Lecture 4

Jan 22, 2019

Regular Expression:- over Σ are the smallest set of expressions including

$\in$	
$'c'$	where $'c \in \Sigma$
$A + B$	where A, B are rexp over Σ
AB	" A, B are ~~reg~~ rexp over Σ
A^*	where A is rexp over Σ

$L(\in) = \{" "\}$

$L('c') = \{"c"\}$

$L(A+B) = L(A) \cup L(B)$

$L(AB) = \{ab \mid a \in L(A) \text{ and } b \in L(B)\}$

$L(A^*) = \bigcup_{i \geq 0} L(A^i)$

- Keywords :- 'if', 'else', 'begin'

Note 'else' abbreviates 'e' 'l' 's' 'e'

- Integers :- non-empty set of digits

digits = '0' + '1' - - - - + '9'

integer = digit digit* = digit+

Abbreviation $\boxed{A^+ = AA^*}$

* Identifier :- string of letters or digits starting with a letter

letter = 'A' + - - - - 'Z' + 'a' + - - - - + 'z'

identifier = letter (letter + digit)*

Whitespace :-

(' ' + '\n' + '\t')+

Example :-

jakaur @ ucalgary.ca

Σ = letters $\cup$ {., @}

name = letter$^+$

address = name '@' 'name'. 'name'

- Regular Expression describe many useful languages.
- Reg. ~~expression~~ languages are language specification
  - need an implementation

A | B $\longrightarrow$ A + B

A + ϵ

Range:- 'a' + 'b' + --- 'z' = [a-z]

complement of [a-z] : [^ a-z]

Lexical Specification

(4)

$$s \in L(R)$$

rexp for lexemes of each token

Number = digit$^+$

keywords = 'if' + 'else'

identifier = letter (letter + digit)*

openPar = '('

- - - -

Finite Automata :-

rexp → specification

finite automata → implementation

- A finite automaton consists of
  - input alphabet Σ
  - a set of states s
  - a start state n
  - a set of accepting stat $F \subseteq S$
  - a set of transition state → sta

Finite Automata ....

Transition

$$S_1 \xrightarrow{a} S_2$$

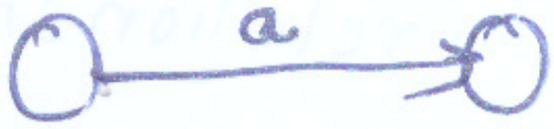
in state S_1 an input "a" go to state S_2

State Graph:

A state 

The start state 

An accepting state 

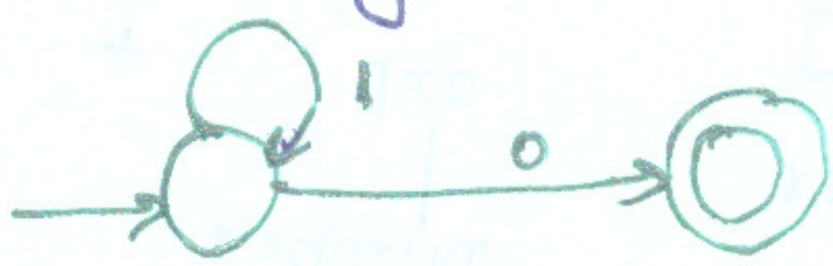
transition 

if end of input and in accepting state $\Rightarrow$ accept

otherwise $\Rightarrow$ reject

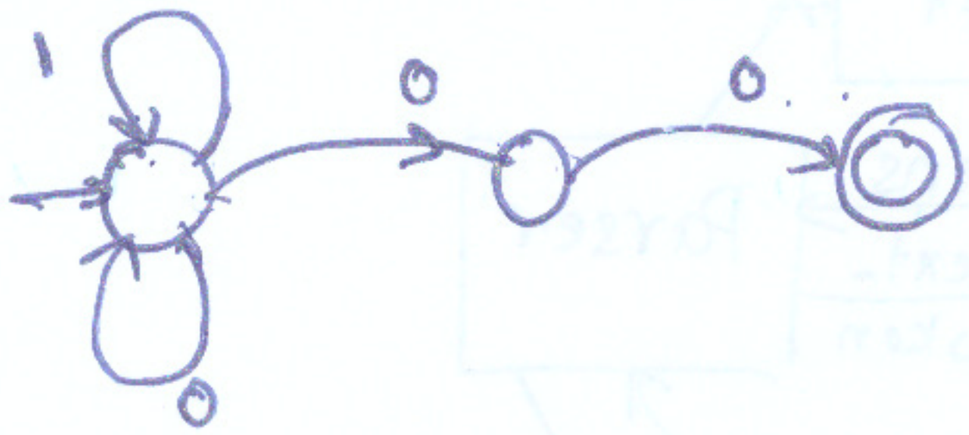
A finite automaton that accepts only "1"

A finite automaton accepting any number of 1's followed by a single 0.



