Regular Expressions and DFAs

Regular Expression

- Notation to specify a language
  - Declarative
  - Sort of like a programming language.
    - Fundamental in some languages like perl and applications like grep or lex
  - Capable of describing the same thing as a NFA
    - The two are actually equivalent, so RE = NFA = DFA
  - We can define an algebra for regular expressions

Algebra for language

- Previously we discussed these operators:
  - Union
  - Concatenation
  - Kleene Star

Remember

* has a higher precedence than concatenation and concatenation has a higher precedence than +

RE Examples

$L(\mathbf{001}) = \{001\}$

$L(\mathbf{0+10^*}) = \{ 0, 1, 10, 100, 1000, 10000, \ldots \}$

$L(\mathbf{0^*10^*}) = \{1, 01, 10, 010, 0010, \ldots\}$ i.e. $\{w \mid w$ has exactly a single $1\}$

$L(\Sigma\Sigma)^* = \{w \mid w$ is a string of even length$\}$

$L((\mathbf{0(0+1)})^*) = \{ \varepsilon, 00, 01, 0000, 0001, 0100, 0101, \ldots\}$

$L((\mathbf{0+\varepsilon})(\mathbf{1+\varepsilon})) = \{\varepsilon, 0, 1, 01\}$

$L(1\emptyset) = \emptyset$; concatenating the empty set to any set yields the empty set.

$R\varepsilon = R$

$R+\emptyset = R$

Exercise 1

- Let $\sum$ be a finite set of symbols
- $\sum = \{10, 11\}$, $\sum^* = ?$

Answer:

$\sum^* = \{\epsilon, 10, 11, 1010, 1011, 1110, 1111, \ldots\}$

Exercises 2

- L1 = {10, 1}, L2 = {011, 11}, L1L2 = ?
- L1L2 = {10011, 1011, 111}

Exercises 3

- Write RE for
  - All strings of 0’s and 1’s
    - (0|1)*
  - All strings of 0’s and 1’s with at least 2 consecutive 0’s
    - (0|1)*00(0|1)*
  - All strings of 0’s and 1’s beginning with 1 and not having two consecutive 0’s
    - (1+10)*

More Exercises

- All strings of 0’s and 1’s ending in 011
  (0|1)*011

- any number of 0’s followed by any number of 1’s followed by any number of 2’s
  0*1*2*

- strings of 0,1,2 with at least one of each symbol
  00*11*22*

More Exercise

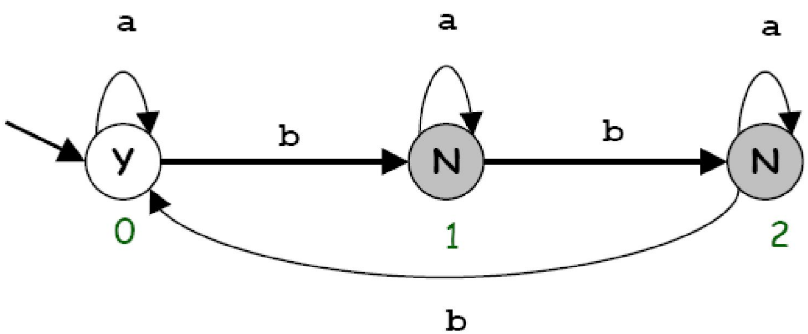
- The set of all strings whose number of 0’s is divisible by 3
  - (1+01*01*0)*

Theory of DFAs and REs

- RE. Concise way to describe a set of strings.
- DFA. Machine to recognize whether a given string is in a given set.
- **Duality**: for any DFA, there exists a regular expression to describe the same set of strings; for any regular expression, there exists a DFA that recognizes the same set.

