

CS 3240 Project

Notes:

1. This project has two phases. Phase 1 is due by April 14th by 5pm. Phase 2 is due by April 28th by 5pm.
2. There will be no extensions for either phases
3. You will work in groups of three
4. Each group should submit a report and source code for each phase. If multiple source files, they must be tarred along with the makefile
5. You can program in C, C++ or Java. Do not use tools (like lex and yacc) or the standard template library
6. Code should be properly documented with meaningful variable and function names. Short elegant code will get bonus points.
7. You will find the course slides on DFA/NFA/scanner/recursive descent parser useful.
8. Each phase of the project is worth 100 points. The bonus section is worth 50 points.

Phase 1:

Objective: To write a scanner and parser which can construct and execute an NFA for any regular expression.

Consider the language of regular expressions. The alphabet of this language is the set $\{a, b, *, +, (,), ., | \}$ (commas and spaces are not part of the language). Using this alphabet one can write any regular expression. Our goal in this project is to be able to read any regular expression described by the following grammar and construct primitive NFAs and join them together to form a NFA that will recognize strings described by the regular expression. We will do this step by step by developing answers to the following questions. The production rules for this language are given by

$R \rightarrow R^*$

$R \rightarrow R^+$

$R \rightarrow (R)$

$R \rightarrow (R | R)$

$R \rightarrow R.R$

$R \rightarrow a$

$R \rightarrow b$

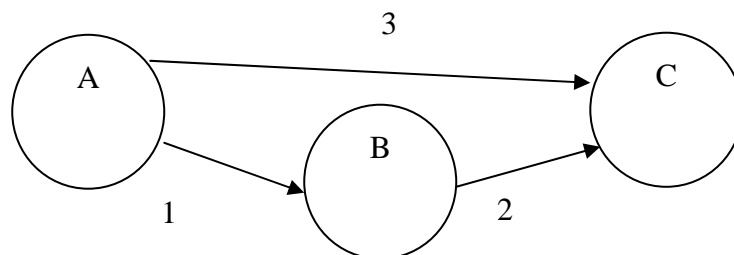
Question 1: Rewrite the grammar to remove left recursion.

Question 2: Identify the tokens of this language and write a scanner program which can scan this language and return tokens .

Question 3: Write a *recursive descent* parser which can parse this language (based on the modified grammar which removed left recursion) and yield a parse tree. Note that this grammar has implicit precedence. That is for a regular expression, $a.b^*$ the “*” operates on “b” and not $a.b$ as a whole. This is true unless it is bracketed. In, $(a.b)^*$ on the other hand, the “*” operates on $(a.b)$ When you build a parse tree you must take care of such precedences

Question 4: Now you need to write a program which can construct a NFAs based on the parse tree based on primitive NFAs. As discussed in class, primitive NFAs should be joined together to form NFA for the complete regular expression. This final NFA will be represented as an adjacency matrix described below. Thus the output of this program should be an *adjacency matrix*.

Adjacency matrix: Any NFA is a *directed graph*. A directed graph G consists of a set of *nodes* (in our case states) and *directed edges* (in our case, transitions). For example, in the graph below, A,B,C are nodes and 1,2,3 are edges



Any directed graph can be represented by an adjacency matrix. For example, the matrix below represents the graph. Since edge “1” connects A to B, there is a “1” in the row corresponding to “A” and the column corresponding to “B”.

	A	B	C
A		1	3
B			2
C			

Similarly an NFA can be represented by an adjacency matrix. Note that more than one element can be present in a cell. For example, in the NFA if the edge from A to B is labeled a,b then you would have both “a” and “b” in the corresponding cell.

Question 5: Given such an adjacency matrix of an NFA and given an input string consisting of a’s and b’s write a program to *simulate* the NFA and output if the string is accepted or rejected. Note : NFAs can progress on multiple paths and you should simulate this effect – if one of the paths results in accept state then the input string is accepted by NFA.

Phase 2: To write a program which will construct a DFA from any NFA. You will use adjacency matrix as the representation and use epsilon closures to generate DFA. Finally write a program to simulate the DFA.

Bonus: Given an adjacency matrix for a DFA, write a program to produce minimal DFA by state merging.