NFA and DFA language.	DFA	NFA
	• Every input string leads to the unique state of finite automata.	• for the same input there can be more than one state
	• conversion of regular expression to DFA is complex	• Regular expression can be easily converted to NFA using Thompson's construction.
	• DFA requires more memory for storing the state information	• NFA require more computati to match regular express with input
	• DFA is not possible to move next state without reading any symbol	• In NFA we can move to next state without readi any symbol
	number of states in DFA ≥	number of stat in NFA

NFA Acceptance



input :: 1 0 0

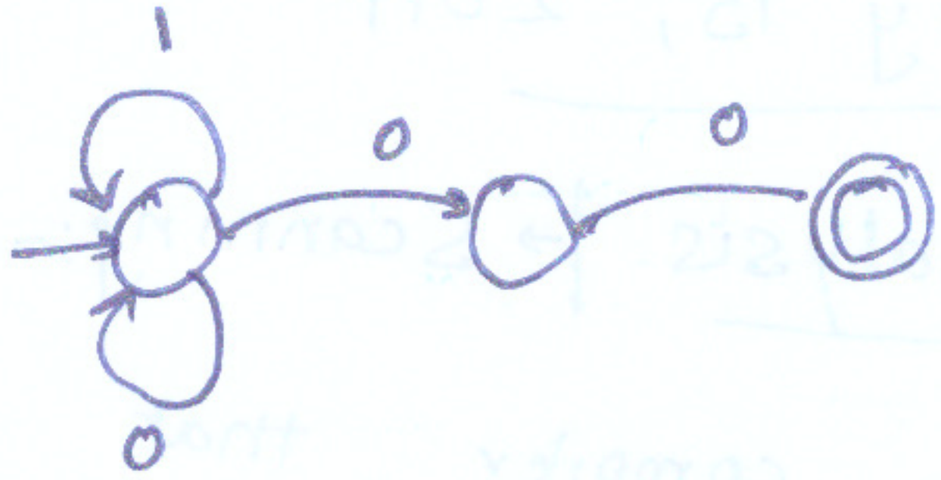
Rule : NFA accepts if it can get to final state

Execution of finite Automata

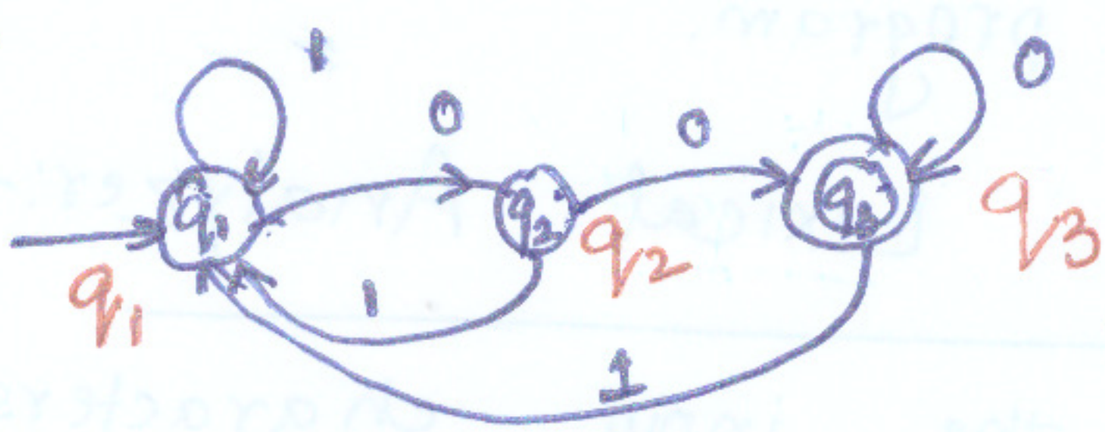
- A DFA can take only one path through the state graph
  - completely determined by input
- NFA can choose:
  - whether to make ∈ mov
  - which of multiple transitions for a single input to take.

