CS 340, Fall 2015 — Oct 1st — Exam 1 Name: Solution

Note: in all questions, the special symbol ϵ (epsilon) is used to indicate the empty string.

Question 1. [10 points] Define a regular expression for the language containing all strings of symbols in the alphabet {**a**, **b**, **c**} where each occurrence of **b** is immediately preceded and followed by at least one occurrence of **c**.

Examples of strings in the language:

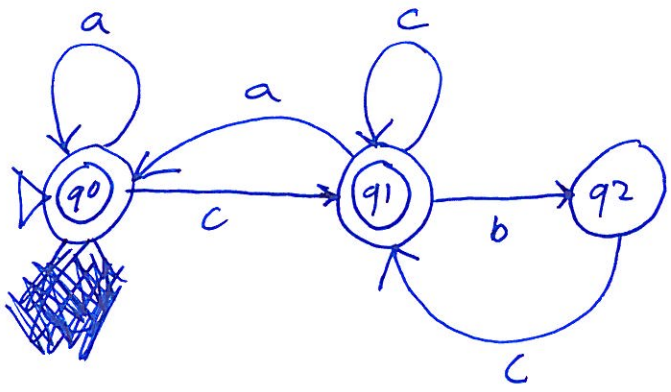
- ϵ
- **a**
- **ca**
- **cac**
- **acbcc**
- **cbcccacbca**

Examples of strings not in the language:

- **bc**
- **cbbc**
- **abc**
- **accb**

$(a|c|cbc)^*$

Question 2. [10 points] Specify a *deterministic* finite automaton that recognizes the language described in Question 1. Make sure that the start state and final state(s) are labeled, and that each transition indicates the direction and what input symbol is consumed.



q0: start of string, or prev. symbol was an 'a'

q1: prev. symbol was a 'c'

q2: prev. symbol was a 'b', so next symbol must be a 'c'

Question 3. [10 points] Define a regular expression that generates the language over the alphabet {**a**, **b**, **c**} of all strings in which

- every other symbol (first, third, fifth, etc., or second, fourth, sixth, etc.) is a **b**
- there are never two or more adjacent occurrences of **b**

Note that the empty string *is* in this language.

Examples of strings in the language:

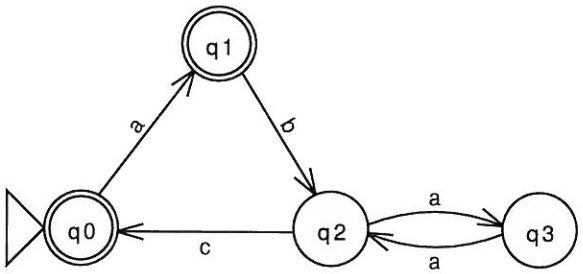
ϵ
a
abc
abcbc
abcbcb
bcb
bcba

Examples of strings not in the language:

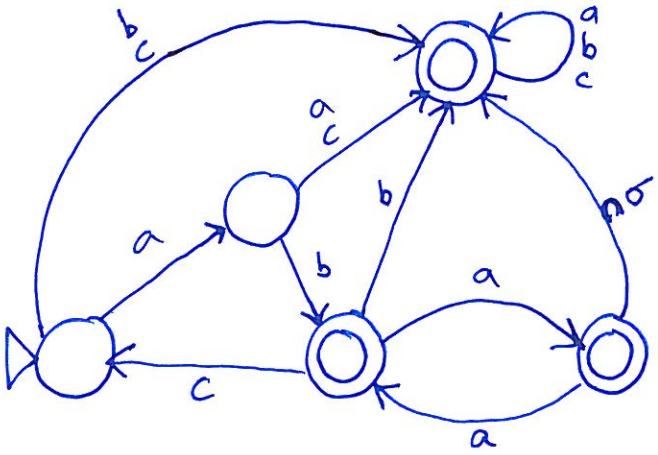
abca
abccb
babbb
ababaa

$(ba|bc)^*(b|\varepsilon) \mid (ab|cb)^*(a|c|\varepsilon)$

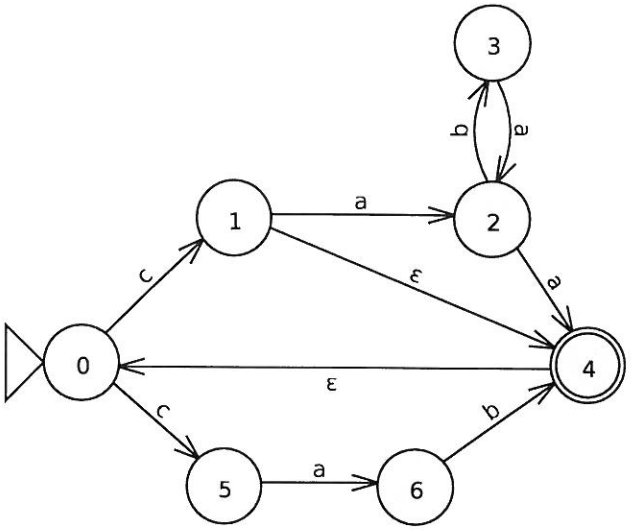
Question 4. [10 points] Consider the following deterministic finite automaton (DFA):



Show a DFA that recognizes the complement of the language accepted by this DFA (over the same alphabet). In other words, if a string is accepted by the DFA, your DFA should reject it, and if a string is rejected by the DFA, your DFA should accept it.

