UCS1602:
COMPILER DESIGN

Finite Automata

Session Objectives

- To learn concepts finite automata
- To study about Non deterministic finite automata, Deterministic finite automata
- To study about conversion of RE to NFA
- To study about conversion of NFA to DFA
- To study about conversion of RE to DFA

Session Outcomes

- At the end of this session, participants will be able to
  - Understand the concepts of NFA, DFA
  - Understand the conversion of RE to NFA, NFA to DFA and DFA minimization

Outline

- Introduction about NFA and DFA
- Conversion of RE to NFA
- Conversion of NFA to DFA
- Conversion of RE to DFA

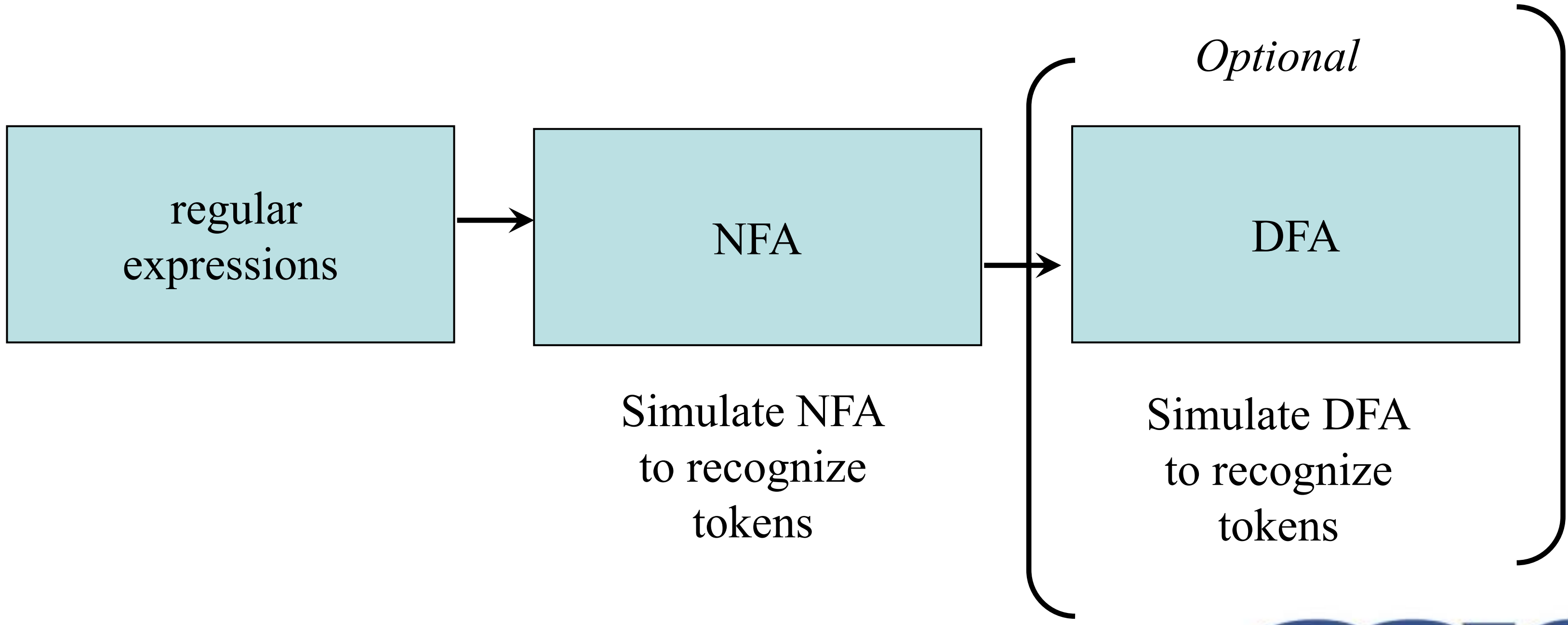
Finite Automata

Outline

- Regular expressions = specification
- Finite automata = implementation
- 2 Types DFA, NFA
- Deterministic Finite Automata (DFA)
  - One transition per input per state
  - No ε-moves
- Nondeterministic Finite Automata (NFA)
  - Can have multiple transitions for one input in a given state
  - Can have ε-moves
- *Finite* automata have *finite* memory
  - Need only to encode the current state

