**CS 330 Mid Term Key**

INSTRUCTIONS:

* Do not begin or open this test until directed to do so!
* This test is closed book, closed notes, and no computer access is allowed. However, you are allowed **two letter-size (8.5X11) cards with handwritten** notes on both sides.
* Absolutely no collaboration. Any violations of this policy may result in a 0 on the test.
* Carefully read all problems and only do the number of questions as required. Don’t do more than asked for.
* Questions may contain more than one part.
* You must show your work where applicable!
* The test’s time limit is **75 minutes**. Be sure to manageyour time appropriately.

GRADING BREAKDOWN:

|  |  |  |
| --- | --- | --- |
| Part | points possible | points earned |
| I | 17 |  |
| II | 56 |  |
| III | 30 |  |
| Total | 103 |  |

Additional Curve: +3

Class distribution:

30-59: 10

60-69: 4

70-79: 3

80-89: 8

90-99: 7

100 – 104: 3

**Part I. Matching** – Fill in each blank with the **letter** for an answer that ***best*** fits the definition or completes the sentence. ~~Note that NO answer should be used more than once~~, except perhaps EE. (1x16=17 points)

A. adaptive B. automatic C. bottom-up D. one

E. factoring F. finite G. handle H. Left to right

I. look ahead J. left-recursive K. pop M. Nondeterministic

N. parsing O. predict P. push down Q. production

R. reverse S. yacc T. reduce U. regular

V. rewind W. ANTLR4 X. shifting Y. scanning

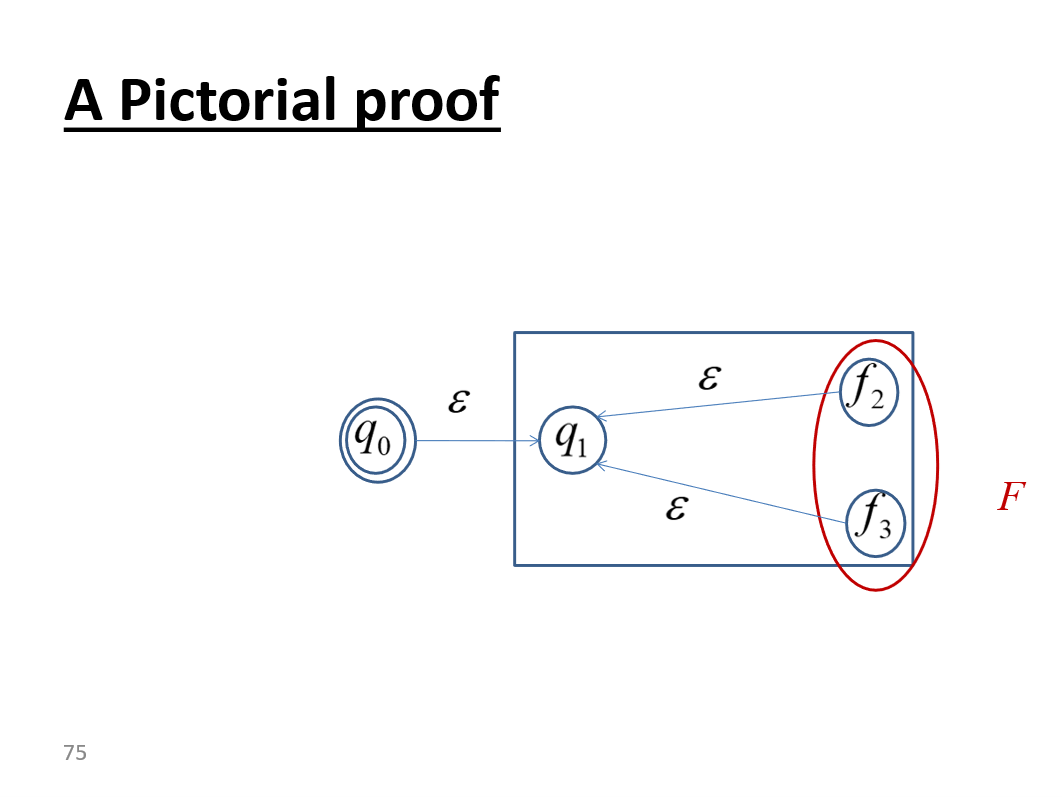
Z. item AA. top-down BB. multiple CC. Terminal

