Lexical Analysis

Files & media	Lexical1.pdf Lexical2.pdf Lexical3.pdf
Name	Week 10-11
Review	☐

Introduction to lexical analysis

Input and output: source program to token

group character into meaningful words

lexical analysis from input to output

Regular Expression: specify tokens

token

- syntactic category
- classify program substring according to its syntactic role
- output: to parser(syntax analysis)

Regular expression

regular expression(pattern)

to Lex(software tool): recognize token

to Finite Automaton: recognize token

Regular Expression

- alphabet string, language

- regular expression
- regular set(regular language)

pattern

- lexical analyzer - identify lexeme

e.g. rules specify token pattern are regular expression

Alphabet & string

alphabet: **Σ**

string **s: drawn from Σ**

length: $|s|$

Kleene Closure

Kleene closure of Σ = **Σ***

Σ* = { Σ0, Σ1, Σ2, ...}

ε : null expression

Union: L U M

Concatenation: L · M

Kleene closure:

Union of *L* and *M*	*L* U *M L* | *M L* + *M*	$s \mid s \in L$ or $s \in M$
Concatenation	*LM L* · *M* (Multiplication)	st $s \in L$ and $t \in M$
Kleene closure	*L**	L* = ⋃L*(i)*, where i = 0,1, ..., ∞
Positive closure	*L*+	L+ = ⋃L*(i)*, where i = 1, 2, ..., ∞

Kleene closure > concatenation > union

Example:

Given L = {a, b} and M = {a, bb}
$L \cup M$ = {a, b, bb}
LM = {aa, abb, ba, bbb} = multiplication
L* = L0 ∪ L1 ∪ L2 ∪ L3 ∪ … = {ε} ∪ {a, b} ∪ {aa, ab, ba, bb} ∪
{aaa, aba, baa, bba, aab, abb, bab, bbb} ∪ …
L+ = L1 ∪ L2 ∪ L3 ∪ … = {a, b} ∪ {aa, ab, ba, bb} ∪
{aaa, aba, baa, bba, aab, abb, bab, bbb} ∪ …

Regular expression:

a | b = {a, b}

(a | b) (a | b) = {aa, ab, ba, bb} OR aa | ab | ba | bb

a* = {ε, a, aa, aaa, ...}

(a | b)* = {ε, a, b, aa, bb, aaa, bbb, ...}

a | (a* b) = {a, b, ab, aab, aaab, aaaab, ...}

string: letter & digit starting with letter: Letter (letter | digit)*

Notation for remember:

one or more instance: r+ OR rr*

zero or one instance: r? OR r | ε

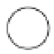
character classes: xx | yy OR [x-y]

Finite Automata

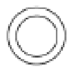
NFA(Nondeterministic Finite Automata)

1. input alphabet
2. states(s)

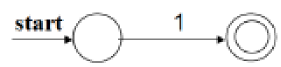
3. start state(s0)
4. accepting states
5. move(transition function)