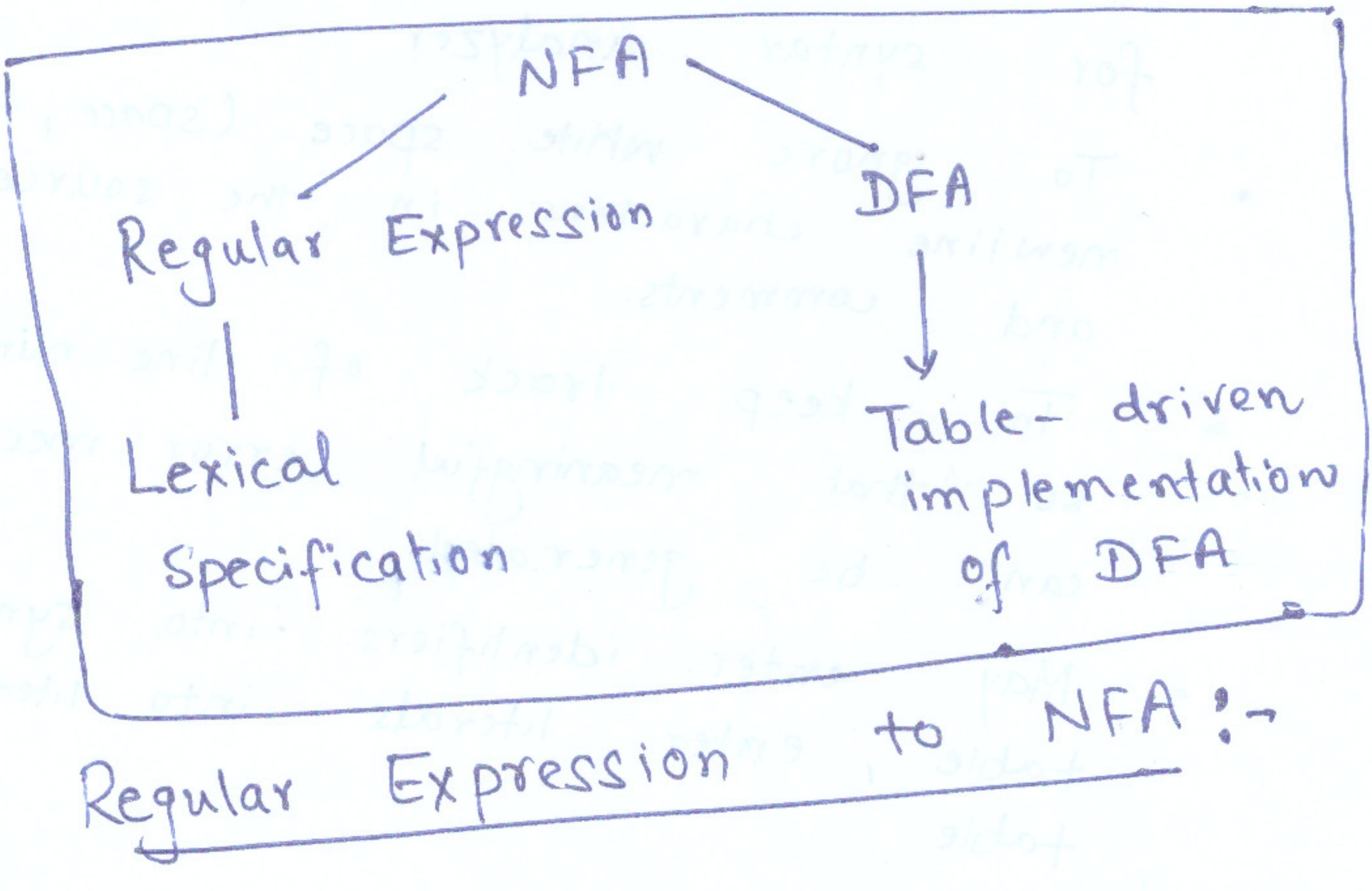
NFA vs DFA :→

NFA



DFA :-

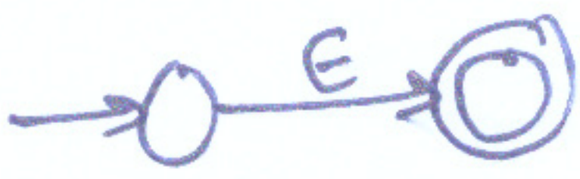




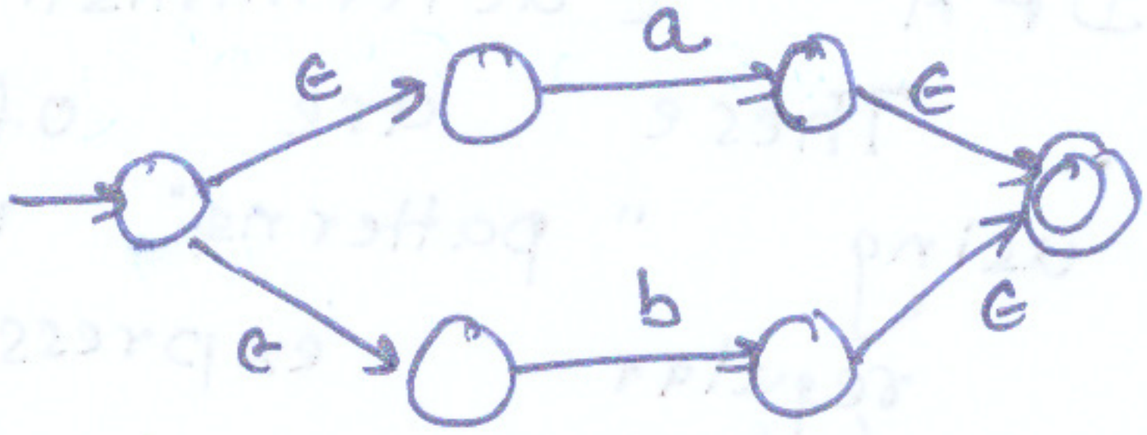
Regular Expression to NFA :-

For input a