Duality Example

- DFA for multiple of 3 b's:

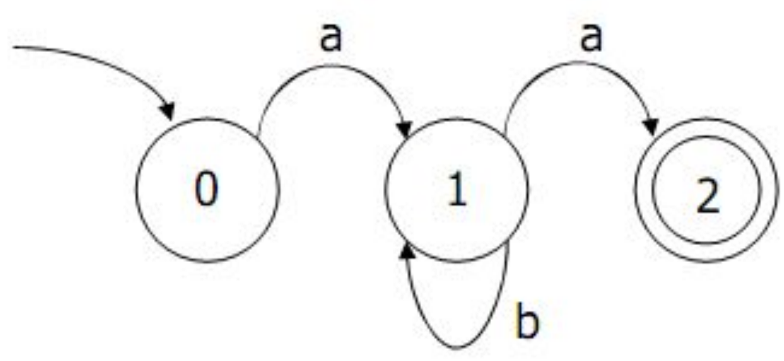


- RE for multiple of 3 b's:

(a*ba*ba*ba*)* a*

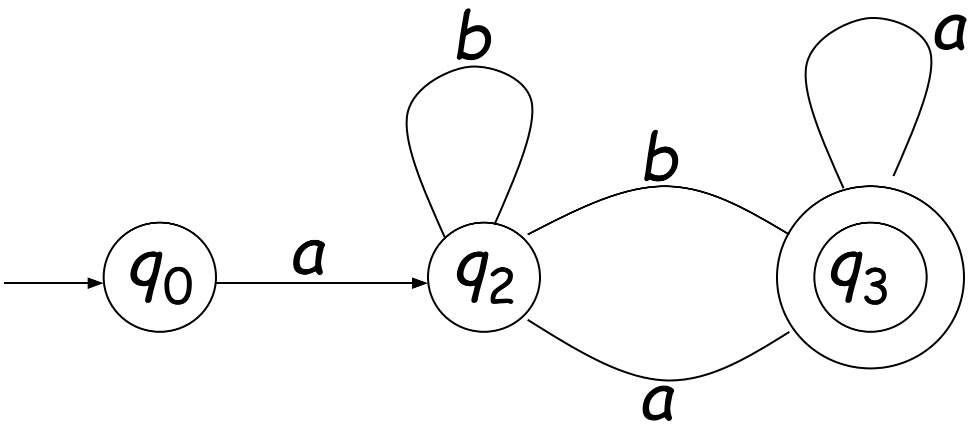
Problem 1

- Make a DFA that accepts the strings in the language denoted by regular expression ab*a



Problem 2

- Write the RE for the following automata:



- a(a|b)*a

DFA to RE: State Elimination

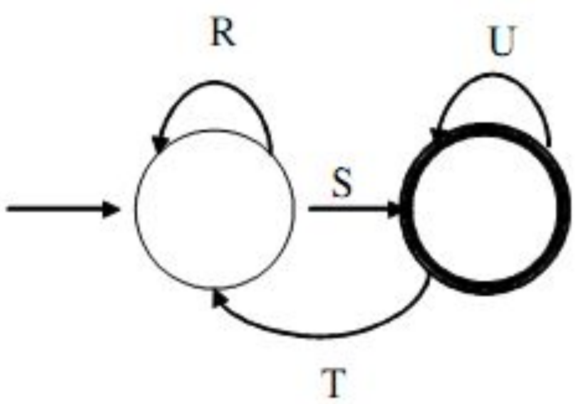
- Eliminates states of the automaton and replaces the edges with regular expressions that includes the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE

DFA to RE via State Elimination (1)

- Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.
- The result will be one or two state automaton with a start state and accepting state.

DFA to RE State Elimination (2)

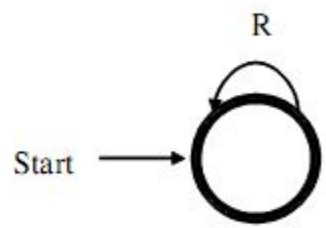
- If the two states are different, we will have an automaton that looks like the following:



- We can describe this automaton as: (R | SU*T)*SU*

DFA to RE State Elimination (3)

- If the start state is also an accepting state, then we must also perform a state elimination from the original automaton that gets rid of every state but the start state. This leaves the following:



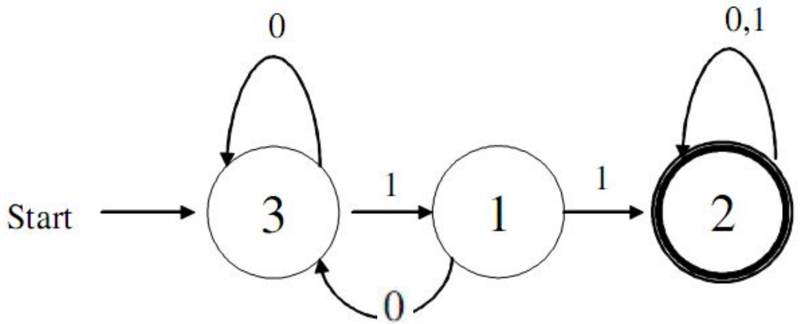
- We can describe this automaton as simply R*

DFA to RE State Elimination (4)

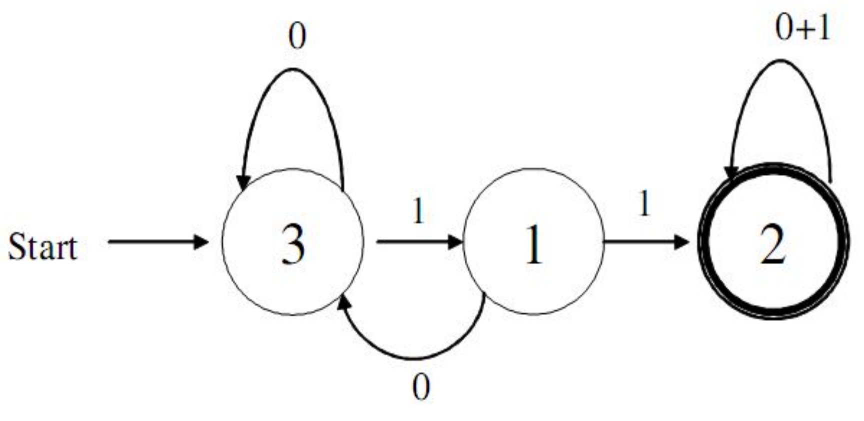
- If there are n accepting states, we must repeat the above steps for each accepting states to get n different regular expressions, R1, R2, … Rn.
- For each repeat we turn any other accepting state to non-accepting.
- The desired regular expression for the automaton is then the union of each of the n regular expressions: R1 U R2… U RN

DFA->RE Example

- Convert the following to a RE:

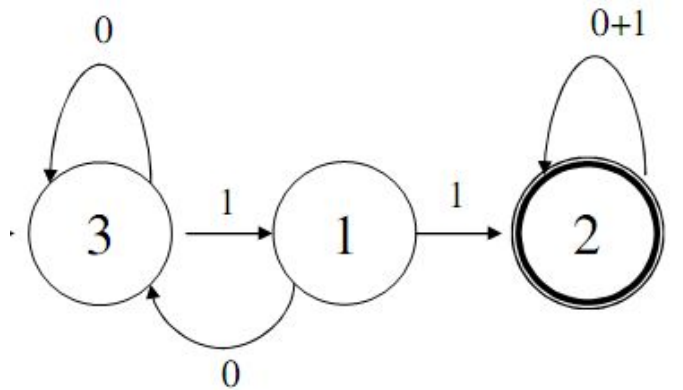


- First convert the edges to RE's:

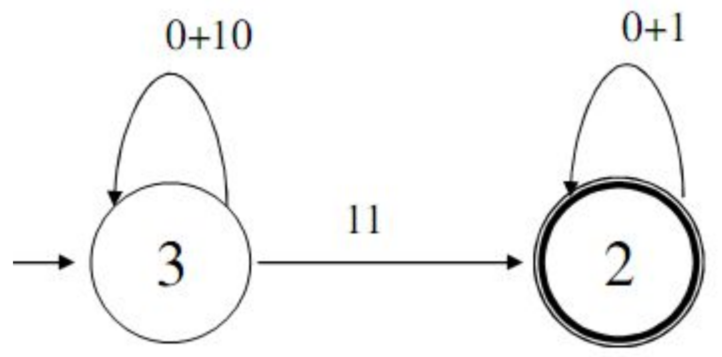


DFA -> RE Example (2)

