WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques
ECOTE
part 3- Scanner

DSc. dr eng. Ilona Bluemke

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund

Lexical analysis

Scanner reads the source program, one character at a time, translates it into tokens (lexical atoms).

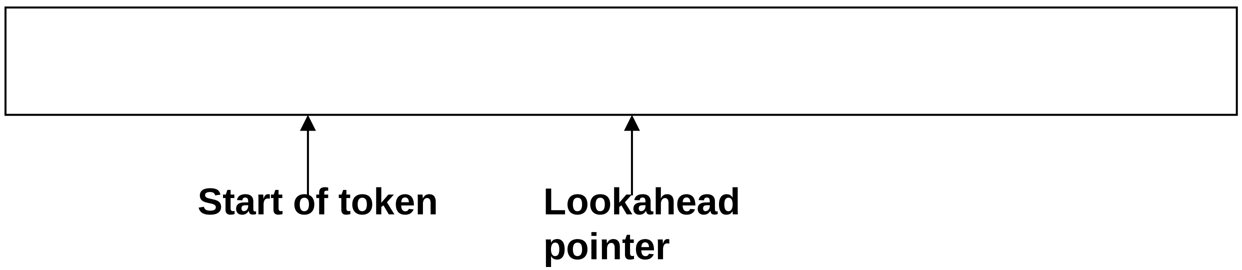
Needs:

- method of describing possible tokens (regular expressions)
- mechanism to recognise these tokens in the input stream (transition diagrams, finite automata)
- mechanism to perform various actions as tokens are recognised (enter into symbol table, generate output, produce diagnostic message)

Other functions:

- keeping track of lines numbers
- produce output listing
- stripping out white space
- delete comments

input buffering



Lookahead pointer – scans ahead until the token is discovered

Conventions:

- at the char last read
- at the char ready to read

Keywords

- Identifiers, often entered initially into the symbol table with the codes for their tokens.

Regular expressions

- Can be automatically converted into finite automata (formal specification of transition diagrams)

Notation:

- Alphabet – character, symbol set **{0,1}**
- String – finite sequence of symbols **001**
- Length of a string x $|x|$
- Empty string ε
- Concatenation of x and y **xy**

  $x \bullet y$

  $x\varepsilon = \varepsilon x = x$
- string x repeated i-times x^i

  $x^0 = \varepsilon$
- closure (any number of) operator *****
- positive closure operator **+**
- or operator **|**
- to group subexpressions **()**

Precedence rules:

* highest

• (product)

| (sum)

Examples:

(a | b)*

a | ba*

Axioms:

1. $r \mid s = s \mid r$ — $\mid$ is commutative
2. $r \mid (s \mid t) = (r \mid s) \mid t$ — $\mid$ is associative
3. $r \bullet (s \bullet t) = (r \bullet s) \bullet t$ — $\bullet$ is associative
4. $r \bullet (s \mid t) = r \bullet s \mid r \bullet t$ — $\bullet$ distributes over $\mid$

   $(s \mid t) \bullet r = s \bullet r \mid t \bullet r$
5. $\varepsilon r = r \varepsilon = r$ — ε identity for concatenation

Finite automata

Recognizer of a language L:

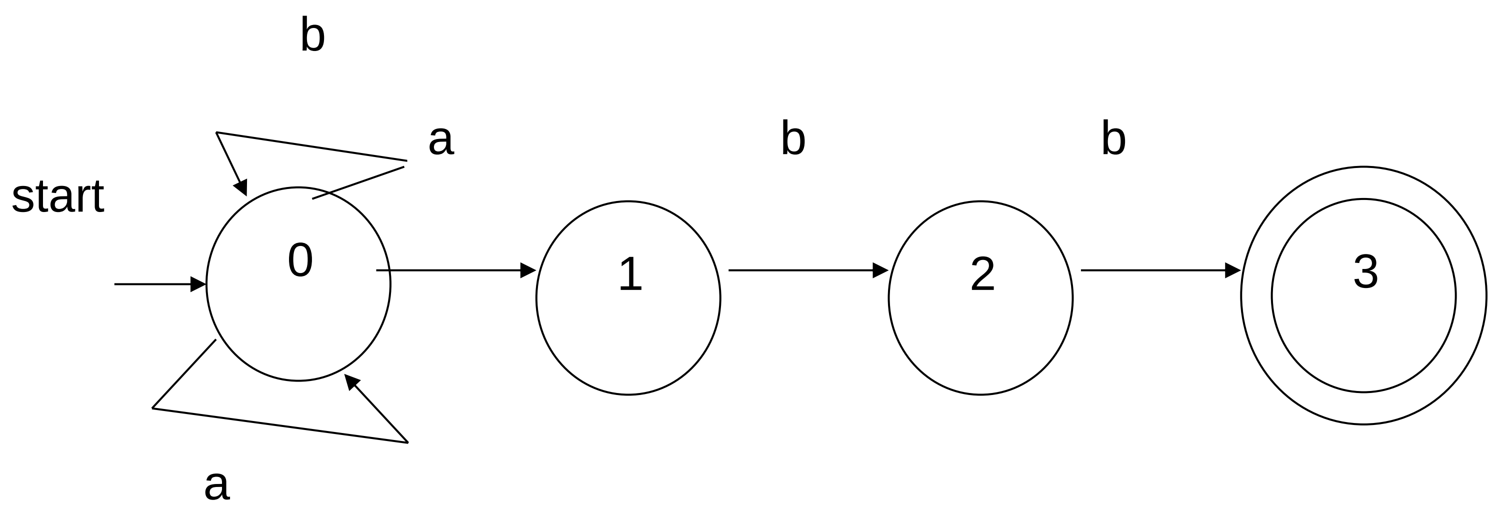
takes as input string x and

- answers „yes” if x is a sentence of L
- „no” otherwise

NFA – nondeterministic finite automata

- generalised transition diagram – edges can be labelled by ε,
- the same char can label more than one transition

NFA for (a | b)*abb



State 3 – **final state, accepting state**

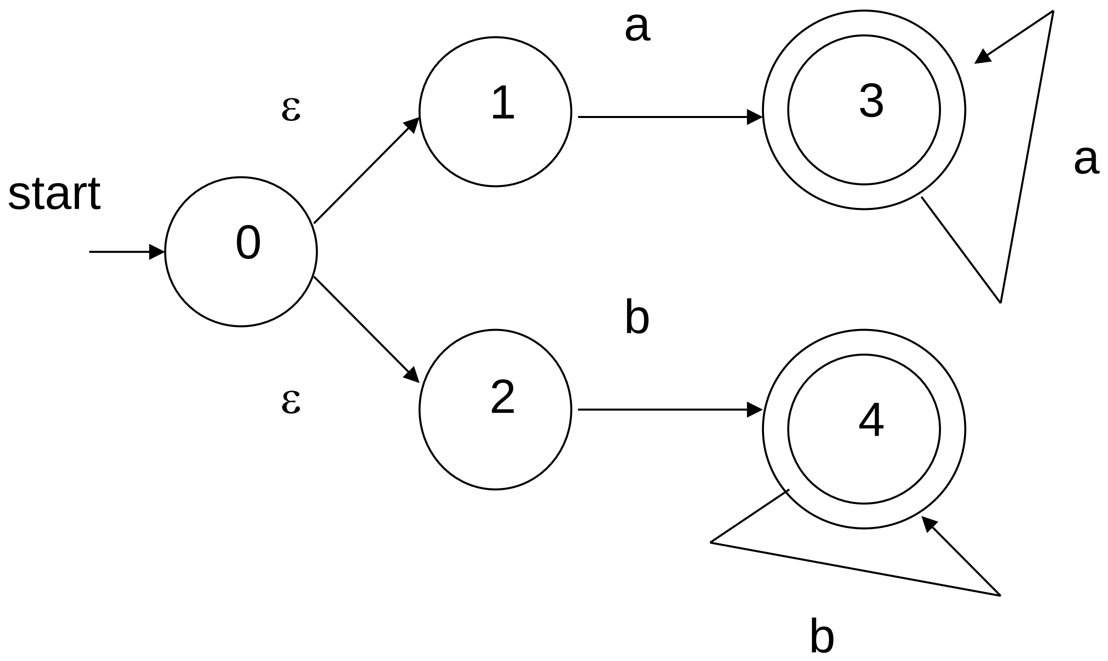
NFA can be represented in transition table:

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

NFA accepts an input string x if there is a path from the starting state to some accepting, final state, such that the labels along that path spell out x.

The language defined by NFA is the set of input strings it accepts.

Example NFA accepting aa* | bb*



All paths with the same label must be considered before NFA finds out that a string leads to final state – time

DFA – deterministic finite automata

- no transition on input **ε**
- for each state ***s*** and input symbol ***a*** there is at most one edge labelled ***a*** leaving ***s***.

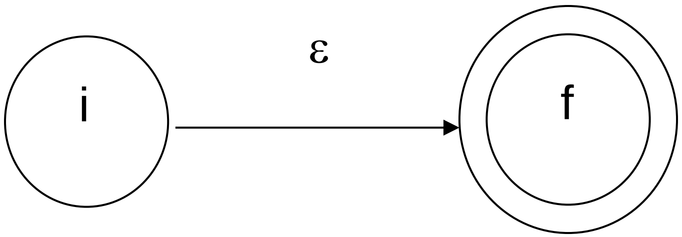
For **each NFA a DFA accepting the same language can be found.**

The number of states of DFA could be exponential in the number of states of NFA.