DD. token EE. None of the above

1. \_\_\_\_Y\_\_ is usually the first step of program compilation. It divides a program into \_\_\_DD\_s, which are the smallest meaningful units of a program.
2. LL parsers are generally referred to as \_\_\_\_AA\_\_ parsers. During parsing process, they continuously select the appropriate grammar rule to expand based on the current non-terminal and the current input token(s), until either a parse tree is generated where the leaf nodes corresponding to the input or it is stuck. To facilitate this process, LL parsers pre-calculate a \_\_\_O\_\_\_\_ set for each unique production. LL(1) parsers can be represented by a PDA with \_\_\_\_D\_\_\_ major state(s).
3. Many LL parser generators require unambiguous grammar such as LL(1), where a unique production can be identified with \_\_D (or EE)\_\_\_ token look-ahead, or LALR(1), which is accepted by \_\_S\_\_. \_\_W\_\_\_, on the other hand, can handle many different types of grammar using a technique called **A**LL(\*) where the A indicates the \_\_\_A\_\_\_ nature of the algorithm, which Parr refers to as being a honey badger.
4. LR parsers, on the other hand, create the parse tree \_\_\_C\_\_\_. Specifically, they attempt to \_\_T\_\_\_ the input token stream into the start symbol by repeatedly \_\_X\_\_\_ each new token onto a stack, and then check to see if symbol(s) at the top of stack match the Right-Hand Side (RHS) of a grammar rule. If a match is found, LR parser will pop and replace the matched symbol(s) (referred to as \_\_G\_\_\_\_ in this situation) with their corresponding Left Hand Side (LHS) of the grammar rule. The actions of an LR parser trace out a rightmost derivation in \_\_\_R\_\_\_ order. To facilitate LR(1) parsing, dots are placed in RHS of grammar rules, forming a(n) \_\_\_Z\_\_\_\_. while LR(1) parsers are mapped to PDA with \_\_BB\_\_\_\_\_ state(s).
5. Name any (your favorite?) tech company CEO \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

**Part II. Regular Language**

1. Give the outline of a proof (by drawing the outline of generic FAs) for the statement “Regular language is closed under the star operator”, (i.e., if R is regular, then R\* is also regular). (6)



1. For each of the following languages on alphabet ∑ = {0, 1}, give two strings that are member and two that are NOT. (4X2=8)
2. 0\*1\*

Member: 1, 0, 01,

Non: 10, 101

1. 1∑\*0∑\*1 // mid is 0

Member: 101, 10011

Non: 11, 1101 // or anything with no 0

1. Give regular expressions for the following languages on alphabet {0, 1}. (6X3=18)
2. Words that do not contain the substring 00.

// in other words, every 0 is followed by 1 or more 1s

1\*(01+)\*

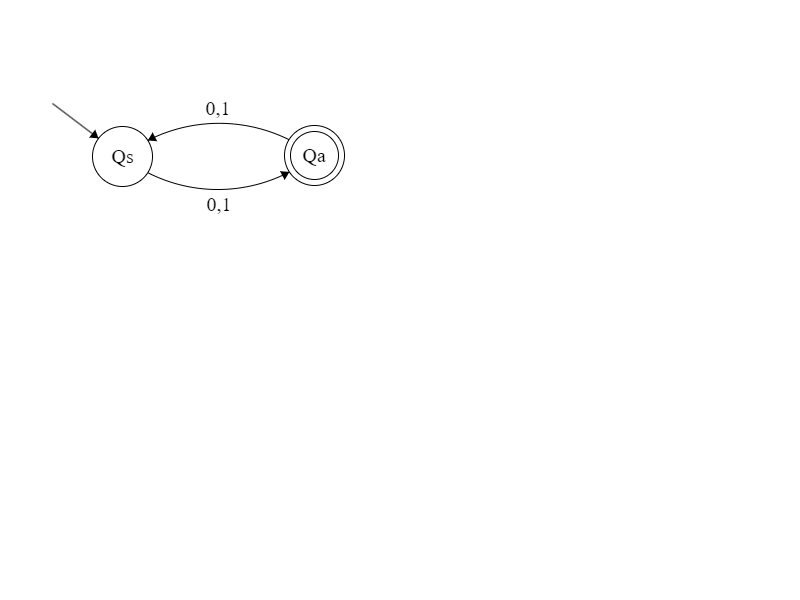
1. Words that contain exactly two 0s.

