

AUTOMATA FORMAL LANGUAGES AND LOGIC



Lecture notes on Converting Regular Expression to Finite Automata

Prepared by

**Ms. Sangeeta V I
Assistant Professor**

**Ms. Preet Kanwal
Associate Professor**

Department of Computer Science & Engineering

PES UNIVERSITY

**(Established under Karnataka Act No.16 of 2013)
100-ft Ring Road, BSK III Stage, Bangalore - 560 085**

Table of Contents:

Section	Topic	Page number
1	Regular Expression to Finite Automata	3
1.1	Basic Regular Expressions to NFA/ λ -NFA	3
1.2	Higher End Problems on converting Regular Expression to Finite Automata	8

Examples Solved:

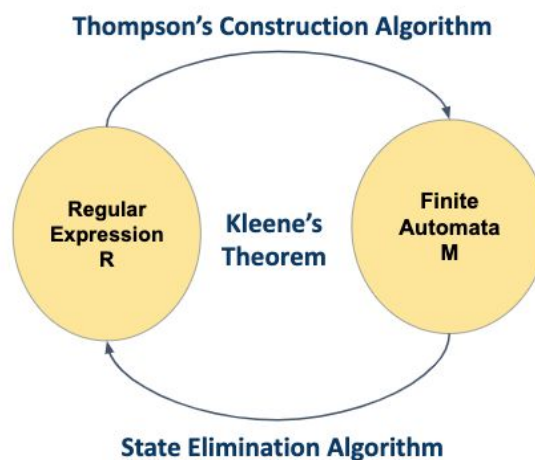
#	Problems on converting Regular Expression to finite automata	Page number
1	Convert the regular expression $a^*+b^*+c^*$ to finite automata.	8
2	Convert the regular expression $(aa)^*(bb)^*+a(aa)^*b(bb)^*$ to finite automata.	10
3	Convert the regular expression $(1+01)^*00(1+10)^*$ to finite automata.	13
4	Convert the regular expression $(0+\lambda)(1+\lambda)(1+2)^*0(2+1)^*$ to finite automata.	14
5	Convert the regular expression $((a+b+c)c)^*(a+b+c+\lambda)$ to finite automata.	15

1. Regular Expression to Finite Automata

Regular expressions and finite automata have equivalent expressive power:

Many software tools work by matching regular expressions against text. Text processing utilities use regular expressions to describe advanced search patterns, but NFAs are better suited for execution on a computer.

The equivalence of regular expressions and finite automata is known as Kleene's theorem. We use Thompson Construction algorithm to convert a RE to NFA. This algorithm is of practical interest, since it can compile regular expressions into NFAs. We use a State Elimination method in order to convert a finite automata to Regex.

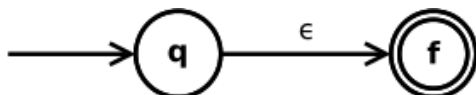


Converting regular expressions to finite automata using Thompson Construction Method:

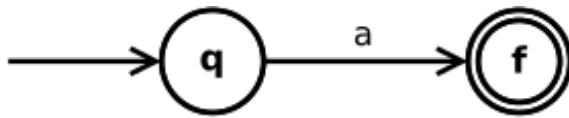
The algorithm works recursively by splitting an expression into its constituent subexpressions, from which the NFA will be constructed using a set of rules.

Following are the rules :

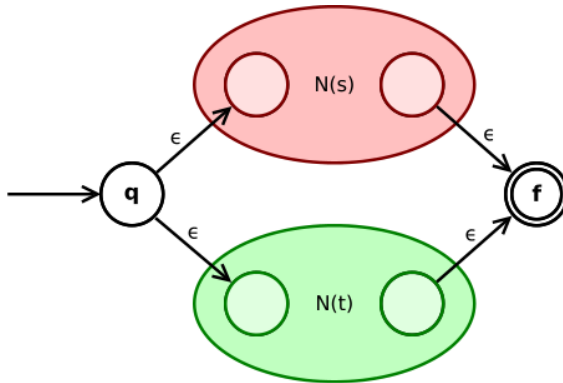
- If the operand is epsilon, then our FA has two states, q (the start state) and F (the final, accepting state), and an epsilon transition from q to F.



- If the operand is a character a, then our FA has two states, q (the start state) and F (the final, accepting state), and a transition from q to F with label a.