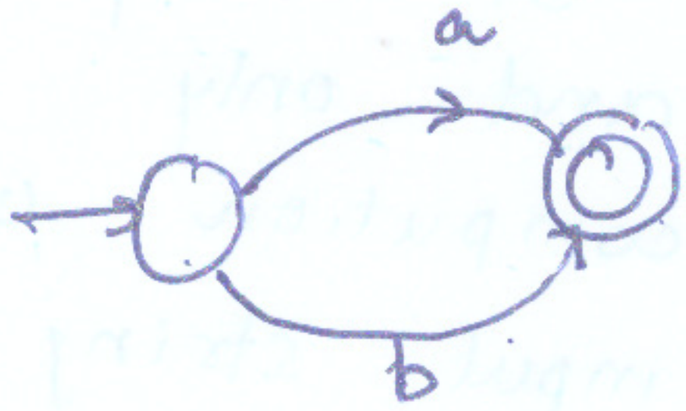
For ϵ



A + B With ∈ move

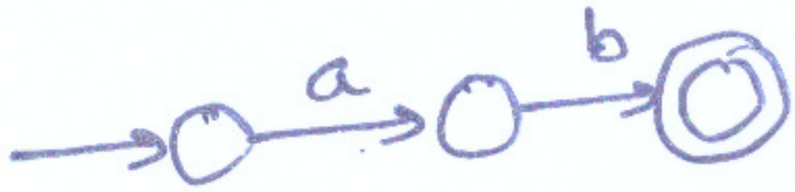


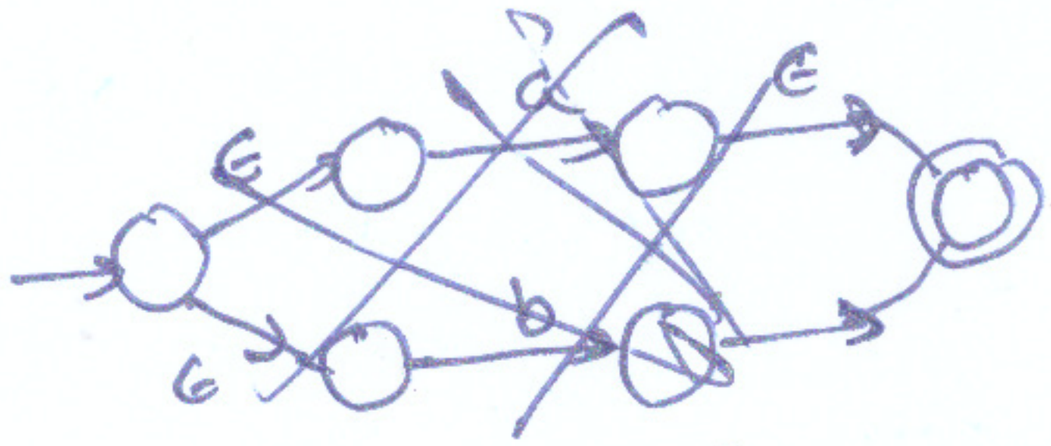
without ∈ move

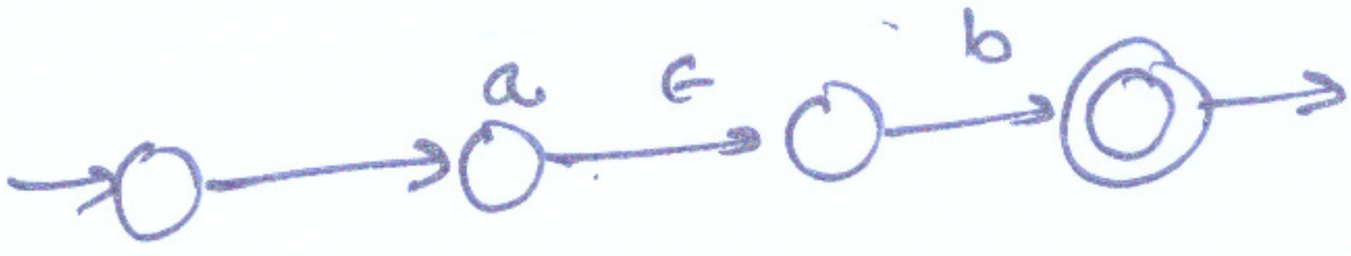


