**Compiler Homewrok #1**

21400 ...... Park**,** 21400549 Seunghwan Lee

1. **Program Definition**

Our program receives a simple regular expression and a sequence of strings and determines whether each string is in the language or not by creating a DFA from regular expression, and checking whether input string reaches the accepting state through the DFA.

1. **Develop Environment**

Developed using c ++ in Windows environment.

1. **Program instruction process**
2. Enter a regular expression from the user.
3. Determines whether the input regular expression is valid or not.
4. If it is invalid, enter the regular expression again.
5. If it is valid, create an epsilon-NFA through a regular expression and use it to create DFA.
6. And, Enter a sequence of strings from the user.
7. Then the string goes into DFA and its state is converted.
8. After that program checks the states and determines whether the input string is in a language or not.
9. Repeat the above procedure.
10. **How it works**
11. **Check regular expression**

stack ,,.... 화이팅...

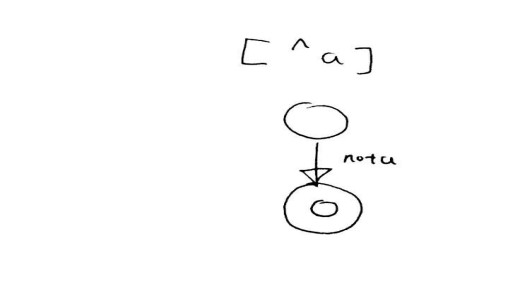
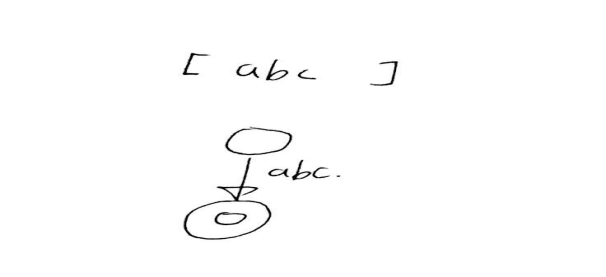
1. **Regular expression to epsilon-NFA**

We implemented epsilon-NFA using graphs. First, we assume that there is no invalid case through the results of the above step. We read the input regular expression and determine the operator. Based on the postfix and recursive logic, it generates an appropriate graph according to the operator. The following figures show which graphs are created for each operator.

1. " [ ] "

If we read the "[" character, we have only two cases.

The first is "[abc]", The second is "[^a] .

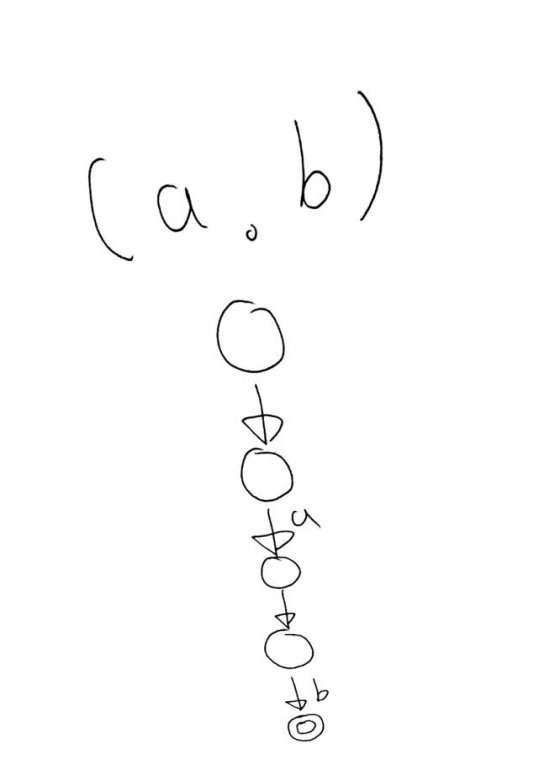
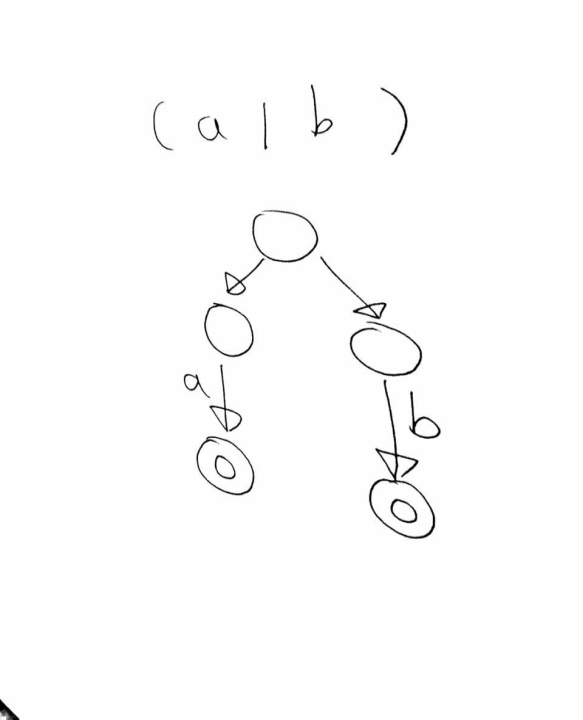
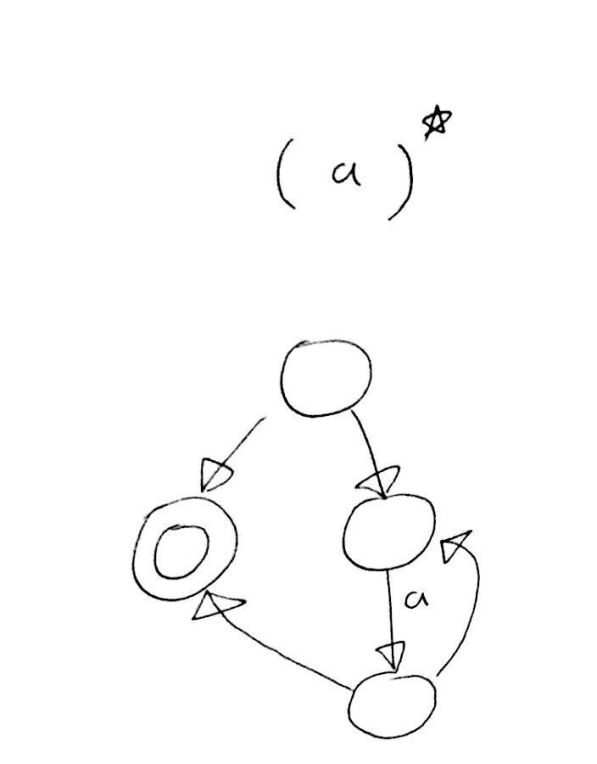