1\*01\*01\*

1. Words begin and end with the same digit.

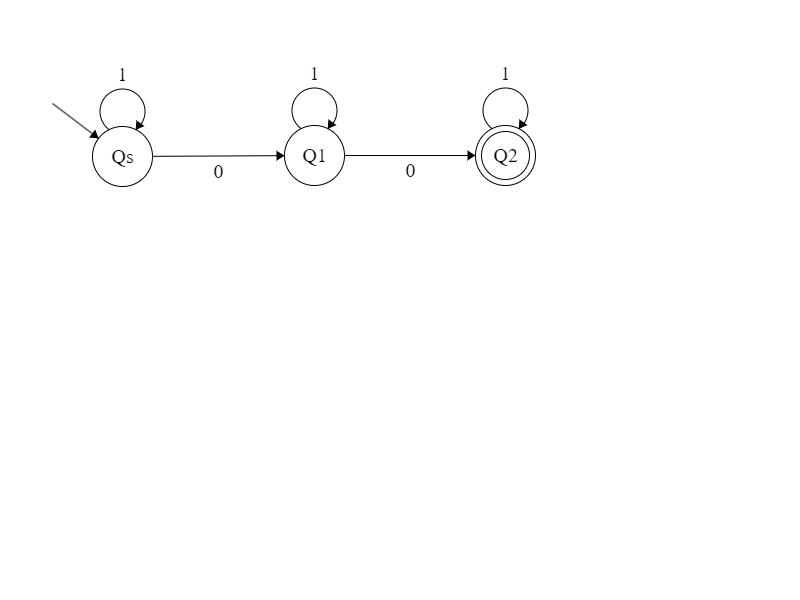
1∑\*1|0∑\*0|1|0

1. Give FA (either DFA or NFA) on alphabet {0, 1} that represents/recognizes for the following languages. **Do any 4 of the 6.** Be sure to indicate which problem each of your answers is for. (4X6=24 points)
2. The language containing words of odd length. ∑(∑2)\*



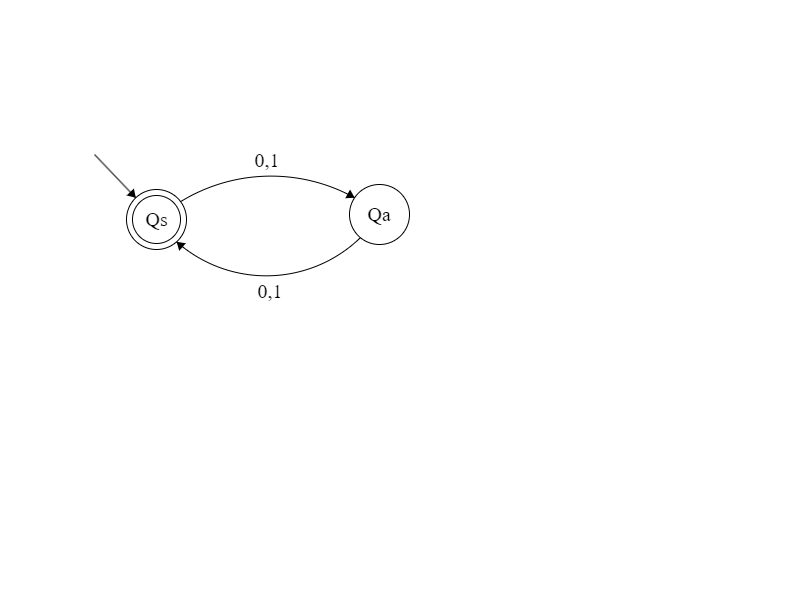
1. The language containing words with exactly two 0s: 1\*01\*01\*