without ∈ more

AB







A*

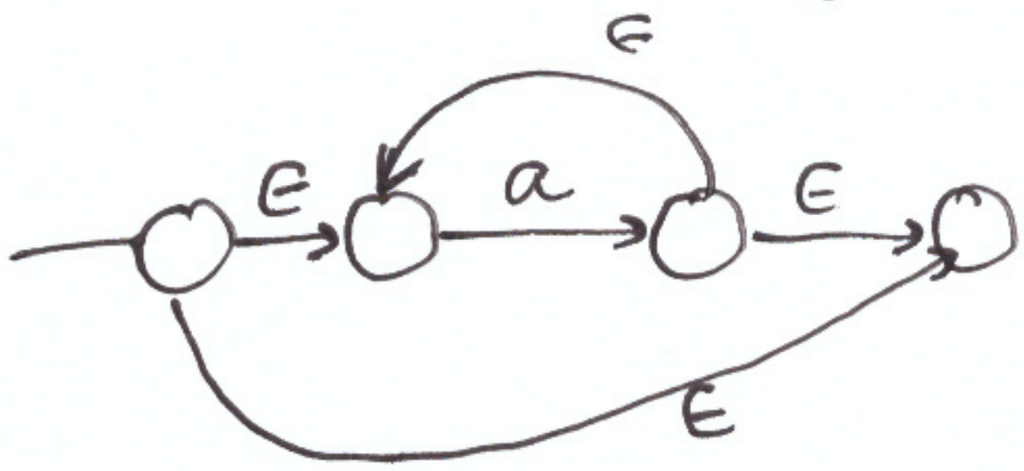
start & final is same

Lecture 5

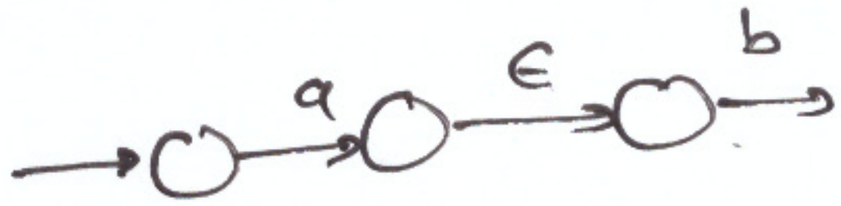
January 24, 201

a^*

Kleene Closure