So if "[" character followed by "^" character is thought to be the first case, otherwise it is the second case, and one of the above two graphs is made.

1. " ( ) "

If we read the "(" character, we have only three cases.

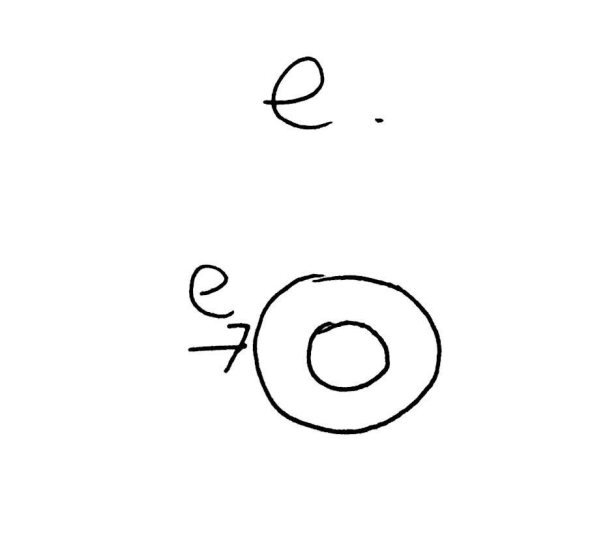
The first is "()\*", the second is "(|)" the last is "(.)"



First we read regular expression immediately following the '(' character.. Then, if the first remaining character is ")", it is considered to be ()\* case. Secondly, if the first remaining character is "|", it is considered to be (|) case. Finally if the first remaining character is ".", it is considered to be (.) case. Then if necessary, read the remaining regular expressions and combine the graph with the results. Eventually one of the three graphs above is created.

1. "Epsilon"

If we read "e" character, we have only one case.



1. Mixed

Because the above three cases are mixed in various forms, recursive implementations will continue to build graphs. Eventually, once all the process is over, a single epsilon-NFA is created

1. **epsilon-NFA -> DFA**
2. eclose..?

how to ?

1. convert?

how?

1. convert2?

how?

1. Set accepting states in DFA

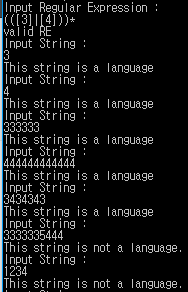
DFA is completed when the above process is over. However, the DFA does not yet have an accepting state set. Therefore, the accepting state must be set.

Fist, we finds the state number of the accepting state of epsilon-NFA. Then, after confirming that the each state of the DFA includes the state number of the accepting state of epsilon-NFA, all applicable states are called accepting state of DFA.

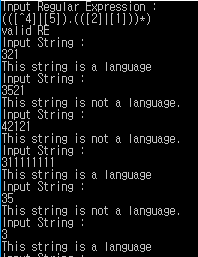
1. **Check whether the input string is a language.**

We change the state by inserting the input string into the DFA table one by one. Make sure that the state converted to the last string input is the accepting state.

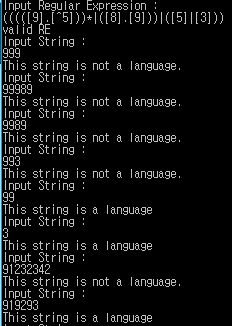
1. **Program Test**
2. (([3]|[4]))\*



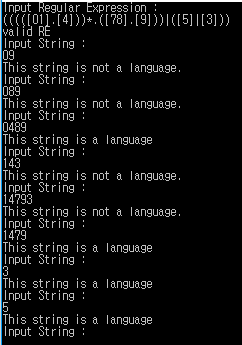
1. (([^4]|[5]).(([2]|[1]))\*)



1. (((([9].[^5]))\*|([8].[9]))|([5]|[3]))



1. (((([01].[4]))\*.([78].[9]))|([5]|[3]))



1. **How to build? (Read Me?)**
2. cmd를 켠다.
3. make..

.