Design of a Lexical Analyzer Generator

- Translate regular expressions to NFA
- Translate NFA to an efficient DFA



Deterministic Finite Automata

- A DFA is a 5-tuple (S, Σ, δ, s_0, F) where

  S is a finite set of *states*
  Σ is a finite set of symbols, the *alphabet*
  δ is a *mapping* from $S \times \Sigma$ to a state
  $s_0 \in S$ is the *start state*
  $F \subseteq S$ is the set of *accepting* (or *final*) *states*

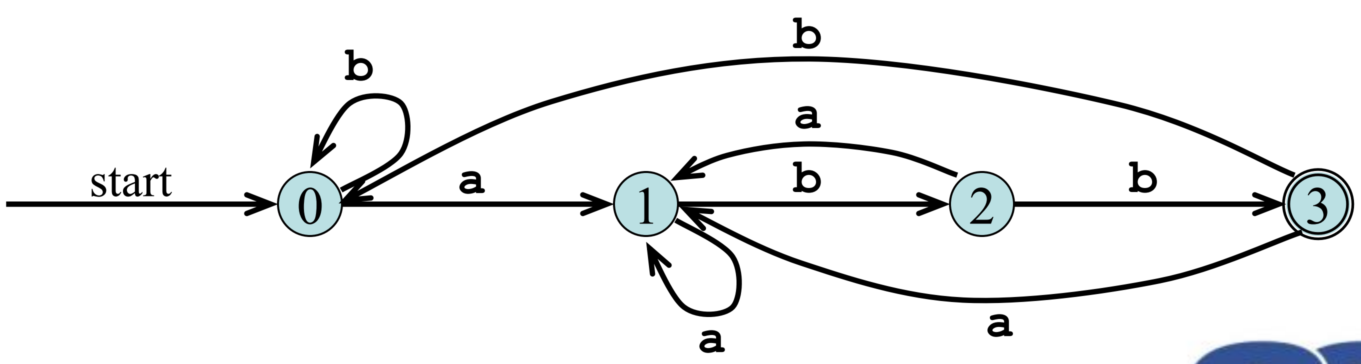
Simulating a DFA

- Input string x terminated by an eof. A DFA D
- Output yes if accepts else no

```
S := s0
a := nextchar()
while a ≠ eof do
      S := move(S,a)
      a := nextchar()
end do
if S is in F then
      return "yes"
else        return "no"
```

Example DFA

A DFA that accepts (**a** | **b**)***abb**



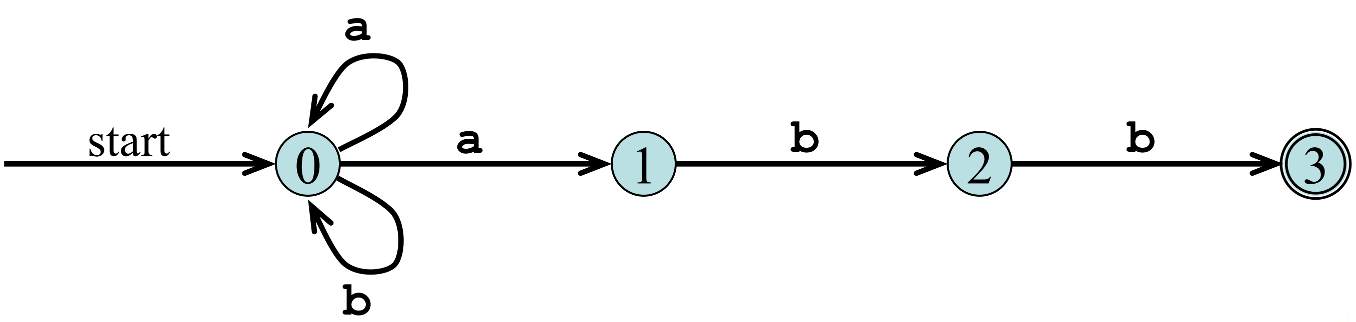
Nondeterministic Finite Automata

- An NFA is a 5-tuple (S, Σ, δ, s_0, F) where

  S is a finite set of *states*
  Σ is a finite set of symbols, the *alphabet*
  δ is a *mapping* from $S \times \Sigma$ to a set of states
  $s_0 \in S$ is the *start state*
  $F \subseteq S$ is the set of *accepting (or final) states*