$a^* \rightarrow \{\epsilon, a, aa \ldots\}$



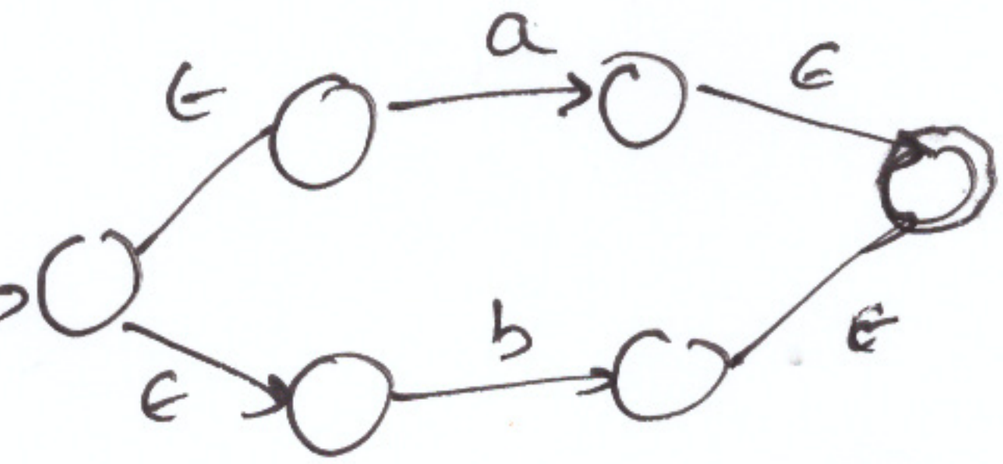
ab

concatenation

$a+b \rightarrow a \cup b$

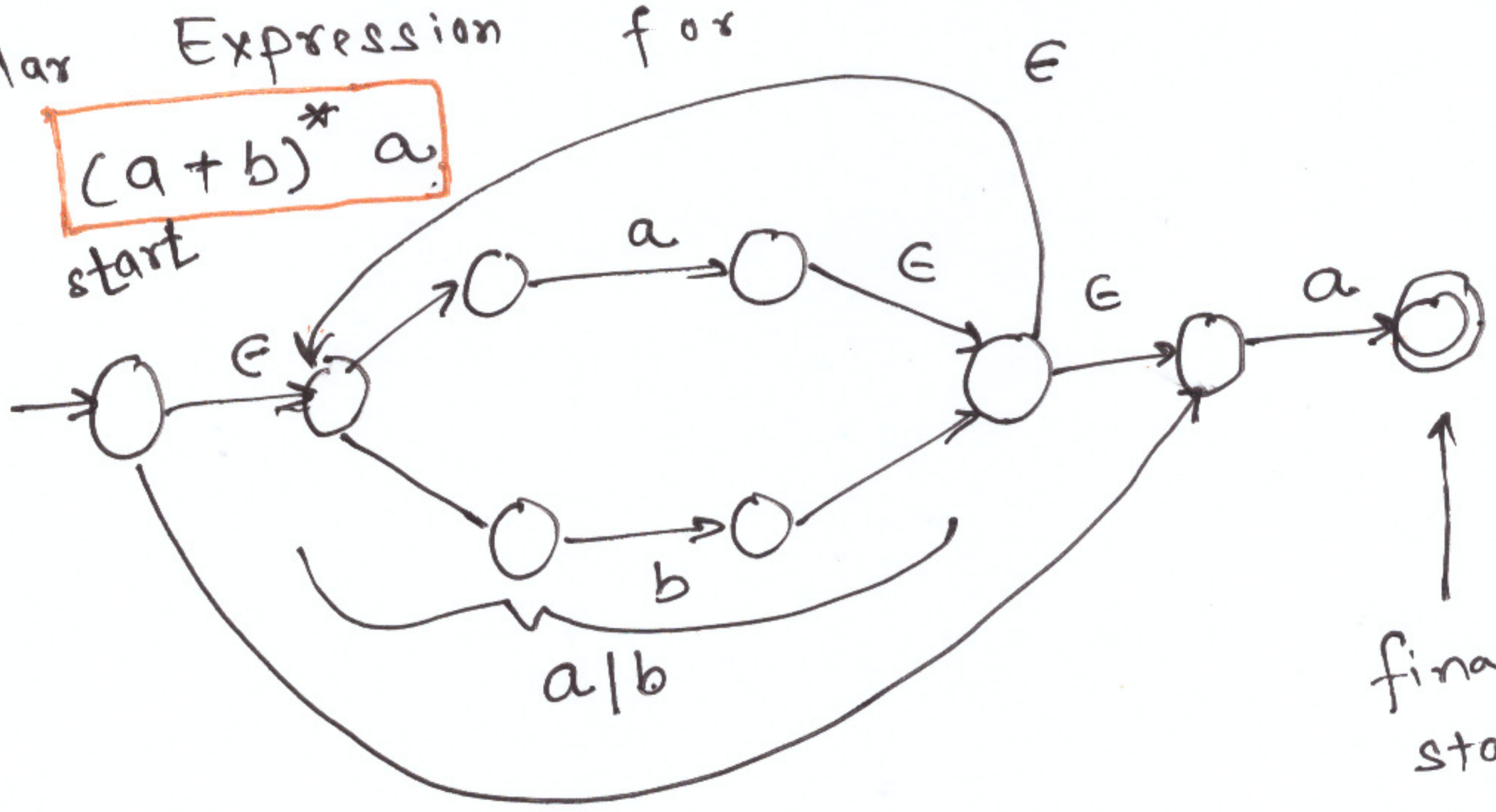
$a|b$

union