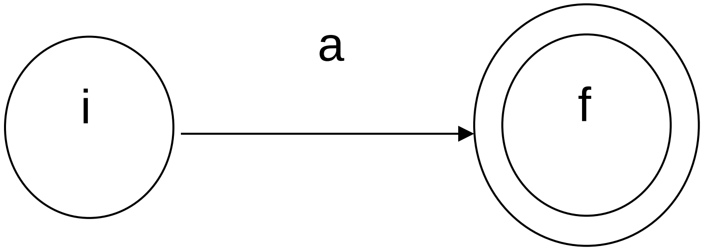
From regular expression to NFA – Thompson construction algorithm

1. Decompose *R* into its primitive components
2. For each component construct a NFA

a) for ε



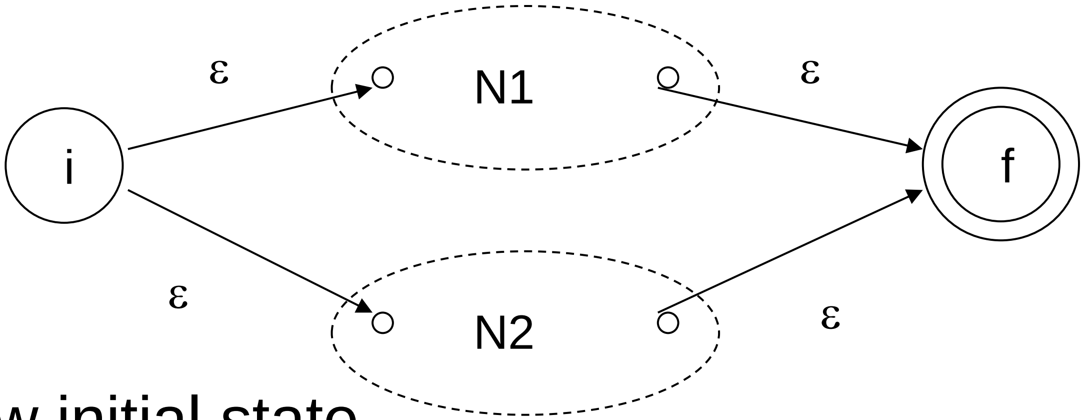
b) for a (a is a symbol in alphabet)



3. Each time a new state is needed **new name** is given to that state
4. Having constructed components we proceed to combine them in a way that correspond to the way compound regular expressions are formed from smaller ones.
5. For all regular expressions a NFA with one initial and one final state is constructed, no edge enters the initial state or leaves the final.

Suppose N1 and N2 are NFA for regular expressions R1 and R2

for R1 | R2



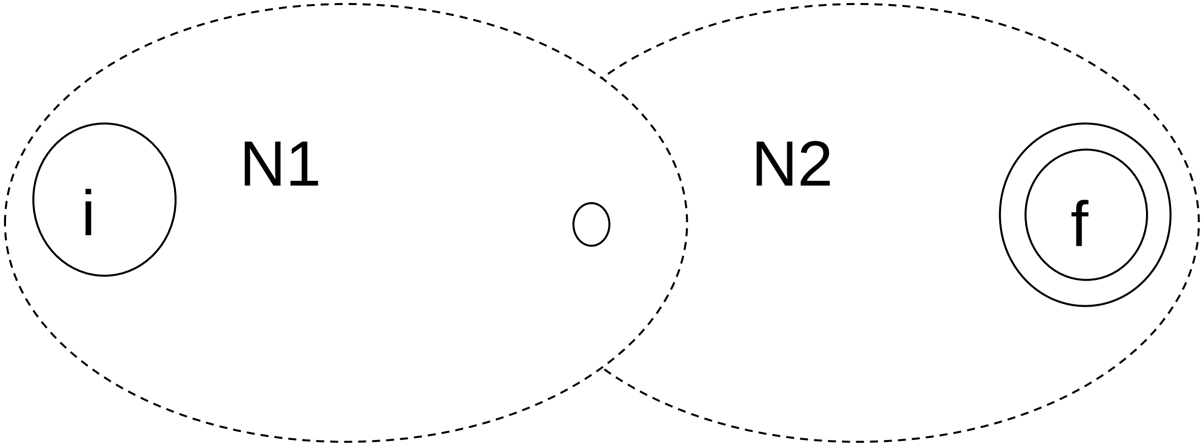
i – new initial state

f – new final state

ε transitions to initial states of N1, N2

ε transitions from the final states of N1, N2 (no longer final)

for R1•R2 (R1R2)