Transition Graph

- An NFA can be diagrammatically represented by a labeled directed graph called a *transition graph*



$S = \{0,1,2,3\}$
$\Sigma = \{\mathbf{a},\mathbf{b}\}$
$s_0 = 0$
$F = \{3\}$

Transition Table

- The mapping δ of an NFA can be represented in a *transition table*

$$\delta(0,\mathbf{a}) = \{0,1\}$$
$$\delta(0,\mathbf{b}) = \{0\}$$
$$\delta(1,\mathbf{b}) = \{2\}$$
$$\delta(2,\mathbf{b}) = \{3\}$$

$\longrightarrow$

State	*Input* **a**	*Input* **b**
0	{0, 1}	{0}
1		{2}
2		{3}

Direct Conversion of RE to DFA

From Regular Expression to DFA Directly (Algorithm)

- Augment the regular expression *r* with a special end symbol # to make accepting states important

  r → *(r)#*

- Construct a syntax tree for *(r)#*

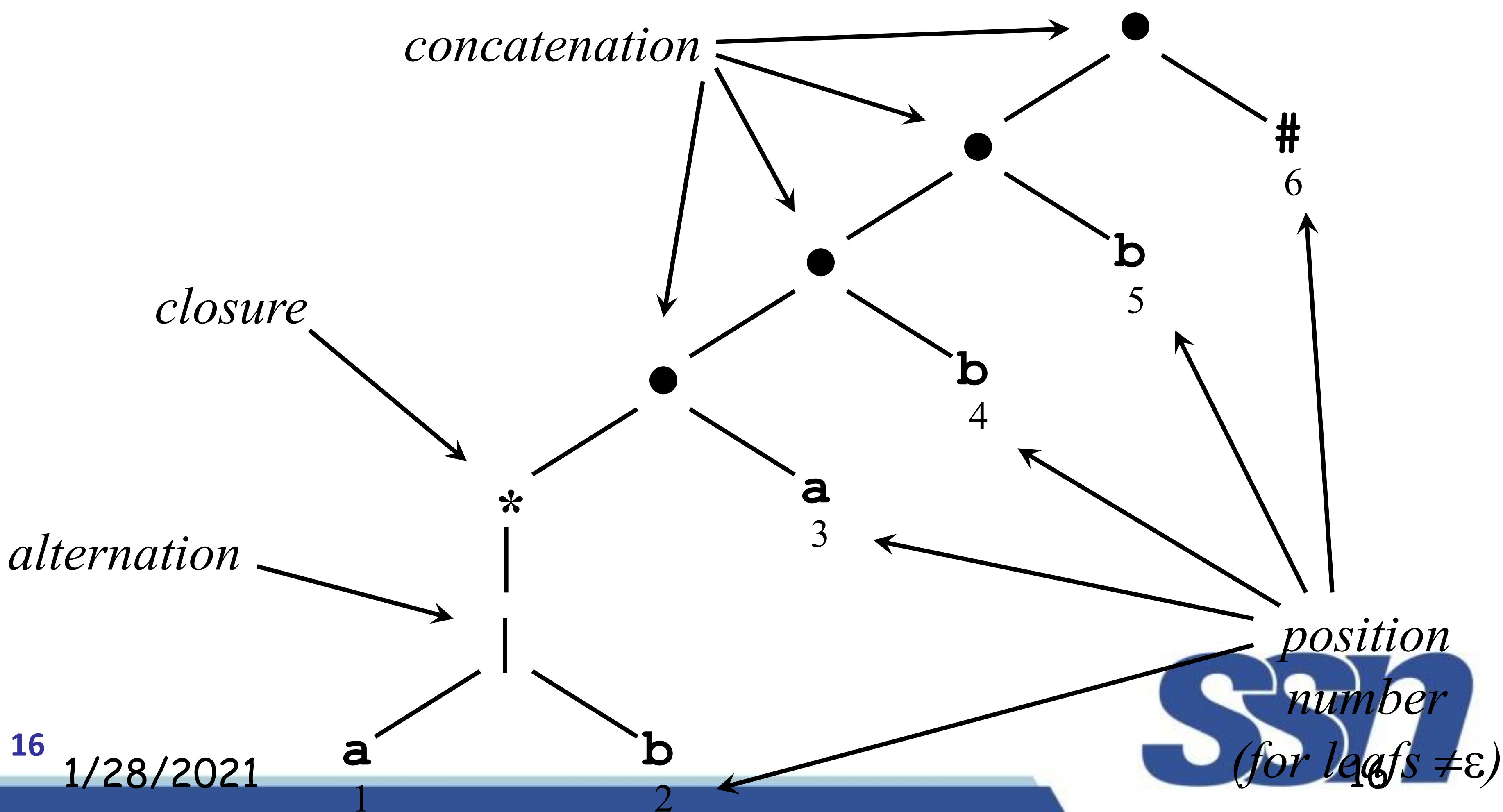
  Alphabets → leaf node

  Operators → inner node

- Number each alphabet including #

- Traverse the tree to construct functions *nullable*, *firstpos*, *lastpos*, and *followpos*