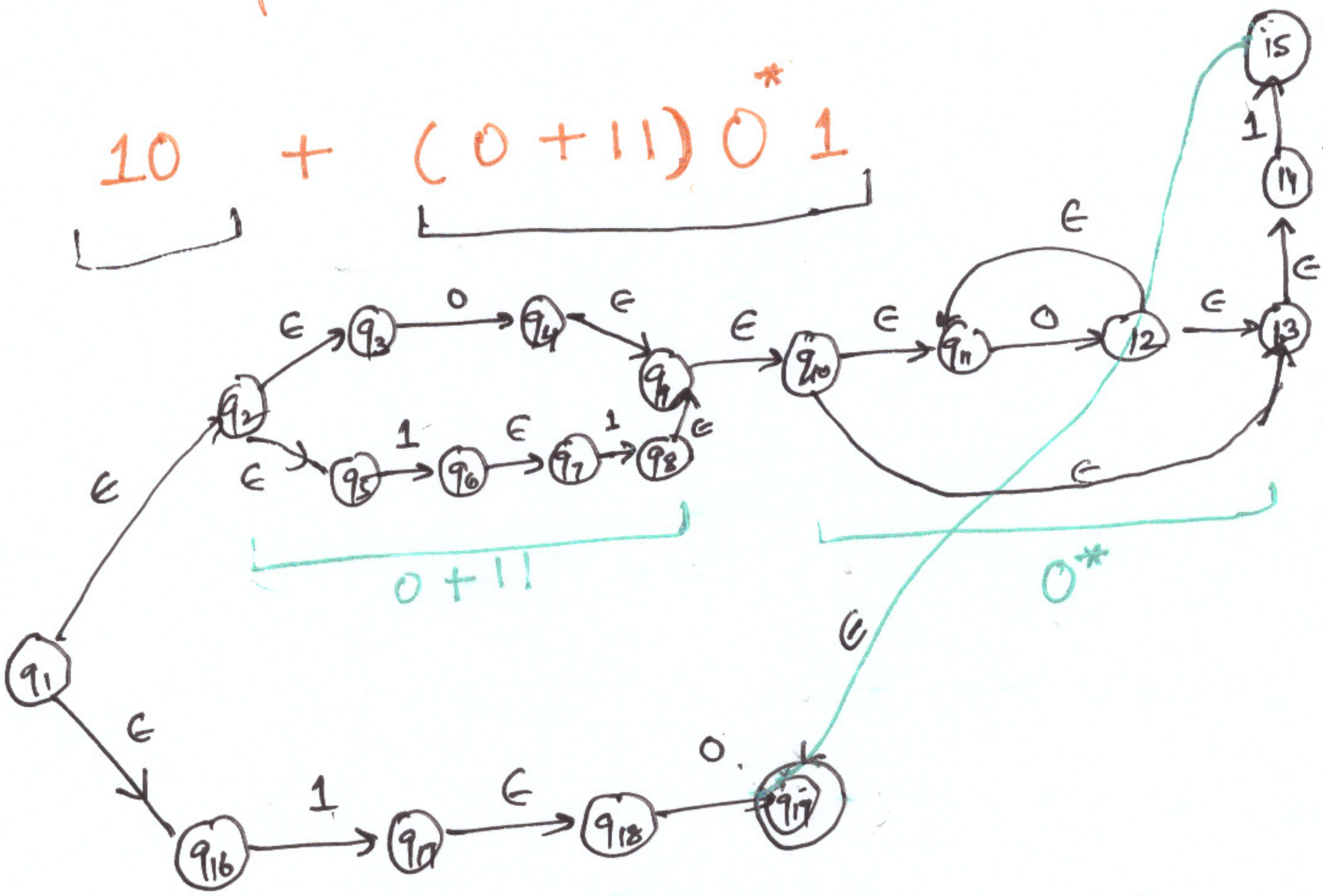
Regular Expression for

$(a+b)^* a$



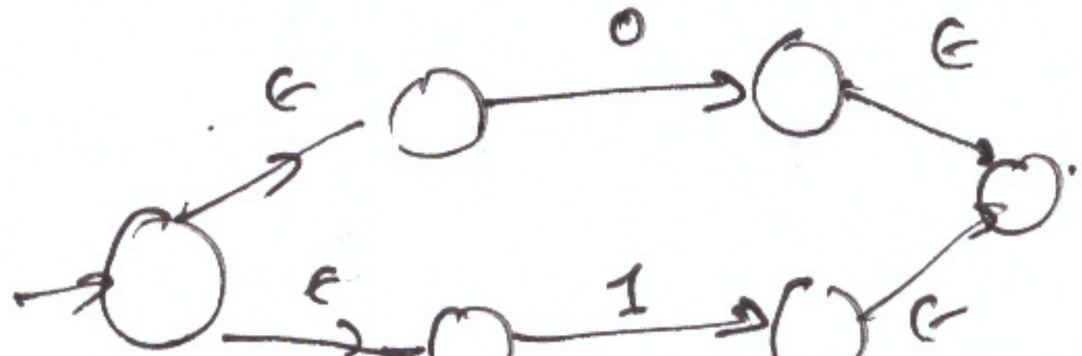
Regular Expression to NFA with epsilon move using Thompson's Construction

