CMSC330 Fall 2025 Quiz 3

Proctoring TA: _____ **Name:** _____

Section Number: _____ **UID:** _____

Problem 1: Basics

[Total 3 pts]

The primary task of a lexer is to check the semantics of a language (T) (F)

If a DFA accepts a string, then there exists exactly one path from the start to an accepting state for that string (T) (F)

The language $\{a^n b^n | n \geq 0\}$ can be generated by a CFG but not accepted by any DFA or NFA (T) (F)

For some languages, the equivalent DFA may have exponentially more states than the NFA (T) (F)

NFAs and DFAs accept the same class of languages — regular languages (T) (F)

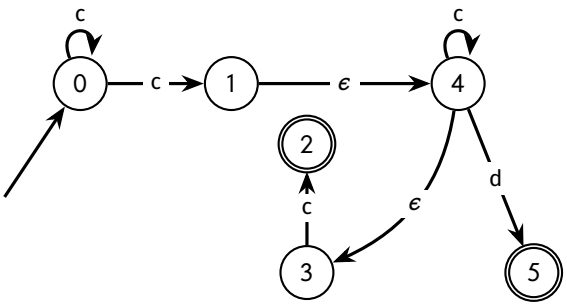
In a DFA, for each state (excluding garbage states) and input symbol there is:

(A) Exactly one transition (B) At most one transition /true (C) No transition (D) At least one transition

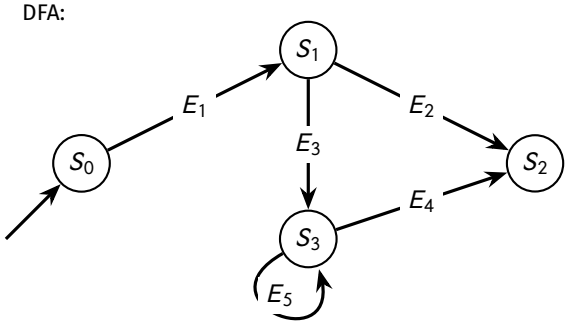
Problem 2: NFA to DFA

[Total 10 pts]

NFA:



DFA:



Scratch Table (for partial credit):

Final?	State	c	d

S_0: S_1:

S_2: S_3:

E_1: E_2:

E_3: E_4:

E_5:

(a) Which states are the final (accepting) states? [1 pts]

(A) S_0 (B) S_1 (C) S_2 (D) S_3

Problem 3: Lexing, Parsing, Interpreting

[Total 5 pts]

Write a recursive descent parser that recognizes strings generated by the following grammar:
$S \rightarrow xySz \mid w$
This grammar generates strings such as: xywz, xyxywzz, ... i.e. strings of the form $x^n y^n w z^n$ where $n \geq 0$.
The input is represented as a list of characters (char list).
You have already been given the implementation for `parse`, the optional functions `lookahead` and `match_tok`,
implement the function `parse_S` such that the examples below function correctly:

```
parse ['w'] = [] (* success *)                    parse ['a'] = (* returns error *)
parse ['x'; 'y'; 'w'; 'z'] = [] (* success *)     parse ['x'; 'y'; 'w'] (* returns error *)
parse ['x'; 'y'; 'x'; 'y'; 'w'; 'z'; 'z'] = [] (* success *)
```

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

```
let match_tok a tokens =
  match tokens with
  | h::t when a = h -> t
  | _ -> raise (ParseError "bad match")
```

```
let parse tokens = match parse_S tokens with
  | [] -> []
  | _ -> failwith "failed to parse the input string"

let rec parse_S tokens =
```

```
  |_->failwith "wrong token. parse error"
```

Problem 4: CFG Derivation

[Total 2 pts]

A grammar is said to be ambiguous if:

(A) It has multiple start symbols
(B) It cannot generate any string
(C) It has no production rules
(D) A string can have more than one parse tree

The lookahead in parsing helps to:

(A) Optimize token generation
(B) Generate intermediate code
(C) Predict which rule to apply next
(D) Simplify the grammar

Complete the CFG, such that it generates all even-length strings over $\Sigma = \{a, b, ..., z\}$ (including empty strings)

$S \rightarrow$

$T \rightarrow \quad a|b|c..|z$