1. Regular Expressions and Finite Automata

a. The following grammar describes a language that is regular:

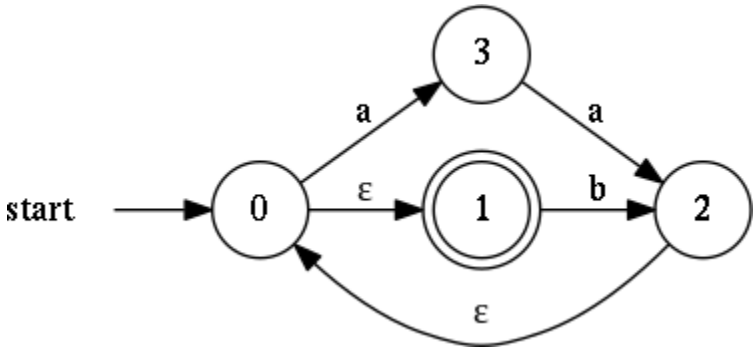
$$E \rightarrow T\ b\ E \mid T$$
$$T \rightarrow a$$

Create a regular expression that matches this language. For full credit, use only the core notations: ε, a, b, AB, A|B, (A), and A* **[6 points]**

Answer: _____

b. Consider the regular expression **c | (a*b)**. Using the method from lecture, construct a NFA that matches the same languages. **[6 points]**

c. Consider the following NFA.



Using subset construction, from lecture, create an equivalent DFA. **[8 points]**

d. Consider the following flex-like lexical specification [**6 points**]:

```
a*b       { print "1" }
ca        { print "2" }
a*ca*     { print "3" }
```

Given the following input string `abcaacacaaabbaaabcaaca`

What does the lexer print?

Answer: _____

2 LL Parsing

Consider the following grammar with terminals n, x, y, -, =, [, and].

A → BC C → n = A | n = y | D

B → x - A | ε D → n | [A A]

(a) The grammar is not LL(1). Explain in one sentence why. **[2 points]**

(b) Fix the grammar to make it LL(1) by filling in the blanks below. You may not need all the blanks. **[4 points]**

A → BC D → _____

B → x - A | ε E → _____

C → _____ _____

(c) [Longer] Compute the first and follow set of the fixed grammar. **[8 points]**

	First	Follow
A	x, n, [	
B		
C		
D		
E		
F		n, x, [,], $

(d) Finish the LL(1) parsing table. **[10 points]**

	n	x	y	-	=	[	]	$
A	B C							
B								
C								
D						[A A]		
E								
F								

3 LR Parsing

Consider the following grammar with terminals *, o, and b.

$$S \rightarrow A$$
$$A \rightarrow A* \mid A\ o\ A \mid b$$

a. Draw the parse tree(s) for "**bob***". **[6 points]**

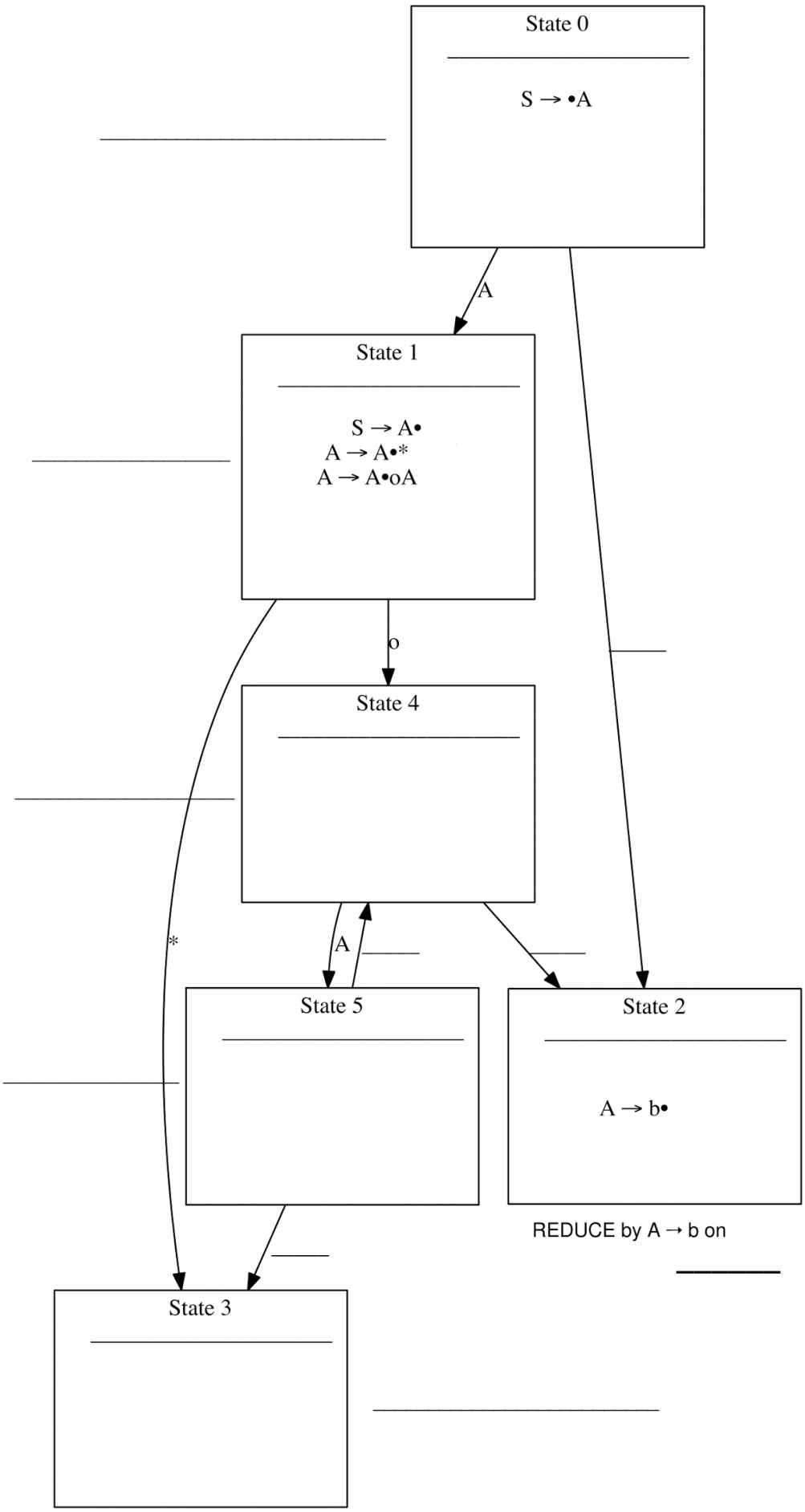
b. Is the above grammar ambiguous? Why or why not? **[2 points]**