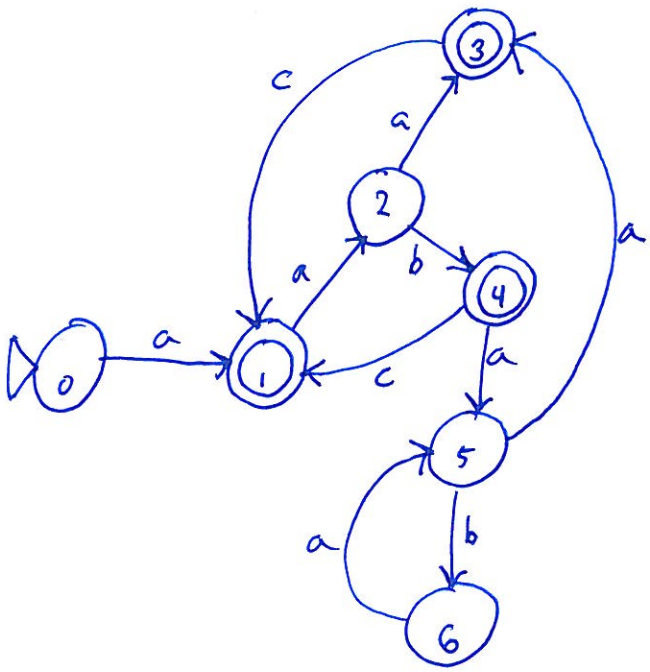


Question 5. [10 points] Consider the following nondeterministic finite automaton (NFA):



Convert the NFA to a DFA (deterministic finite automaton). Include a table showing, for each reachable set of NFA states, the corresponding DFA state.

NFA states	DFA state
{0}	0
{0, 1, 4, 5}	1
{2, 6}	2
{0, 4}	3
{0, 3, 4}	4
{2}	5
{3}	6



Question 6. [10 points] Specify a context-free grammar (CFG) that generates all strings of terminal symbols chosen from **a [ ] ( )** where the square brackets and parentheses are properly balanced.

Examples of strings in the language:

- ϵ
- **a**
- **[a]**
- **(a)[a]**
- **a((a))[a]**

Examples of strings not in the language:

- **[a)**
- **([a]**
- **)a[a]**
- **[(a])**
- **a][a**

Hint: make sure delimiters (brackets and parentheses) are added in balanced pairs.

Be sure to specify which nonterminal symbol is the start symbol. Each production should have a single nonterminal symbol on the left hand side.

start symbol is E

$E \rightarrow \epsilon$

$E \rightarrow EE$

$E \rightarrow (E)$

$E \rightarrow [E]$

Question 7. [20 points] Consider the following context free grammar with the terminal symbols `( ) λ x y f .` and the nonterminal symbols `E V`, where E is the start symbol:

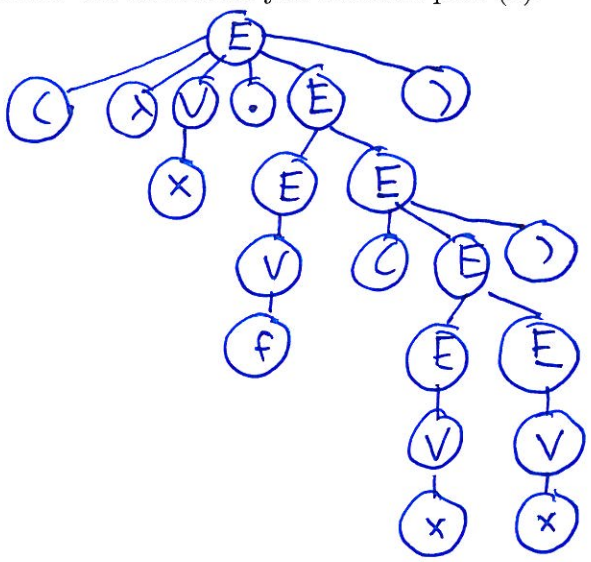
E → (λ V . E)
E → E E
E → (E)
E → V
V → **x**
V → **y**
V → **f**

(a) Show a derivation of the input string `( λ x . f ( x x ) )`. (Continue on the right-hand side if necessary.)

String	Production
E	E → (λ V . E)
(λ V . E)	V → x
(λ x . E)	E → E E
(λ x . E E)	E → V
(λ x . V E)	V → f
(λ x . f E)	E → (E)
(λ x . f (E))	E → E E
(λ x . f (E E))	E → V

String	Production
(λ x . f (V E))	V → x
(λ x . f (x E))	E → V
(λ x . f (x V))	V → x
(λ x . f (x x))	

(b) Draw the parse tree for the derivation you found in part (a).



Question 8. [10 points] Consider the following productions in a context-free grammar (CFG):

R → S **a** T R
R → S

Assume that **a** is a terminal symbol, and R, S, and T are nonterminal symbols. Show the pseudo-code for a recursive descent parse function for the nonterminal R (*i.e.*, a `parseR` function).

Important: Show how to use the lexical analyzer to choose between the two possible productions. Also, show how to raise an error if neither of the productions can be applied.

```
parseR() {
    parseS()
    t = lexer.peek()
    if (t != null && t == 'a') {
        expect('a')
        parseT()
        parseR()
    }
}

expect(tok) {
    t = lexer.next()
    if (t != tok) {
        throw Exception("Unexpected token" + t)
    }
}
```

- assume other parse functions will throw an exception if a parse error occurs
- assume lexer.next() will throw an exception if end of input is reached

Question 9. [10 points] Create a Turing Machine that takes a string of **a** and **b** symbols as input, and as output produces a reversal of the original string. For example, if the input string is **babbaba**, then the output string (the symbols on the tape when the Turing Machine halts) should be **ababbab**.

Hint: The output string does not need to be in the same place as the original string.

Hint: You may find it useful to replace the symbols in the input string with **A** and **B** as part of the reversal process.

Make sure that your start state and final state(s) are indicated, and that each transition specifies an input symbol, output symbol, and direction.