NFA



1. The complement of the language described in A.

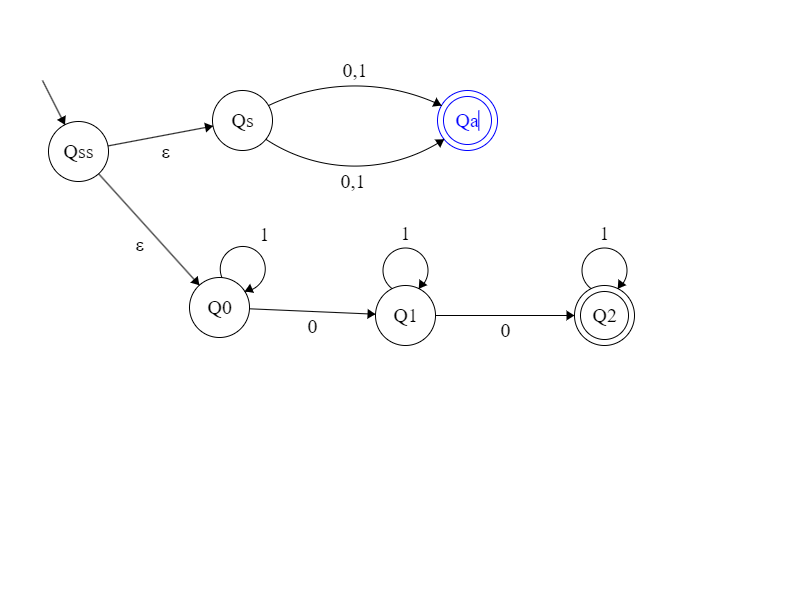
Complement requires original to be DFA which our A is -> swap states



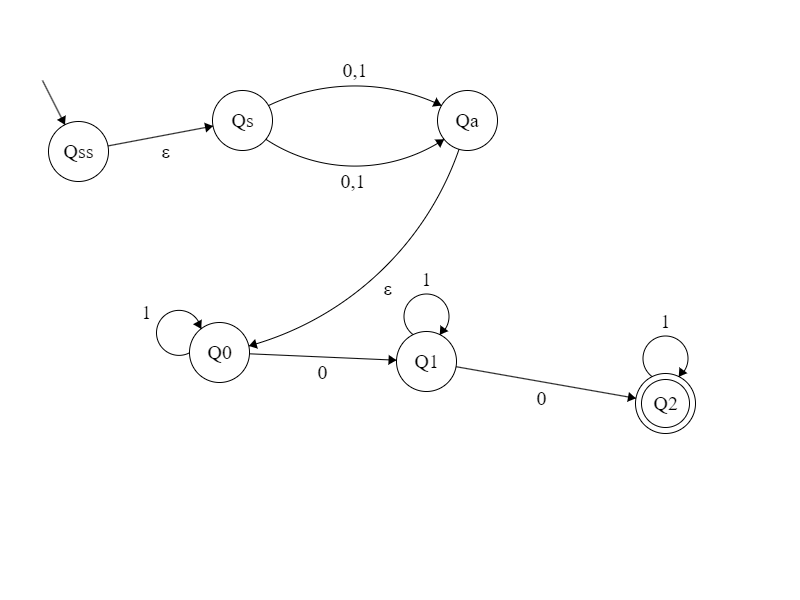
1. Reverse of the language described in B.

No change

1. Union of languages A and B: add new start state with ε transition to starts of A and B



1. Concatenation of the languages A and B: accept of A epsilon to start of B



**Part III. CFL and PDA**

1. Expression grammar - consider the following CFG:

E🡪 E \* E | E - E | E^E | ID

ID 🡪a-z

Where *E* and ID are non-terminals with E being the start symbol, and *-, \*, ^* and letters *a* through *z* are terminals. Do each of the following. (4+4+6=14)

1. Show this language is ambiguous by drawing two parse trees for the same expression of your choice. Be sure to show the expression along with the trees. Note that you can choose any expression as long as it (a) belongs to the language, and (b) has 2 or more parse tree representations.

Just show 2 parse trees for the same sentence ex., a-b\*c

-1 for using operators that are not part of this grammar, ex., +

-3 for wrong tree or two trees for 2 different sentences

1. Modify the grammar so that (a) it is left associative for \* and -, right associative for ^ (b) it has the **correct operator precedence** with ^ over \* over -.