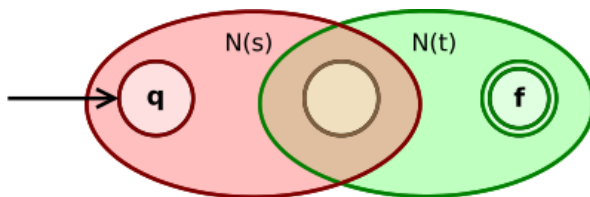


- The **union expression** $s|t$ is converted to



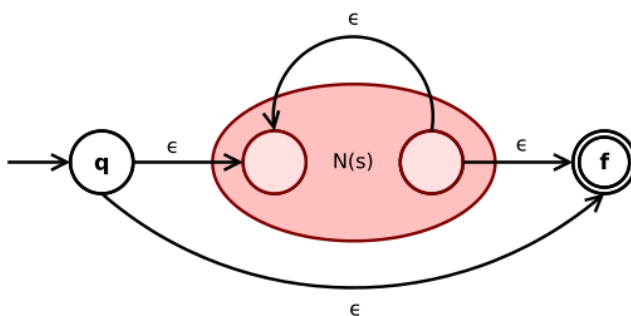
State q goes via ϵ either to the initial state of Automata of s and t . $N(s)$ or $N(t)$. Their final states become intermediate states of the whole NFA and merge via two ϵ -transitions into the final state of the NFA.

- The **concatenation expression** st is converted to



The initial state of $N(s)$ is the initial state of the whole NFA. The final state of $N(s)$ becomes the initial state of $N(t)$. The final state of $N(t)$ is the final state of the whole NFA.

- The **Kleene star expression** s^* is converted to



An ϵ -transition connects the initial and final state of the NFA with the sub-NFA .

$N(s)$ Automata for s in between. Another ϵ -transition from the inner final to the inner initial state of $N(s)$ allows for repetition of expression s according to the star operator.

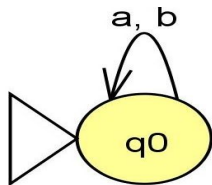
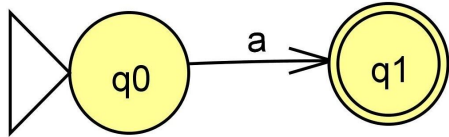
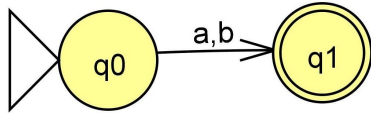
- The **parenthesized expression** (s) is converted to $N(s)$ itself.

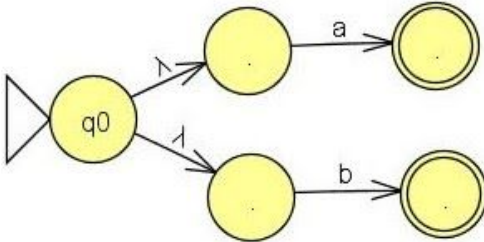
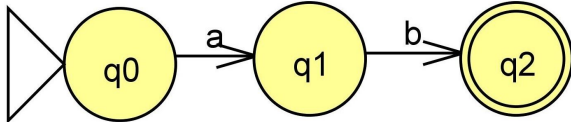
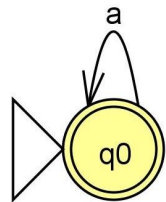
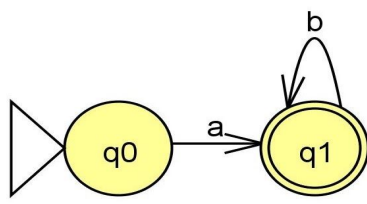
With these rules, using the **empty expression** and **symbol** rules as base cases, it is possible to prove with mathematical induction that any regular expression may be converted into an equivalent NFA.

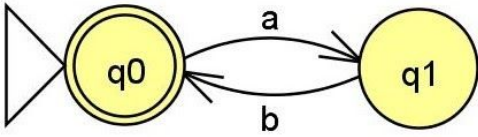
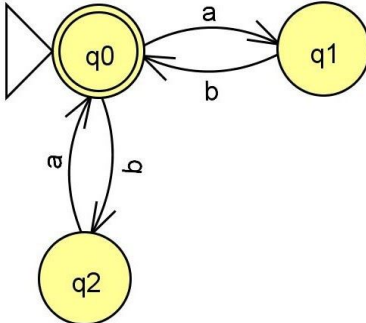
1.1. Basic Regular Expressions to NFA/ λ -NFA

Let us start with some basic regular expressions.