- Eliminate State 1:



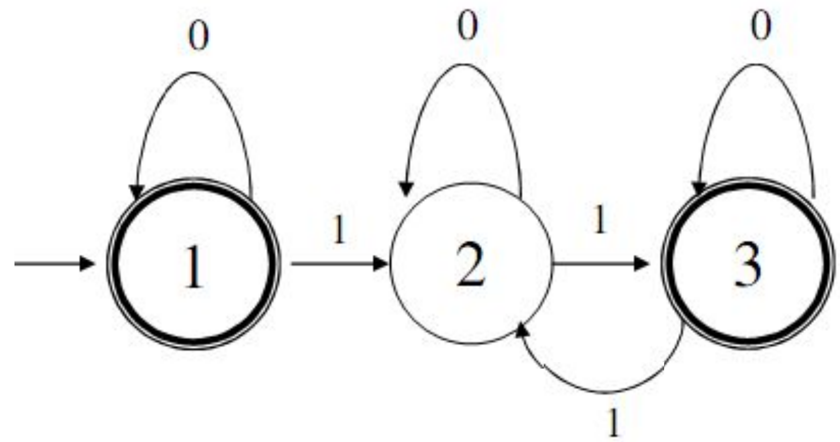
- Note edge from 3->3



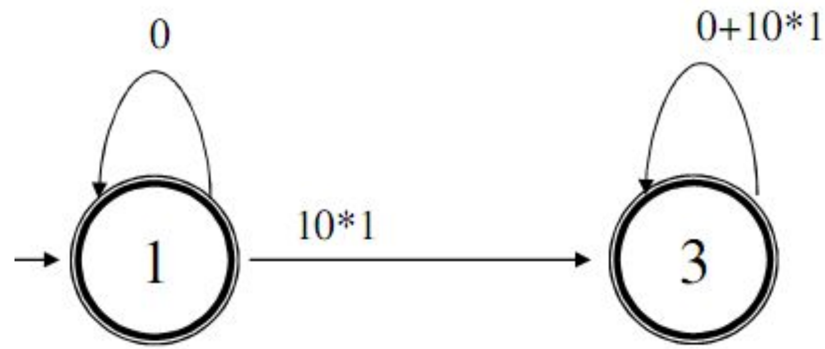
- Answer: (0+10)*11(0+1)*

Second Example

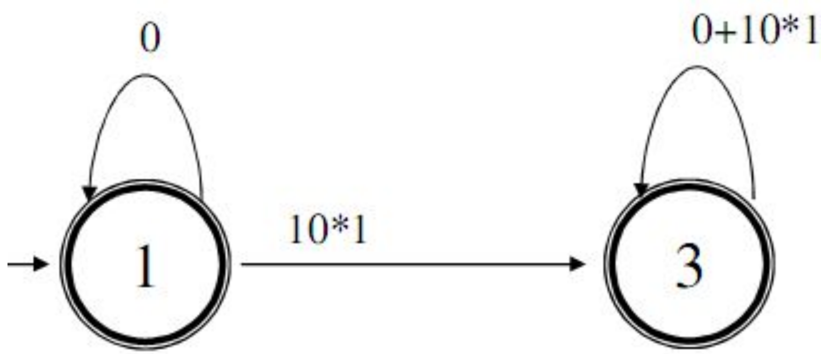
- Automata that accepts even number of 1's