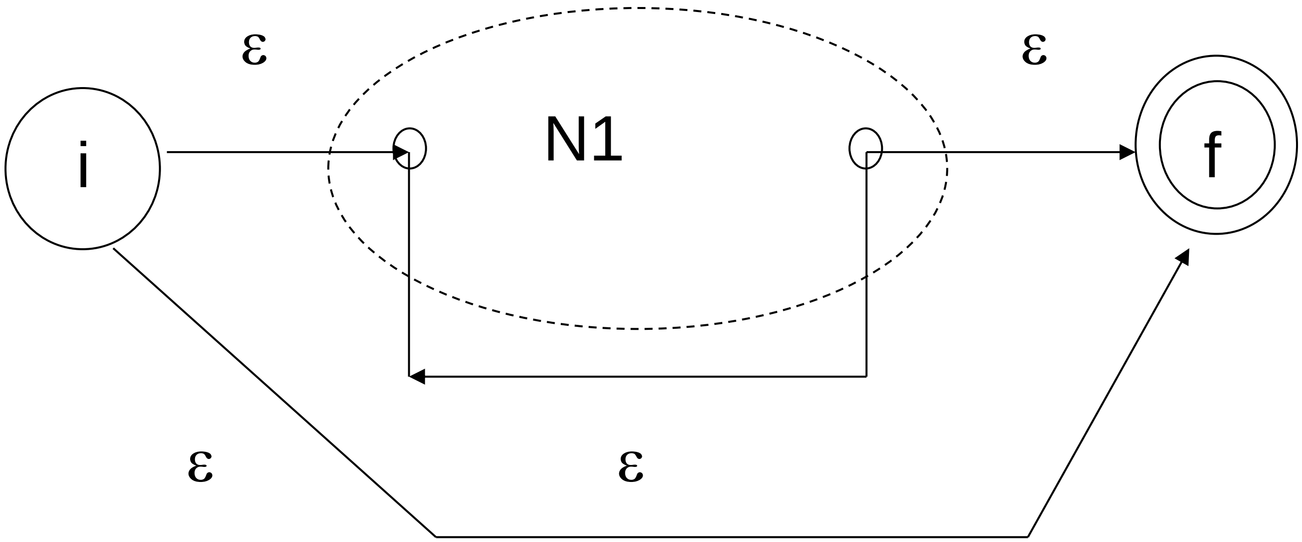


i – initial state (initial state of N1)

f – final state (final state of N2)

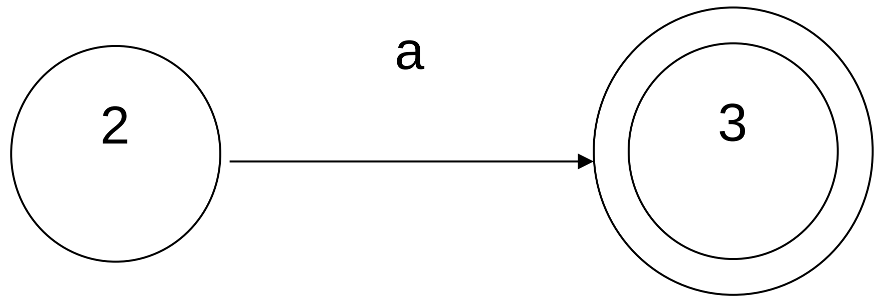
final state of N1 and initial state of N2 are concatenated