Assume $\Sigma = \{a, b\}$.

Regular Expression	NFA/ λ -NFA
ϕ	 <p>ϕ represents not accepting anything that is, no accepting/final state . In state q_0, any input symbol is rejected.</p>
a	 <p>In state q_0, we will wait for input symbol 'a'. Once we see 'a' in state q_0 we will move to state q_1 and accept it.</p>
$a+b$	 <p>We accept either a or b. In state q_0, if we see 'a' we move to state q_1 and accept it or</p>

	<p>if we see 'b' we move to state q1 and accept it.</p> <p>a+b can also be represented by treating a and b as two different regular expressions ,we get the automata as shown below:</p> 
a.b	 <p>a followed by b and accept the string.</p>
a*	 <p>a* represents any number of 'a's including λ (empty string). So we make q0 as the final state. When we see * we put a self loop.</p>
ab*	 <p>ab*. 'a' is the minimum string.</p> <p>In state q0 we look for the symbol 'a' . Once we see the symbol 'a' in q0 we move to state q1 and</p>

	<p>accept it .</p> <p>In state q1 we can accept any number of b's. When we see a * we put a loop.</p>
$(ab)^*$	 <pre> graph LR start(()) --> q0((q0)) q0 -- a --> q1((q1)) q1 -- b --> q0 q0 --> accept((())) </pre> <p>$(ab)^*$=sequences of $ab = \lambda, ab, abab, ababab, \dots$</p> <p>We look for 'a' followed by a 'b' . In state q0 we look for 'a' and move to state q1. In state q1 we see a 'b' and move back to state q0 and accept it. Since there is a *,we make q0 as the final state.</p>
$(ab+ba)^*$	 <pre> graph TD start(()) --> q0(((q0))) q0 -- a --> q1((q1)) q1 -- b --> q0 q0 -- b --> q2((q2)) q2 -- a --> q0 </pre> <p>Strings accepted by automata=Sequences of "ab" and/or sequences of "ba".</p>

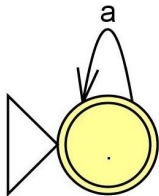
1.2: Higher end Problems on converting Regular Expression to Finite Automata

Problem #1:

Convert the regular expression $a^*+b^*+c^*$ to finite automata.

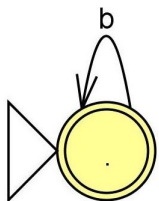
Solution :

Automata accepting a^* :



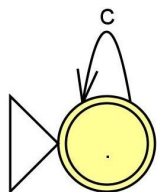
- a^* indicates any number of a's including λ .
- When we see $*$ we put a loop.

Automata accepting b^* :



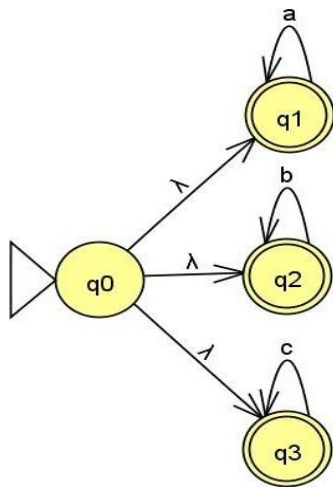
- b^* indicates any number of b's including λ .
- When we see $*$ we put a loop.

Automata accepting c^* :



- c^* indicates any number of c's including λ .
- When we see $*$ we put a loop.

We can combine the automata accepting a^* , b^* and c^* by introducing a new start state q_0 with λ ,-transitions connecting to the old start states of a^* , b^* and c^* .

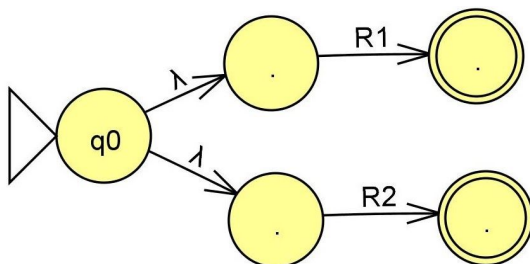


Problem #2:

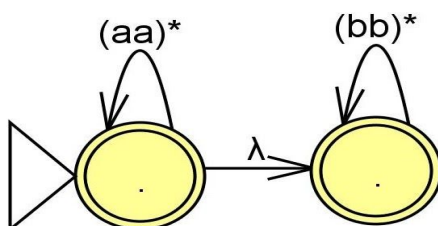
Convert the regular expression $(aa)^*(bb)^*+a(aa)^*b(bb)^*$ to finite automata.