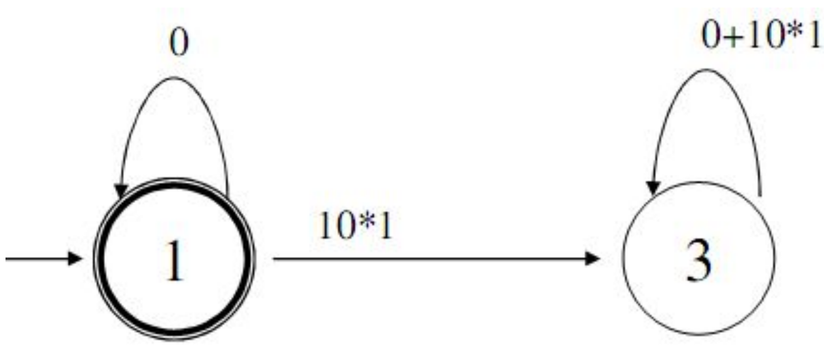
- Eliminate state 2:



Second Example (2)

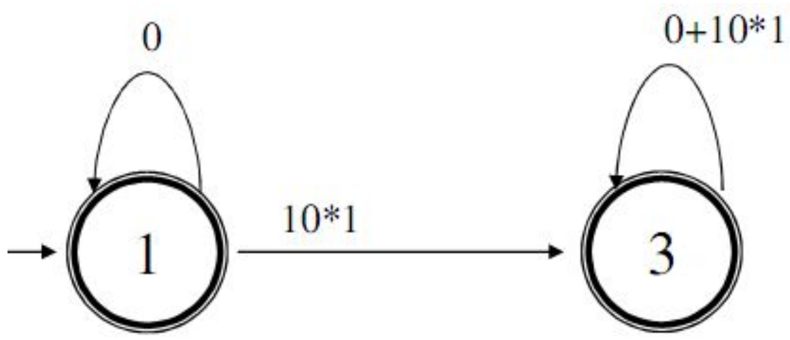


- Two accepting states, turn off state 3 first

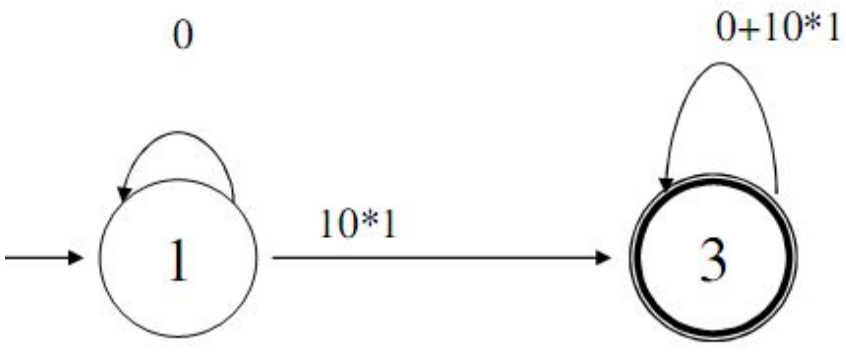


- This is just 0*; can ignore going to state 3 since we would "die"

Second Example (3)



- Turn off state 1 second:



- This is just 0*10*1(0|10*1)*
- Combine from previous slide to get 0* | 0*10*1(0|10*1)*

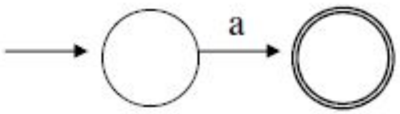
RE ->Automata

- We have shown we can convert an automata to a RE. To show equivalence we must also go the other direction, convert a RE to an automaton.
- We can do this easiest by converting a RE to an ε-NFA
  - Inductive construction
  - Start with a simple basis, use that to build more complex parts of the NFA

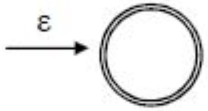
RE ->Automata

- Basis:

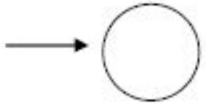
R=a



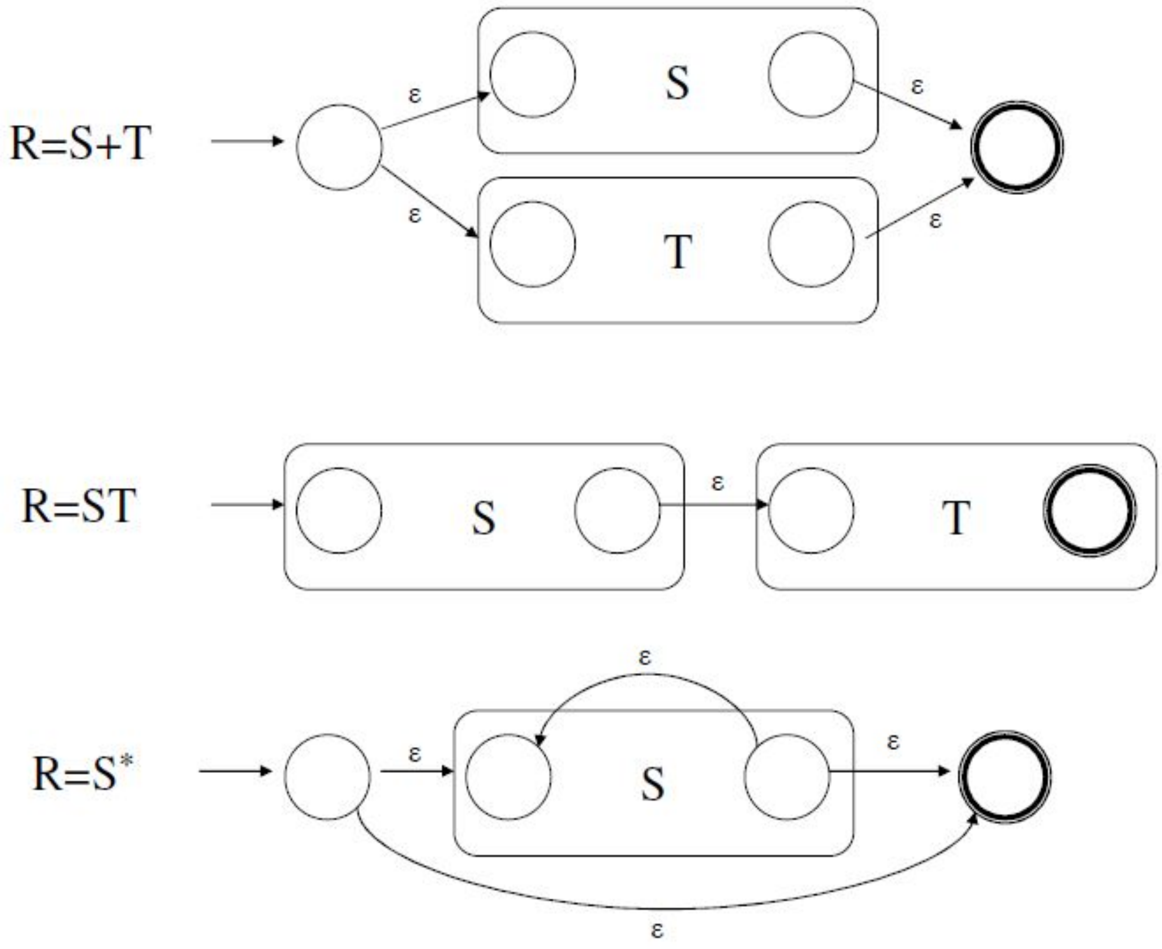
R=ε



R=Ø



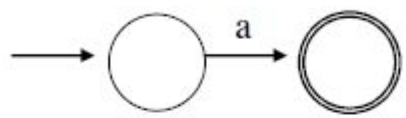
RE ->Automata



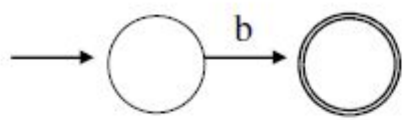
RE ->Automata

- Convert R= (ab+a)* to an NFA
  - We proceed in stages, starting from simple elements and working our way up

