the top-left-most rule is labelled 'num').

$$\text{num}\ \frac{}{A; n \rightarrow n} \qquad \text{lookup}\ \frac{A(x) = v}{A; x \rightarrow v} \qquad \text{let}\ \frac{A; e_1 \rightarrow v_1 \qquad A; x : v_1; e_2 \rightarrow v_2}{A; \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \rightarrow v_2}$$

$$\text{add}\ \frac{A; e_1 \rightarrow v_1 \qquad A; e_2 \rightarrow v_2 \qquad v_3 \text{ is } v_1 + v_2}{A; e_1 + e_2 \rightarrow v_3}$$

$$\text{gt-true}\ \frac{A; e_1 \rightarrow n_1 \qquad A; e_2 \rightarrow n_2 \qquad n_1 > n_2}{A; e_1 > e_2 \rightarrow \mathit{true}} \qquad \text{gt-false}\ \frac{A; e_1 \rightarrow n_1 \qquad A; e_2 \rightarrow n_2 \qquad n_1 \leq n_2}{A; e_1 > e_2 \rightarrow \mathit{false}}$$

$$\text{if-true}\ \frac{A; e_1 \rightarrow \text{true} \qquad A; e_2 \rightarrow v}{A; \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \rightarrow v} \qquad \text{if-false}\ \frac{A; e_1 \rightarrow \text{false} \qquad A; e_3 \rightarrow v}{A; \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \rightarrow v}$$

Name: UID:

(a) [1 point] What is the term for a conclusion that automatically holds (i.e. there is nothing to show in order for us to accept it as true)?

- ◯ Predicate
- ◯ Syllogism
- ◯ Axiom
- ◯ Premise

(b) [1 point] Which rules, if any, from the Operational Semantics above are an instance of your answer to Question 5(a)?

(c) [3 points] The semantics, as given, do not define how to evaluate booleans, even though they are used in the semantics! To understand this, look at the fact that there's no rule where *only* `true` or *only* `false` are evaluated. Add your own rule(s) that fix this issue.

Name: UID:

(d) [10 points] Construct the full proof for the following proposition (you should not use your new rule):

$$A; \texttt{let}\ a = 21 + 21\ \texttt{in}\ (42 + 42) > a \to \text{true}$$

Name: UID:

6. Lambda Calculus

(a) [4 points] Consider the following lambda expression:

$$\lambda j.k\ i\ \lambda h.e\ \lambda k.h\ g\ \lambda f.j\ k$$

Circle the *free* variables:

$$\lambda j.k\ i\ \lambda h.e\ \lambda k.h\ g\ \lambda f.j\ k$$

Circle the *bound* variables:

$$\lambda j.k\ i\ \lambda h.e\ \lambda k.h\ g\ \lambda f.j\ k$$

(b) [6 points] Evaluate the following lambda expression as much as possible:

$$(\lambda m.\lambda n.n\ m)\ (\lambda f.\lambda x.f\ (f\ x))\ (\lambda f.\lambda x.f\ x)$$

Result:

Scratch space for Question 6(b):

Name: UID:

General Scratch Space

Name: UID:

Useful Information

NFA to DFA Algorithm:

NFA (input): $(\Sigma, Q, q_0, F_n, \sigma)$, DFA (output): $(\Sigma, R, r_0, F_d, \sigma_n)$

$R \leftarrow \{\}$
$r_0 \leftarrow \epsilon - \text{closure}(\sigma, q_0)$
while $\exists$ an unmarked state $r \in R$ **do**
  mark r
  for all $a \in \Sigma$ **do**
    $E \leftarrow \text{move}(\sigma, r, a)$
    $e \leftarrow \epsilon - \text{closure}(\sigma, E)$
    if $e \notin R$ **then**
      $R \leftarrow R \cup \{e\}$
    end if
    $\sigma_n \leftarrow \sigma_n \cup \{r, a, e\}$
  end for
end while
$F_d \leftarrow \{r | \exists s \in r \text{with} s \in F_n\}$

Grammar for the Lambda Calculus:

```
e  :=  v
   |   e e
   |   λ v . e
```