Let,

- $R1=(aa)^*(bb)^*$
- $R2=a(aa)^*b(bb)^*$.
- Construct acceptor/automata for R1
- Construct acceptor/automata for R2.
- Introducing a new start state on λ we reach the start state of R1 and R2.

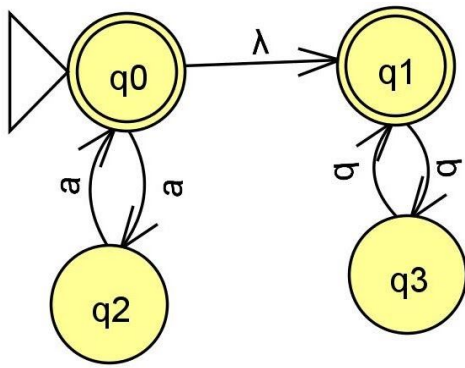


Brief diagram of automata for $R1=(aa)^*(bb)^*$



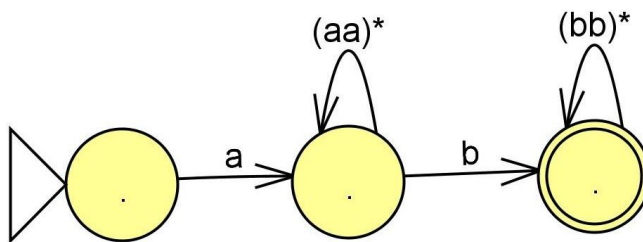
We look for sequences of any number of "aa" 's followed by sequences of any number of "bb"'s.

Expanded form of automata for $R1=(aa)^*(bb)^*$



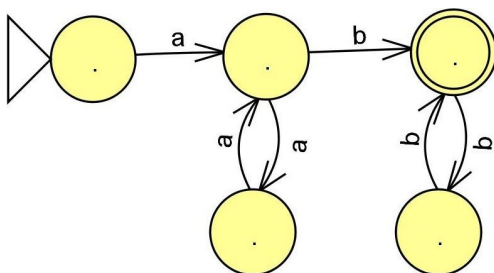
In state q_0 we look for 'a' followed by symbol 'a' or 'b' followed by symbol 'b'.

Brief diagram of automata for $R_2 = a(aa)^*b(bb)^*$



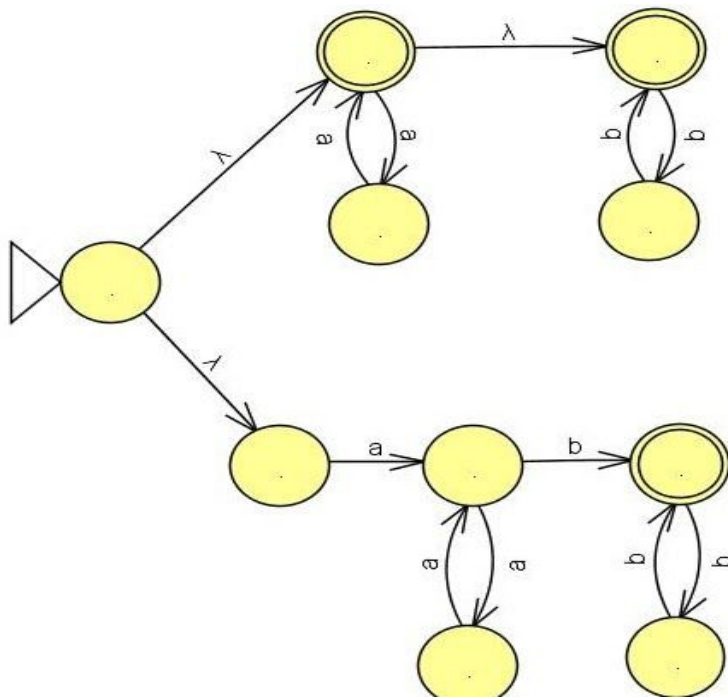
We look for "a " followed by any number of sequences of "aa" ,followed by a "b" and sequences of any number of "bb". When we see a star we put a loop.

Expanded form of automata for $R_2 = a(aa)^*b(bb)^*$



Automata for the regular expression $R_1 + R_2 = (aa)^*(bb)^* + a(aa)^*b(bb)^*$.