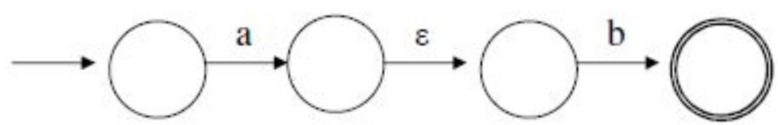
a



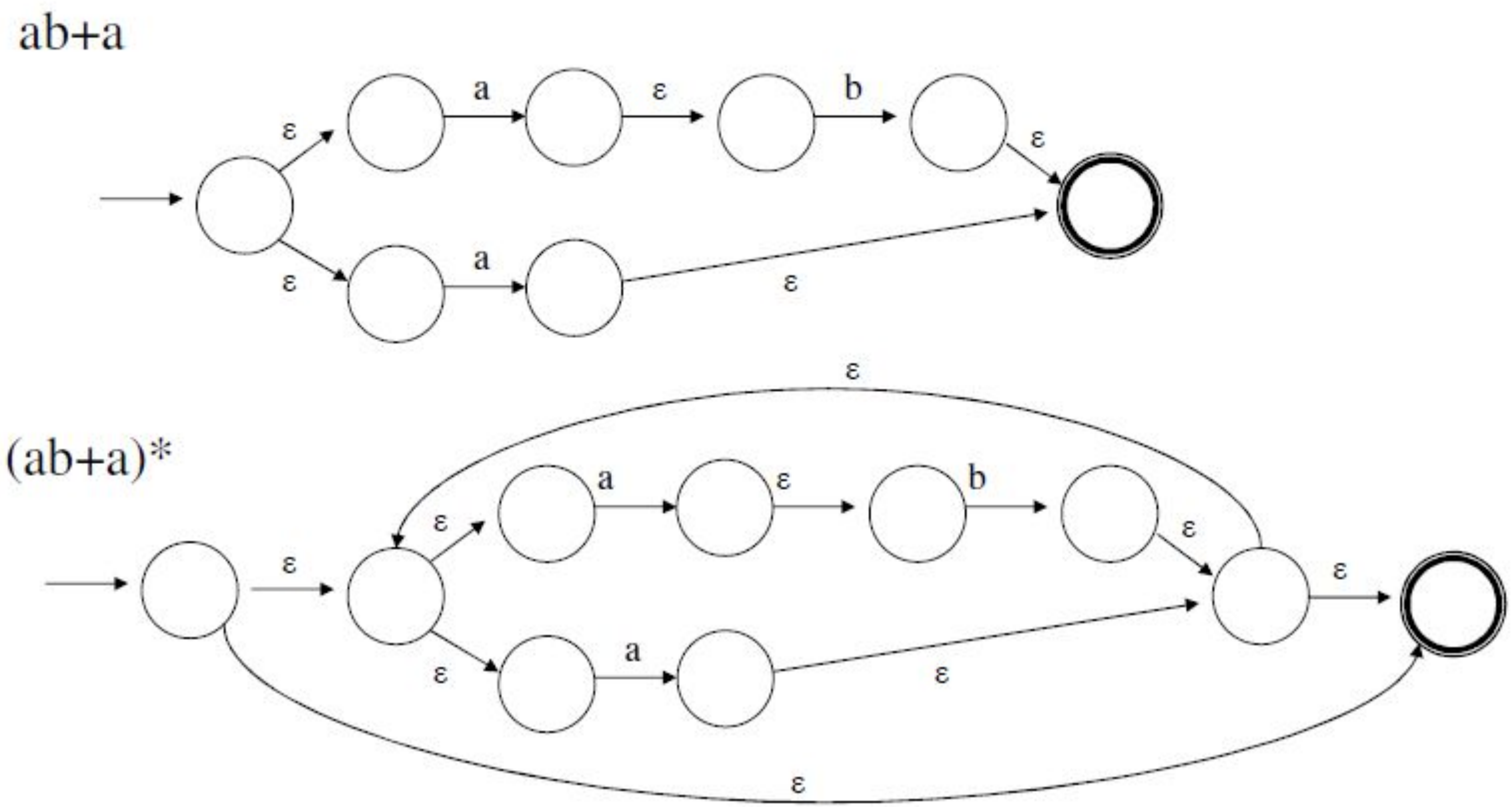
b



ab



RE ->Automata



RE ->Automata

- Another approach
  - Mishra