Syntax Tree of (a|b)*abb#



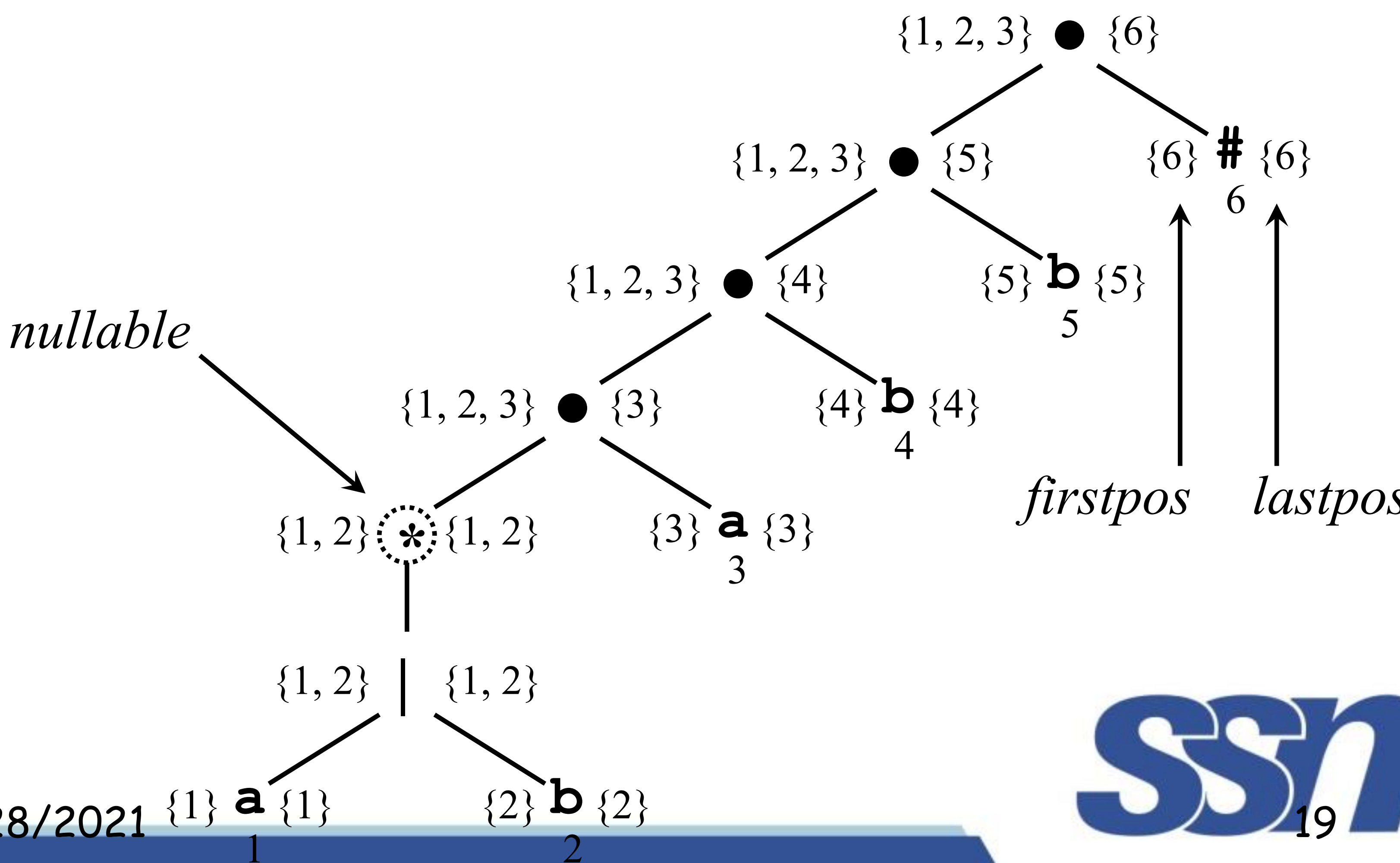
Annotating the Tree

- ***nullable*(*n*):** the subtree at node *n* generates languages including the empty string
- ***firstpos*(*n*):** set of positions that can match the first symbol of a string generated by the subtree at node *n*
- ***lastpos*(*n*):** the set of positions that can match the last symbol of a string generated be the subtree at node *n*
- ***followpos*(*i*):** the set of positions that can follow position *i* in the tree

Annotating the Tree Cont…

Node n	$nullable(n)$	$firstpos(n)$	$lastpos(n)$
Leaf ε	true	$\varnothing$	$\varnothing$
Leaf i	false	$\{i\}$	$\{i\}$
| / \ c_1 c_2	$nullable(c_1)$ or $nullable(c_2)$	$firstpos(c_1)$ $\cup$ $firstpos(c_2)$	$lastpos(c_1)$ $\cup$ $lastpos(c_2)$
• / \ c_1 c_2	$nullable(c_1)$ and $nullable(c_2)$	**if** $nullable(c_1)$ **then** $firstpos(c_1)$ $\cup$ $firstpos(c_2)$ **else** $firstpos(c_1)$	**if** $nullable(c_2)$ **then** $lastpos(c_1)$ $\cup$ $lastpos(c_2)$ **else** $lastpos(c_2)$
* | c_1	true	$firstpos(c_1)$	$lastpos(c_1)$

Syntax Tree of **`(a|b)*abb#`**



followpos

for each node n in the tree **do**
    if n is a cat-node with left child c_1 and right child c_2 **then**
        for each i in $lastpos(c_1)$ **do**
            $followpos(i) := followpos(i) \cup firstpos(c_2)$
        end do
    else if n is a star-node
        for each i in $lastpos(n)$ **do**
            $followpos(i) := followpos(i) \cup firstpos(n)$
        end do
    end if
end do

Algorithm

s_0 := *firstpos*(*root*) where *root* is the root of the syntax tree

Dstates := $\{s_0\}$ and is unmarked

while there is an unmarked state *T* in *Dstates* **do**

  mark *T*

  for each input symbol $a \in \Sigma$ **do**

    let *U* be the set of positions that are in *followpos*(*p*)