A state 

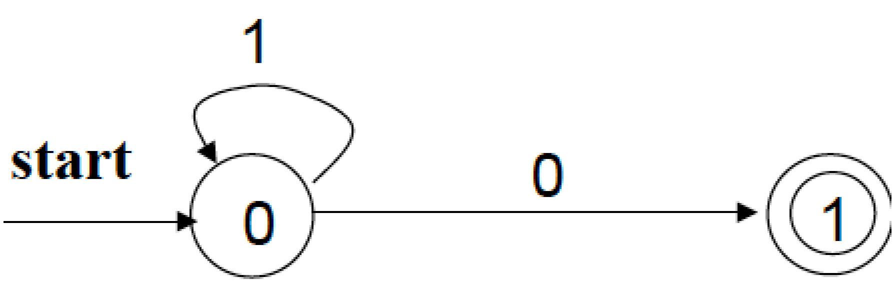
The start state 

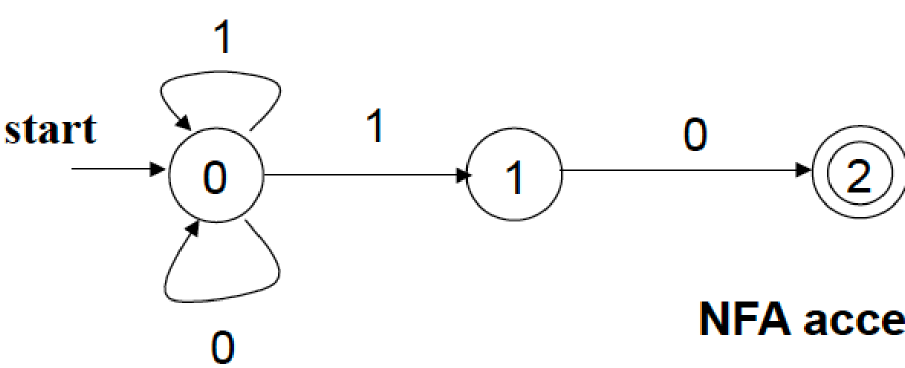
An accepting state 

A transition 

A simple example: a finite automaton that accepts only “1” 