We will combine the automata for R_1 and automata for R_2 by introducing a new start state on λ we reach the start state of R_1 and R_2 .



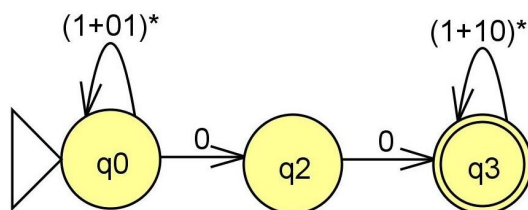
Problem #3:

Convert the regular expression $(1+01)^*00(1+10)^*$ to finite automata.

Solution :

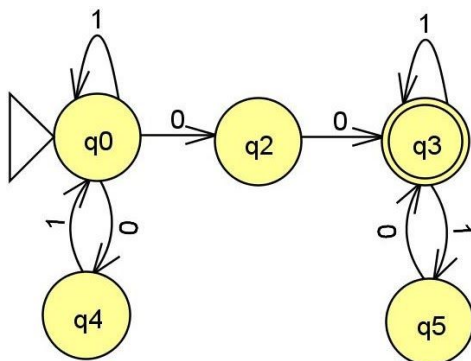
$(1+01)^*00(1+10)^*$ is the concatenation of $(1+01)^*$, 00 and $(1+10)^*$.

Brief diagram of automata for $(1+01)^*00(1+10)^*$



- When we see a $*$ we put a loop.

Expanded form of automata for $(1+01)^*00(1+10)^*$



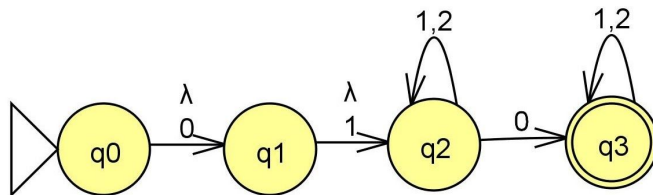
Problem #4:

Convert the regular expression $(0 + \lambda)(1 + \lambda)(1+2)^*0(2+1)^*$ to finite automata.

Solution :

$(0 + \lambda)(1 + \lambda)(1+2)^*0(2+1)^*$ is the concatenation of $(0 + \lambda)$, $(1 + \lambda)$, $(1+2)^*$, 0 and $(2+1)^*$.

Automata for $(0 + \lambda)(1 + \lambda)(1+2)^*0(2+1)^*$



When we see $*$ we put a self loop.

Problem #5:

Convert the regular expression $((a+b+c)c)^*(a+b+c+\lambda)$ to finite automata.

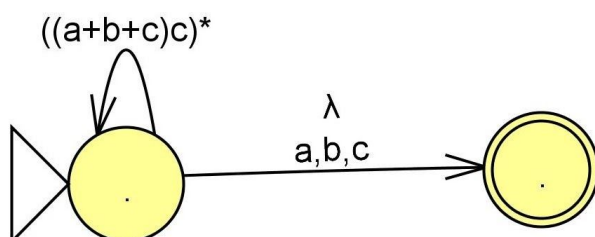
Solution :

$((a+b+c)c)$ indicates a or b or c followed by a c, these are strings of length 2.

$((a+b+c)c)^*$ strings of length 2 repeated any number of times.

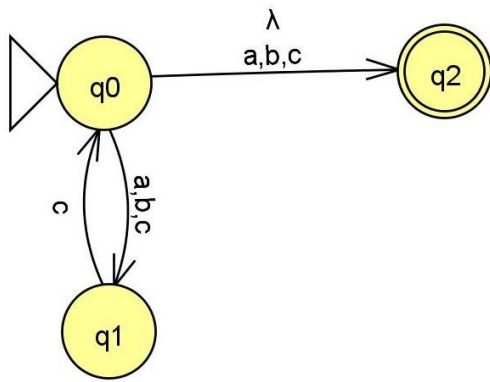
$((a+b+c)c)^*(a+b+c+\lambda)$ strings of length 2 repeated any number of times.

Followed by 'a' or 'b' or 'c' or nothing (λ) .



When we see $*$ we put a self loop.

Expanded form of automata for $((a+b+c)c)^*(a+b+c+\lambda)$



- In q_0 , when we see 'a' or 'b' or 'c' it has to be followed by "c". That is strings of length 2 ('a' or 'b' or 'c' followed by 'c') repeated any number of times.
- Followed by 'a' or 'b' or 'c' or nothing (λ).