**E -> E-T | T**

**T -> T\*F | F**

**F -> B^F | ID**

**ID -> a..z**

1. Further modify the grammar to make it LL(1)

**E -> T TT // Term and TermTail**

**TT -> - T TT | ε**

**T -> F FT**

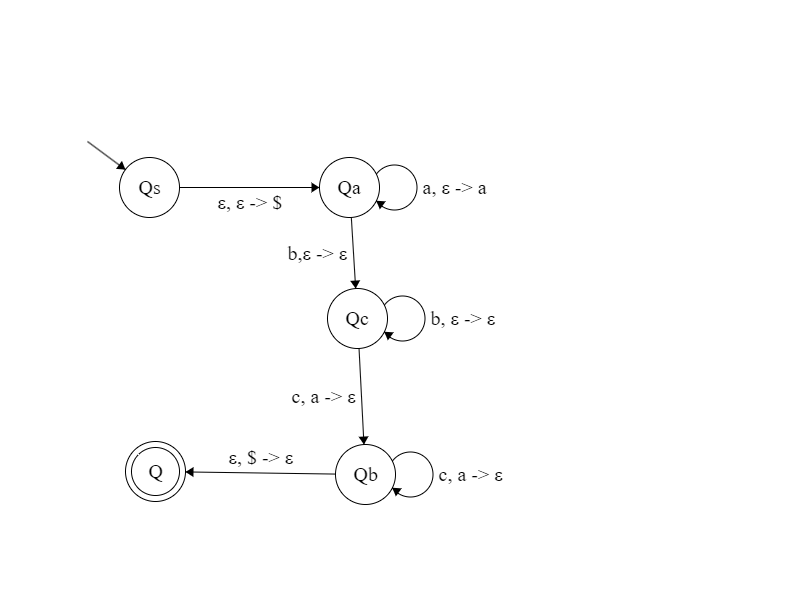
**FT -> \* F FT | ε  
F -> B^F | ID**

**ID -> a..z**

1. Write CFG and draw PDA for **either one of the following two** languages on alphabet {a, b, c}. Be sure to indicate which language your CFG and PDA are for. (8x2=16)
2. The language of ambncm, where m and n are both positive integers.

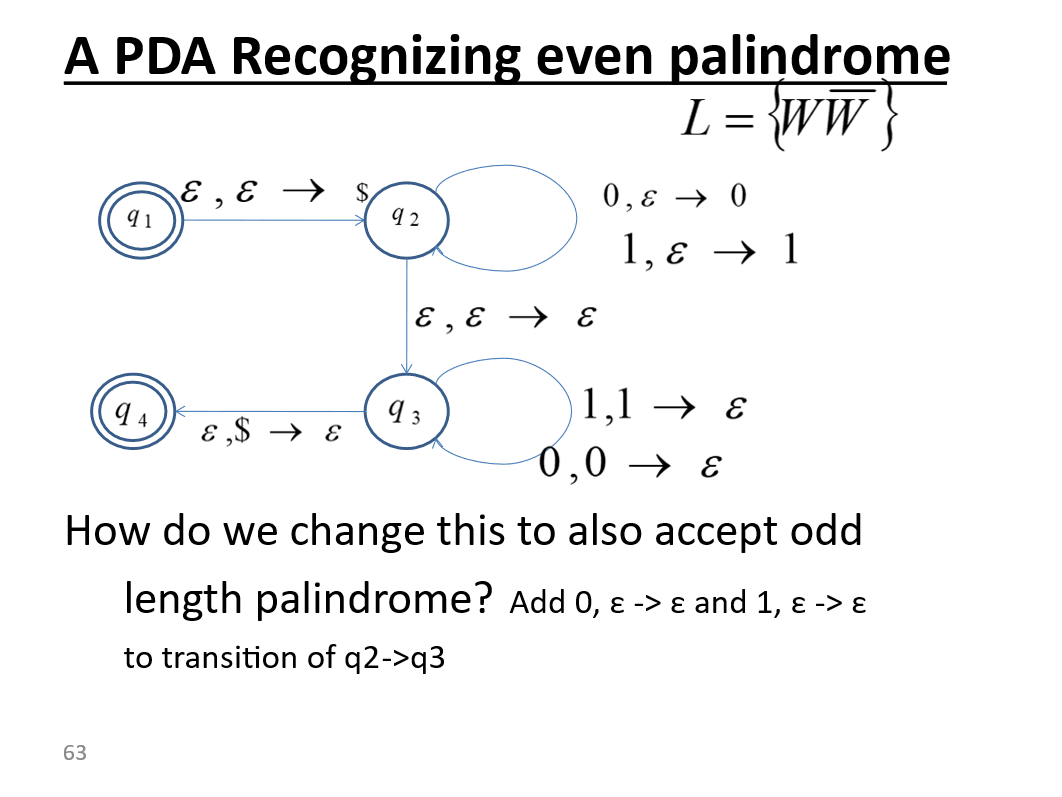
S -> aSc | aBc

B -> bB | b



1. The language of odd length palindromes with at least 3 letters and the middle 3 being cac.

S -> aSa | bSb | cSc | cac



Modify the above PDA by changing q2-q3 transtion to a sequence of transitions:

Q2 c,ε->ε Q21 a,ε->ε Q22 c,ε->ε Q3

The sequence essentially discards cac.