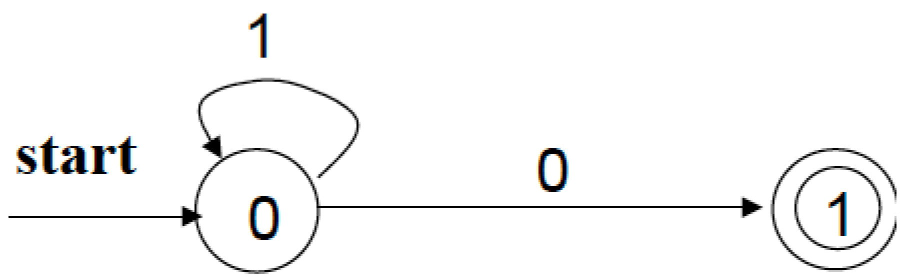
NFA accepting 1*0

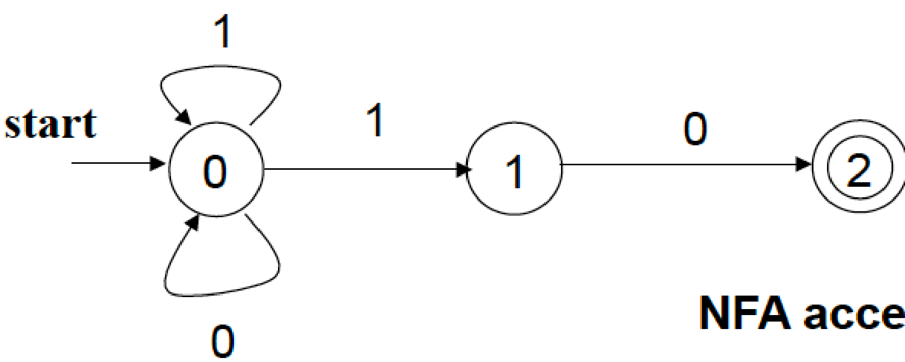




NFA accepting (1|0)*10

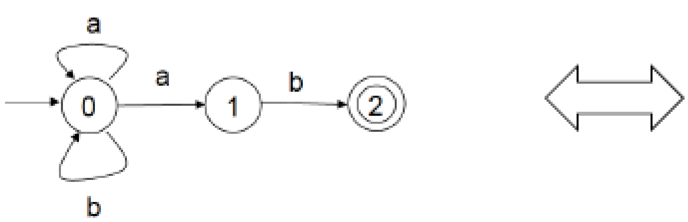
NFA accepting 1*0





NFA accepting (1|0)*10

State Transition Table



STATE	Input Symbols	
	a	b
0	{0, 1}	{0}
1		{2}

ε-Transition - no reading any input

Hard to Implement

multiple transition from 1 input in given state can have ε-Transition

Easy to form from regular expressions

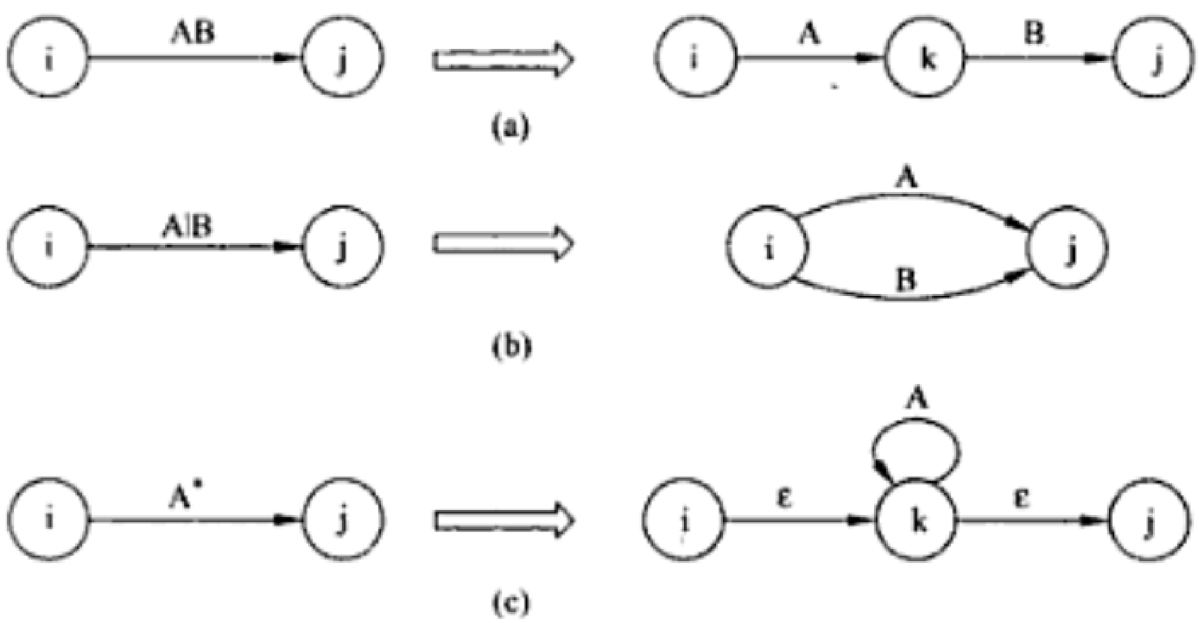
Hard to implement the recognition (decision) algorithm