      for some position *p* in *T*,

      such that the symbol at position *p* is *a*

    if *U* is not empty and not in *Dstates* **then**

      add *U* as an unmarked state to *Dstates*

    end if

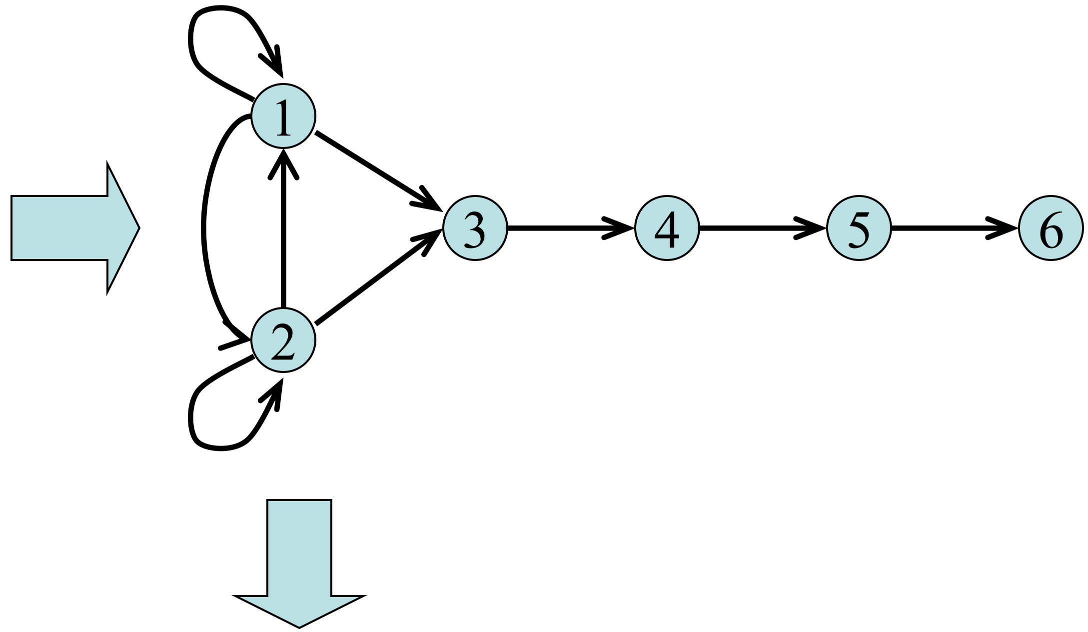
    Dtran[*T*,*a*] := *U*

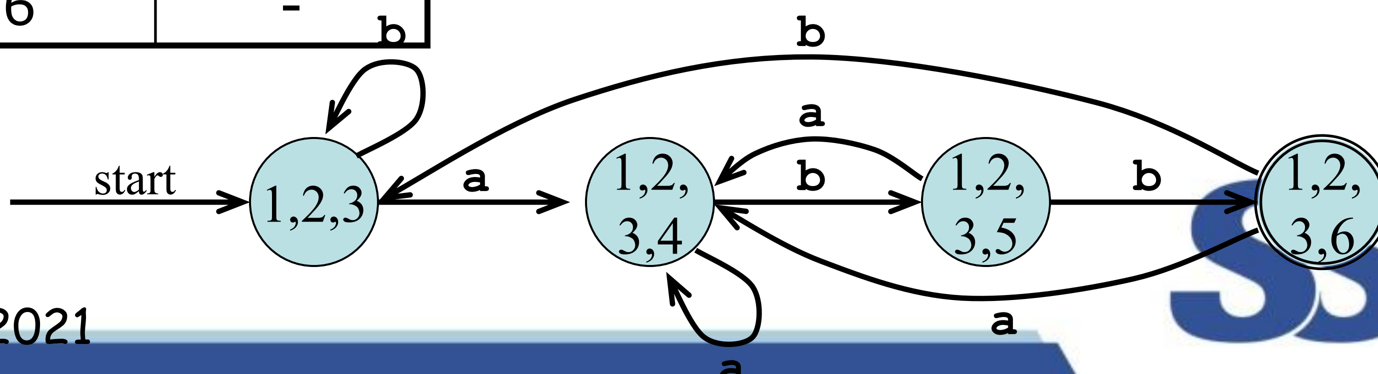
  end do

end do

Example

Node	*followpos*
1	{1, 2, 3}
2	{1, 2, 3}
3	{4}
4	{5}
5	{6}
6	-





DFA states

A= firstpos(root)
A={1,2,3}
Dtran(A,a)=followpos(1) Ufollowpos(3)
= {1,2,3,4} → B
Dtran(A,b)=followpos(2)
= {1,2,3} → A
Dtran(B,a)= followpos(1) Ufollowpos(3)
={1,2,3,4} → B
Dtran(B,b)=followpos(2) Ufollowpos(4)
={1,2,3,5} →C
Dtran(C,a) = followpos(1) Ufollowpos(3)
= {1,2,3,4} → B
Dtran(C,b) = followpos(2) Ufollowpos(5)
={1,2,3,6} → D
Dtran(D,a) = followpos(1) Ufollowpos(3)
= B
Dtran(D,b) = followpos(2)
= {1,2,3} = A

Transition table

States	Input symbol	
	a	b
A	B	A
B	B	C
C	B	D
D	B	A

Summary

- NFA
- DFA
- RE to NFA
- NFA to DFA
- RE to DFA

Check your understanding?

1.Convert the following regular expression into DFA

(a)ba(a/b)*ab

(b) (a* / b*)*

(c) (a/b)*(ac)*