c. **[Long]** Complete the parsing SLR(1) DFA below. **[16 points]**

i. Fill in the item sets for states **3**, **4**, **5**, and the ***rest*** of **0**. State **0** is a good place to start.

ii. Fill in all of the missing transition labels on solid lines.

iii. Write the necessary "**reduce-by**" and "**accept"** labels on **all** states on provided solid lines (State **2** has an example of a "**reduce-by**" label). Not all states will need a label. In the reduce-by label you should specify on which lookahead tokens the reduction can occur.


State 0
S → •A
A
State 1
S → A•
A → A•*
A → A•oA
o
State 4
*
A
State 5
State 2
A → b•
REDUCE by A → b on
State 3

d. [Depends on c] For each state with a conflict, fill in the state, the lookahead token, and the type of conflict (i.e., shift/reduce conflict, or reduce/reduce conflict). Fill in as many spaces as necessary - you may not need all of them. **[4 points]**

State _____, **on** _____, _____ conflict.

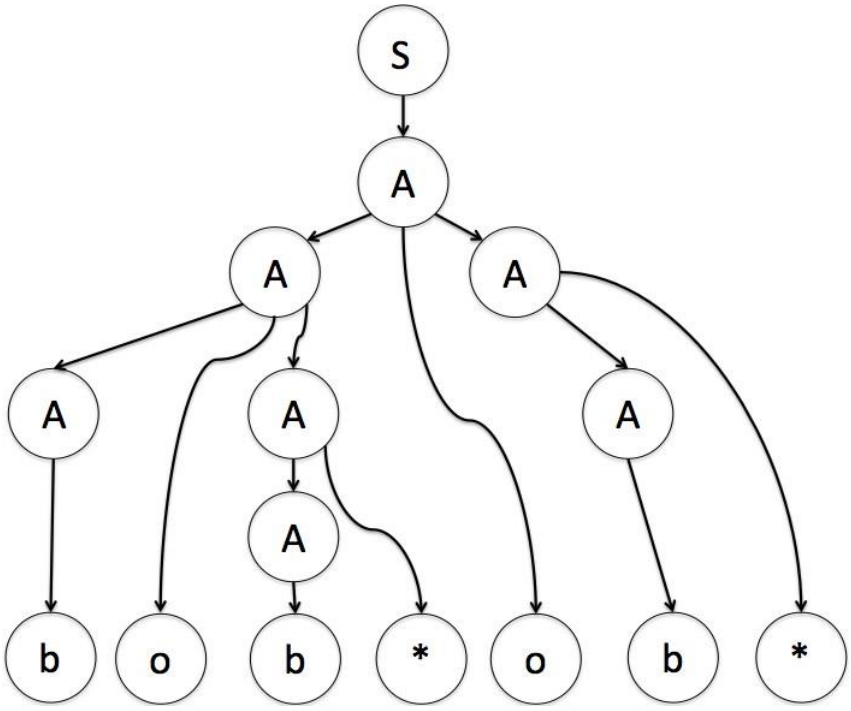
State _____, **on** _____, _____ conflict.

State _____, **on** _____, _____ conflict.

State _____, **on** _____, _____ conflict.

State _____, **on** _____, _____ conflict.

Suppose we want the string "**bob*ob***" to have only the following parse tree (call this property P).



e. Describe in English the precedence and associativity rules necessary to ensure property P. **[4 points]**

_____ is _____-associative and has _____ precedence than _____