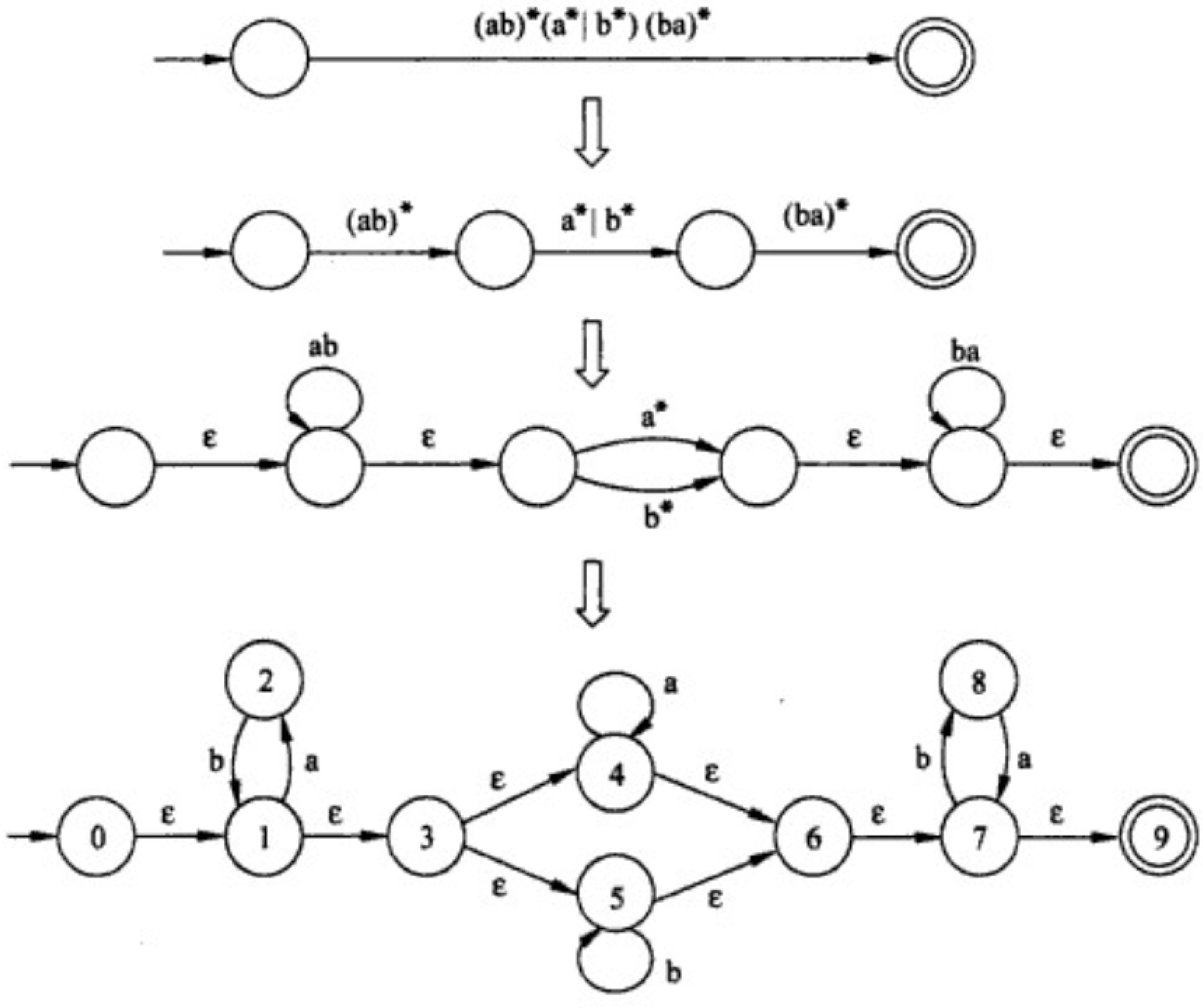
DFA(deterministic Finite Automata)

1 transition per input

no ε-transition

Conversion

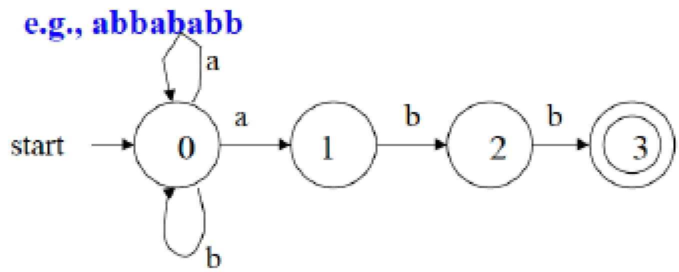




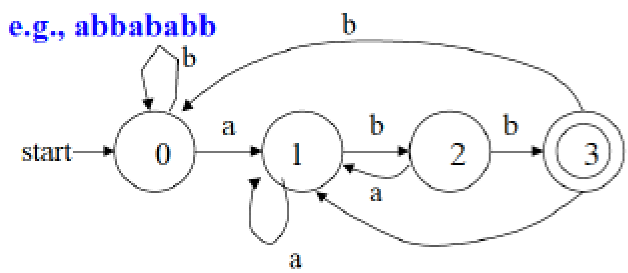
Compare b/t DFA & NFA

NFA	DFA
easy to generate string	easy to generate & recognize string
may go many states with 1 input	go only 1 deterministic state with 1 input
may go another state when there's no input(ε-transition)	don't go anywhere when no input

Given alphabet $\Sigma = \{a, b\}$,
an NFA recognizing the language $(a|b)^*abb$



A DFA recognizing the language $(a|b)^*abb$



NFA to DFA

DFA is special case of NFA

no ε-transition

ε-closure: reachable state

ε - NFA to NFA

Table-driven Implementation of DFA