for R1*

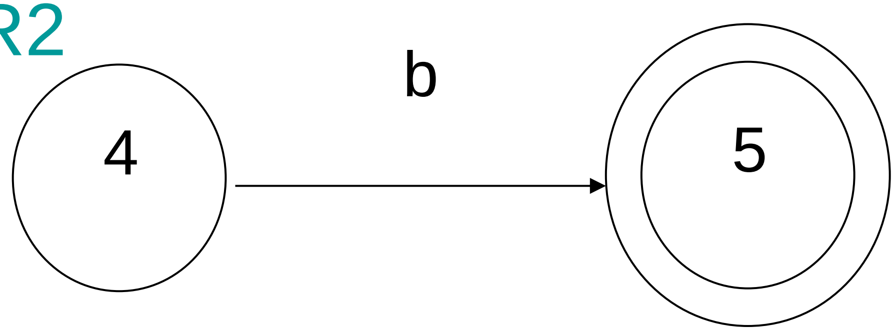


Example $R = (a \mid b)^* abb$

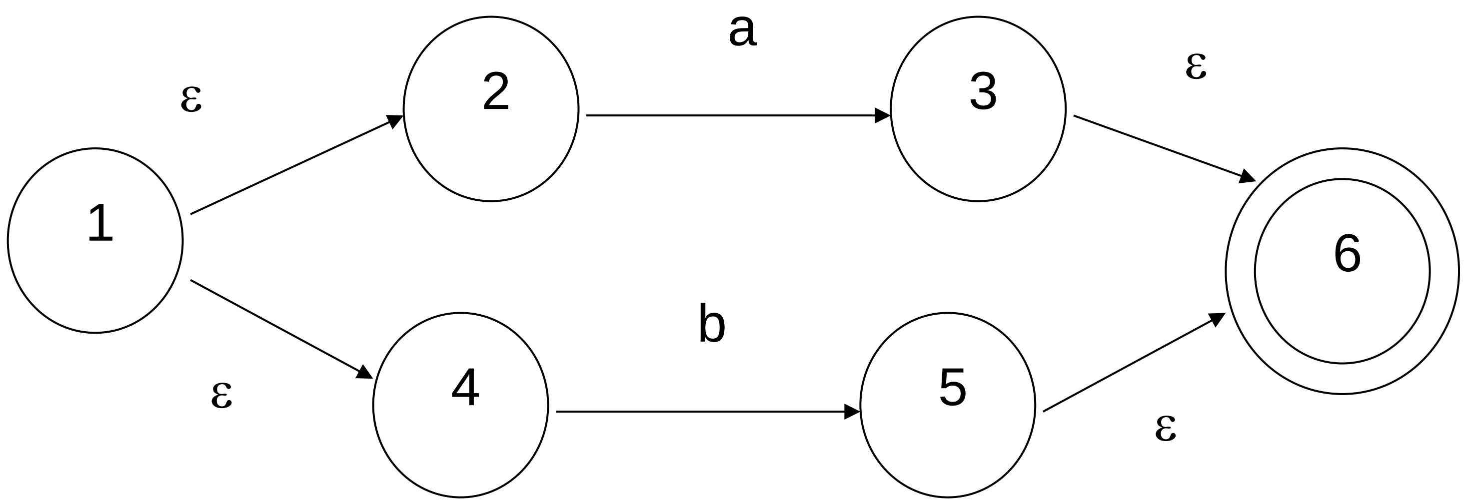
N1: for R1



N2: for R2

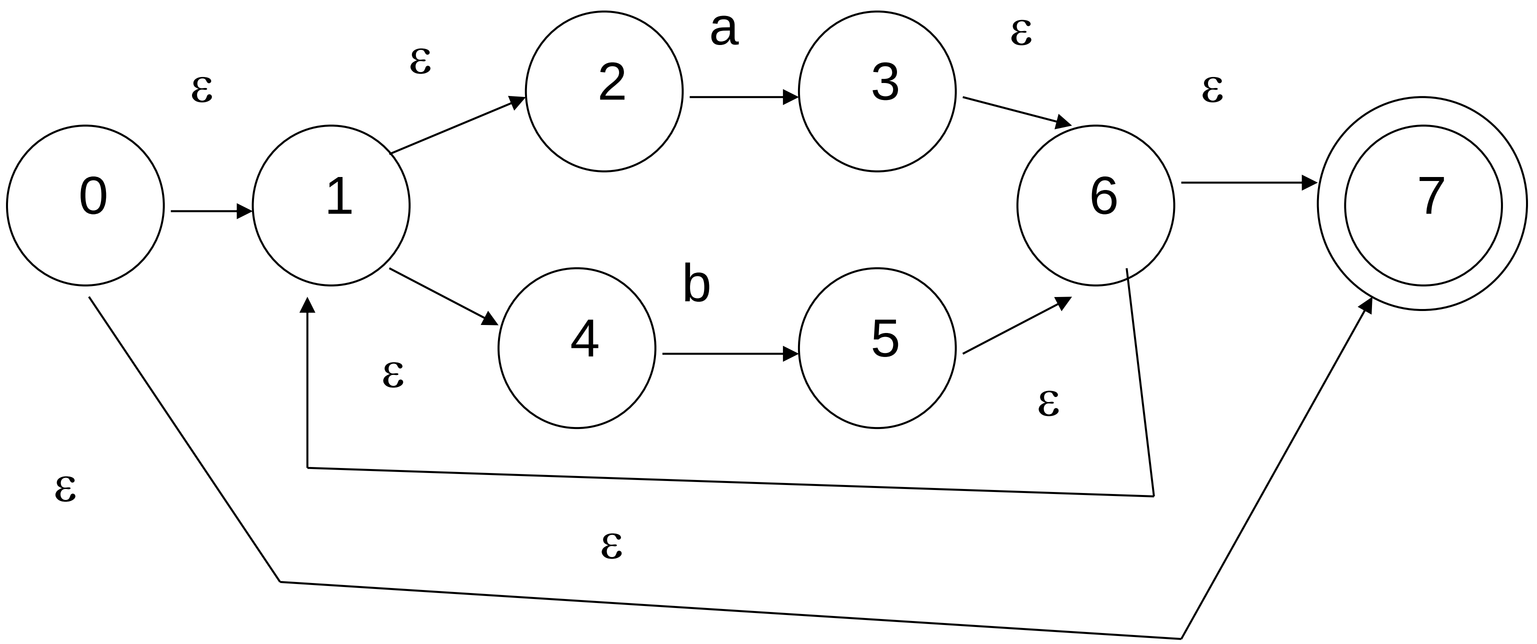


N3: R3=R1 | R2

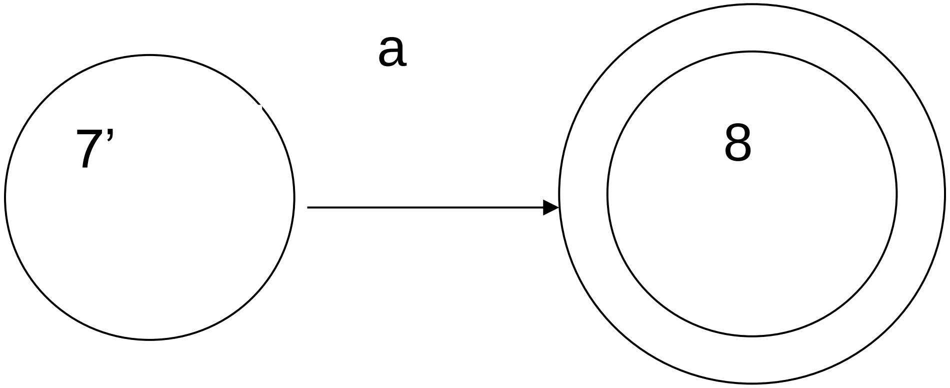


N4: R4= (R3)=R3

N5: R5=R4*



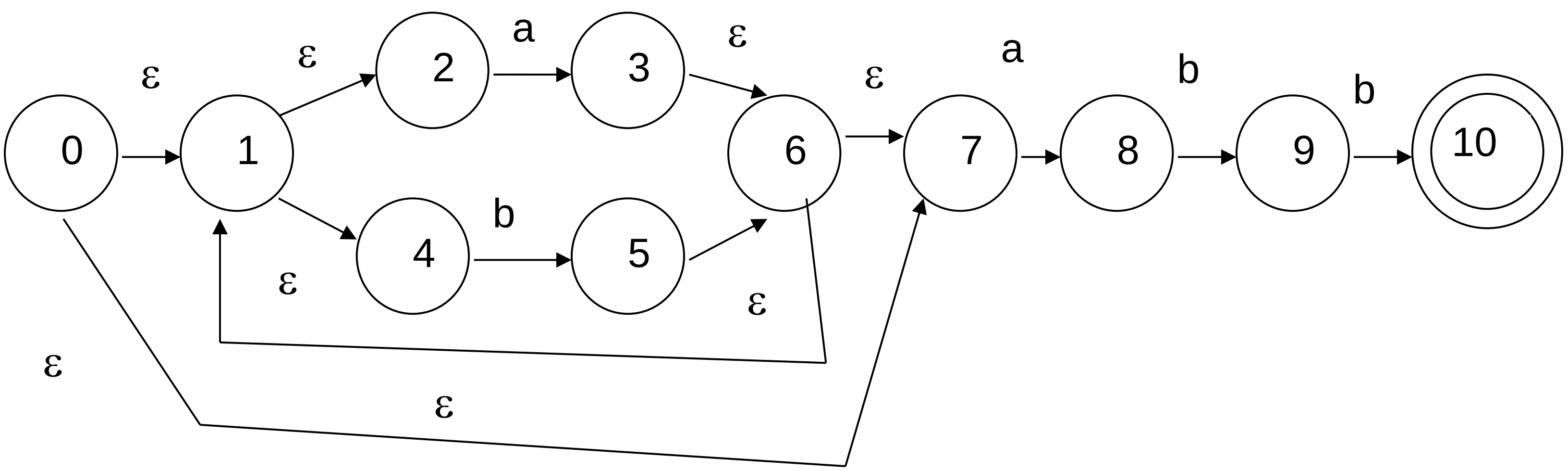
N6: R6=a



N7: R7=R5R6

Identify 7 with 7’

Result:



NFA ⇒ DFA
(subset construction algorithm)

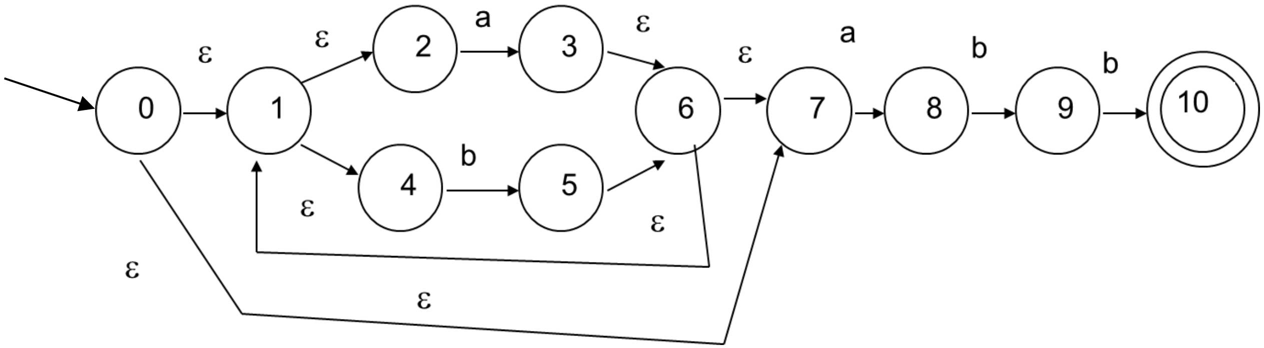
- Each state of DFA is a **set of states of NFA**, which NFA could be in after reading some sequence of input symbols.
- Initial state of DFA – the set consisting $\mathbf{s_0}$ (initial of NFA) together with all states of NFA that can be reached **from $\mathbf{s_0}$ by ε-transitions** only
- Final states of DFA – set of states of NFA that contain at least one final state of NFA

Function ε-closure(s)

Set of states of NFA built by applying rules:

1. **s** is added to **ε-closure(s)**
2. If ***t*** is in **ε-closure(s)** and there is an edge labelled **ε** from ***t*** to ***u***, then ***u*** is added to ε-closure(s) if ***u*** is not already there.
3. Repeat rule 2 until no more states can be added to **ε-closure(s)**

Examples ε-closure



- ε-closure (0) :
  {0,1,2,4,7}
- ε-closure (3) :
  {3, 6,7,1,2,4}
- ε-closure (1) :
  {1, 2,4}

Construction of DFA states and their transitions

- Initial DFA state = **ε-closure(s_0)** where ***s_0*** is an initial NFA state
- Initially all DFA states are **unmarked**
- **ε-closure(*T*)** –set of NFA states reachable from a state ***s*** in ***T*** only by **ε-transitions**
- **move(*T*,*a*)** set of NFA states reachable from a state ***s*** in ***T*** only by ***a*-transitions**
- **Dstates** – set of DFA states
- **Dtran** – DFA transition table

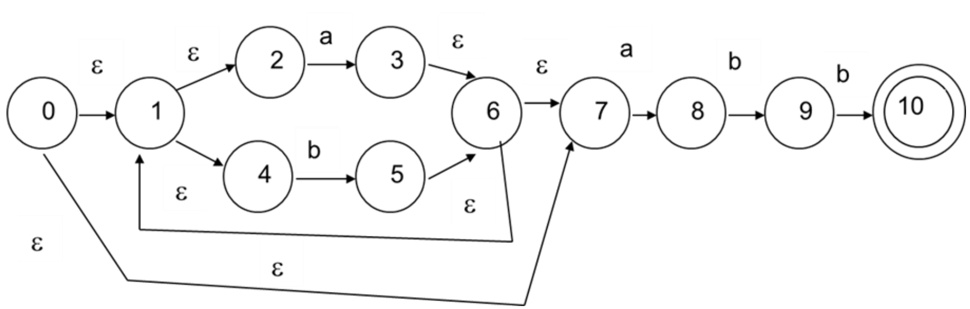
Perform algorithm:

```
while there is an unmarked state T in Dstates
do begin
    mark T;
    for each input symbol a
        do begin
        U := ε-closure(move(T,a));
        if U not in Dstates
        then
        add U as an unmarked state to Dstates;
                Dtran[T,a] := U;
        end
    end
```

Example

Initial DFA state

- A = ε-closure(0) = {0,1,2,4,7}
- ε-closure(move(A,a))= ε-closure (move ({0,1,2,4,7},a)) = ε-closure({3,8}) = {1,2, 3,4,6,7,8} = B
- Dtran[A,a] = B
- ε-closure(move(A,b))= ε-closure (move ({0,1,2,4,7},b)) = ε-closure({5}) = {1,2,4,5,6,7} = C
- Dtran[A,b] = C
- ε-closure(move(B,a))= ε-closure({3,8})=B
- ε-closure(move(B,b))= ε-closure({5,9}) ={1,2,4,5,6,7,9} = D