$$10 + (0+11)^* 01$$



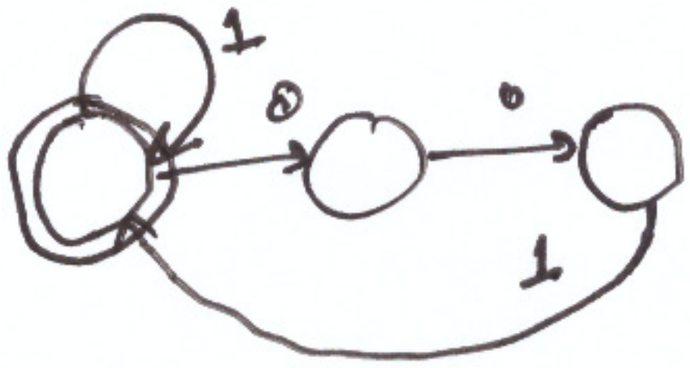
No. of states = 19

ε moves = 15



Regular Expression to NFA

$1^*(001^+)^*$



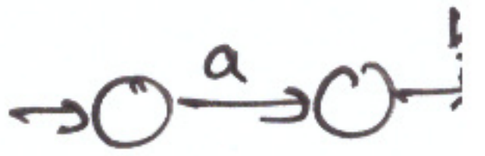
0 0 1 0 0 1 Y

1 1 1 1 Y

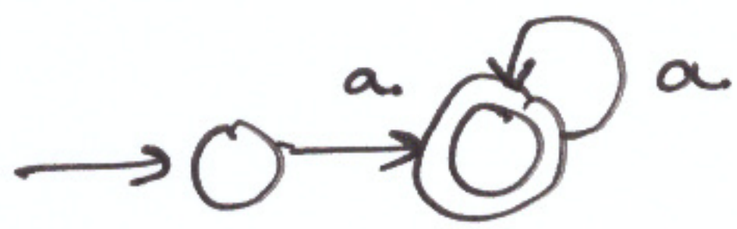
1 0 0 1

a^*

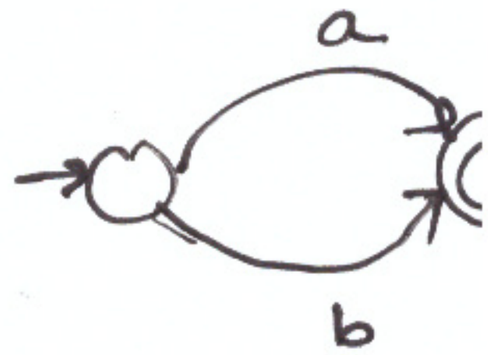
ab



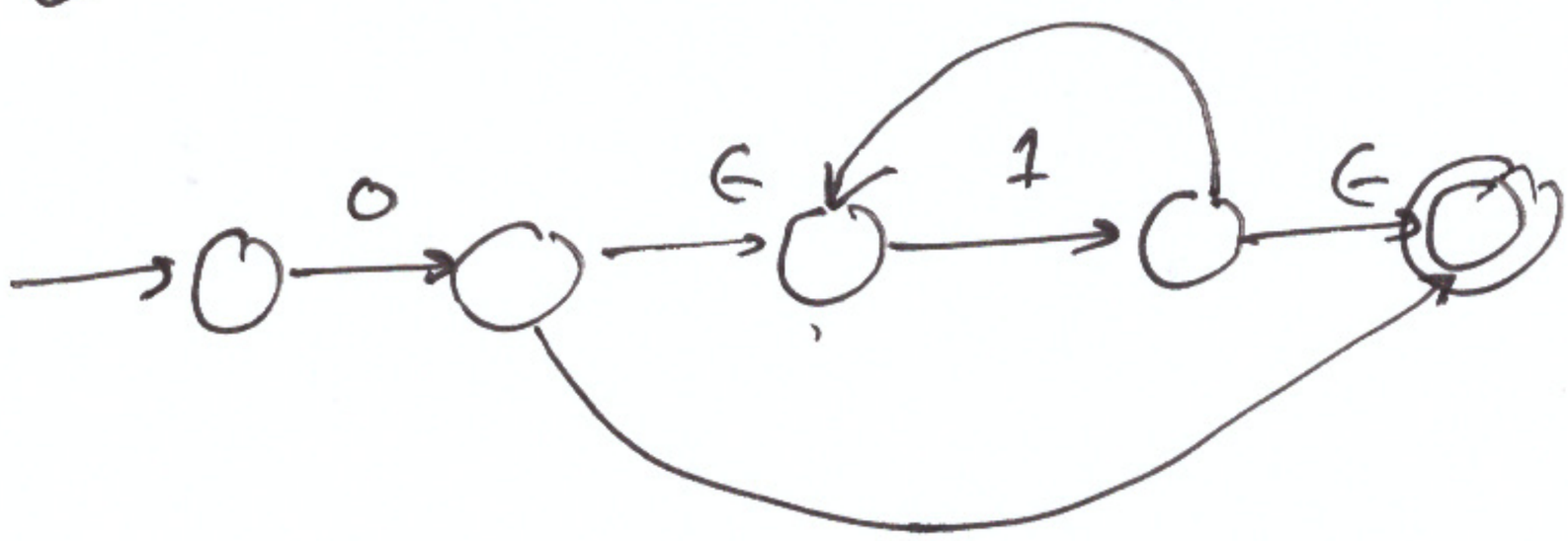
a^+ aa^*



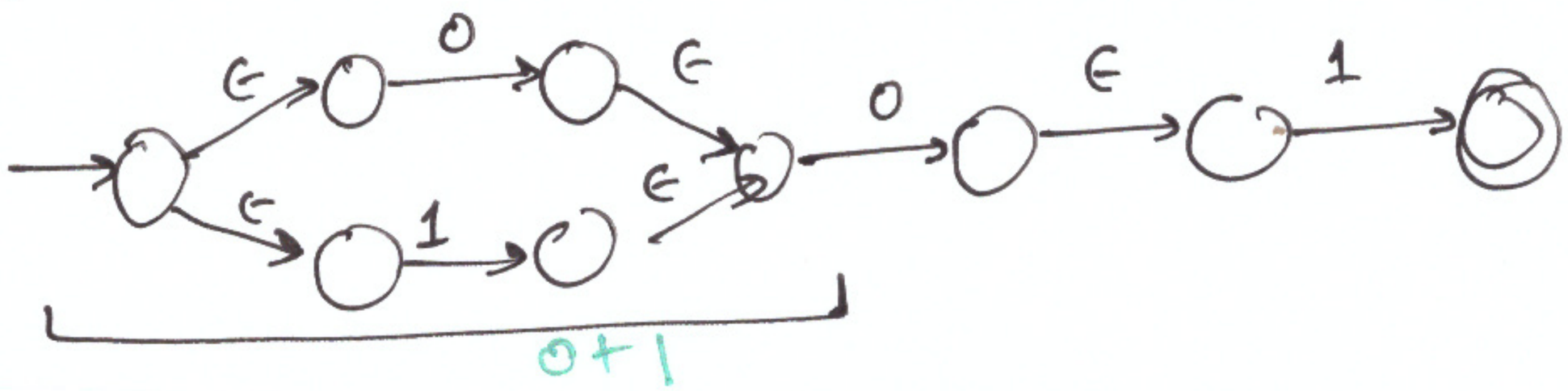
$a+b$



01^* to ϵ-NFA



$(0+1)01$



$a^* b c$

$(1^* 0 0)^*$

$\in$ moves