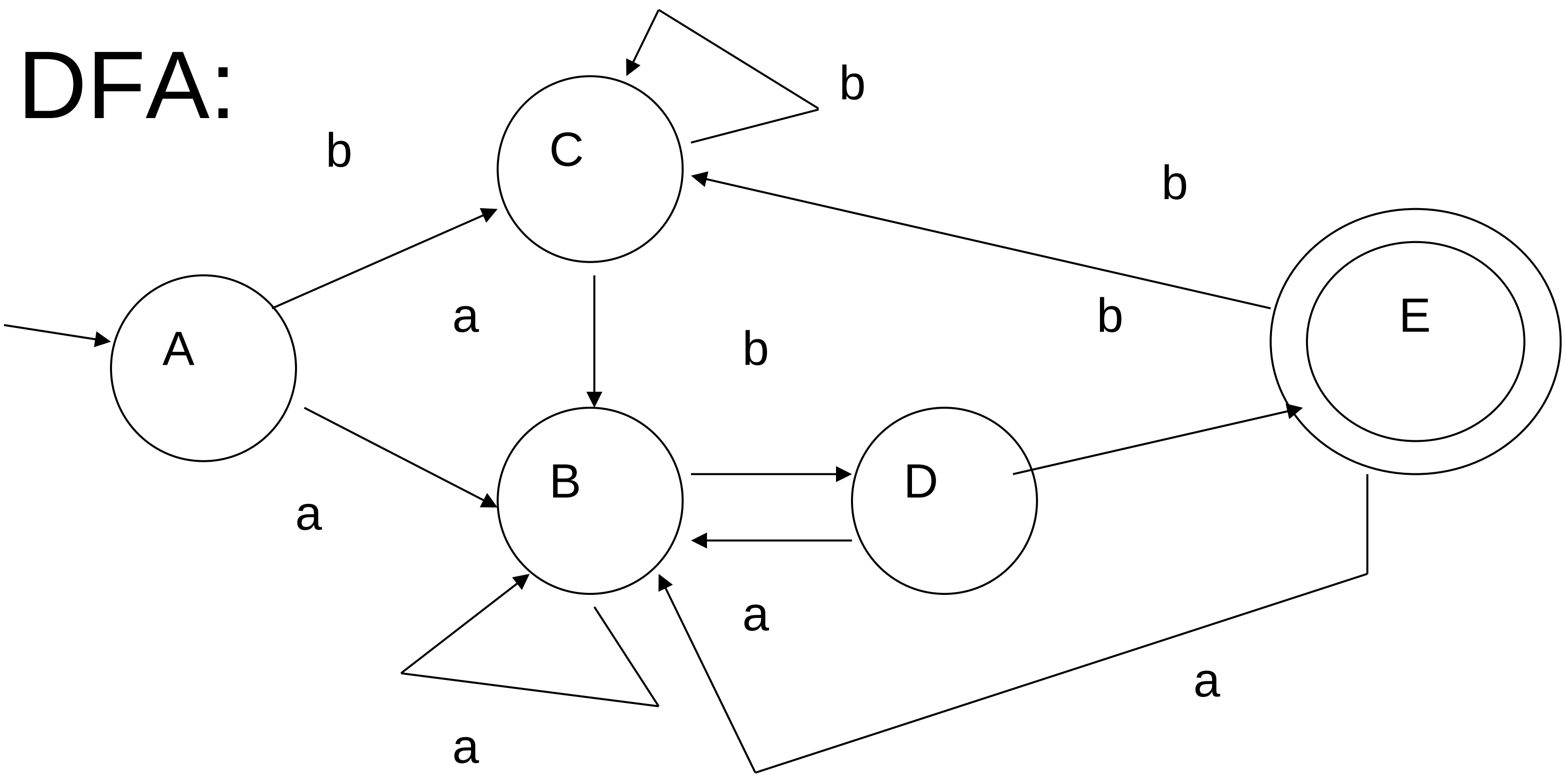


- ε-closure(move(C,a))= ε-closure({3,8})=B
- ε-closure(move(C,b))= ε-closure({5})=C
- ε-closure(move(D,a))= ε-closure({3,8})=B
- ε-closure(move(D,b))= ε-closure({5,10}) = {1,2,4,5,6,7,10} = E – final state
- ε-closure(move(E, a))= ε-closure({3,8})=B
- ε-closure(move(E,b))= ε-closure({5})=C

Transition table

State	a	b
A={0,1,2,4,7}	B	C
B={1,2,3,4,6,7,8}	B	D
C={1,2,4,5,6,7}	B	C
D={1,2,4,5,6,7,9}	B	E
E={1,2,4,5,6,7,10} Final state	B	C

DFA:



Direct construction of DFA from regular expression

1. Concatenate regular expression ***r*** with **#** (special symbol), so ***r*** accepting state is an **important state** (having non-ε out transition)
2. Construct syntax tree for augmented regular expression ***r#***:

- **Nodes** – operators
- **Leafs** – input symbols or ε
- Leafs with input symbols receive unique identifying number – **position**

3. For each node, leaf compute functions:

- **nullable(*n*)** – true if from node ***n*** empty string can be generated
- **first(*n*)** - gives the set of positions that can match the first symbol of a string generated by the subexpression rooted at ***n***
- **last(*n*)** - gives the set of positions that can match the last symbol of a string generated by the subexpression rooted at ***n***
- **follow(*p*)** if ***p*** is a position, then follow(*p*) is the set of positions ***j*** such that there is some input string ...cd.. such that *p* corresponds to this occurrence of c and *j* to this occurrence of d.

4. Construct DFA states and transition table

   Initial state – first(root) of the syntax tree

   Final state – state with # position

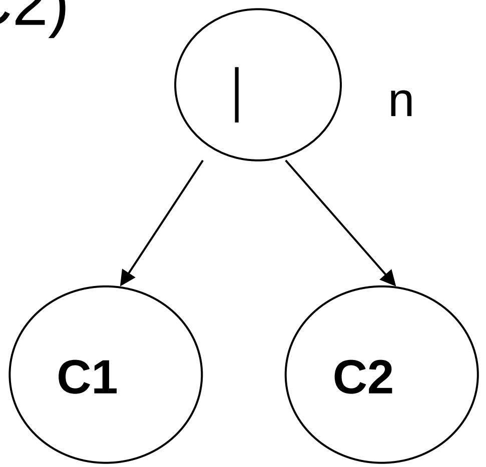
   Add **first(root)** as unmarked state in ***Dstates*;**

while there is an unmarked state ***T*** in *Dstates*
do begin
    mark ***T***;
    for each input symbol ***a***
        do begin
        let ***U*** be the set of position that are in ***follow(p)***
        for some ***p*** in *T*, such that the symbol at position ***p*** is
            a;
        if ***U*** is not empty and is not in *Dstates*
        then add ***U*** as an unmarked state *T* to *Dstates;*
        Dtran[T,a] := U;
        end
    end

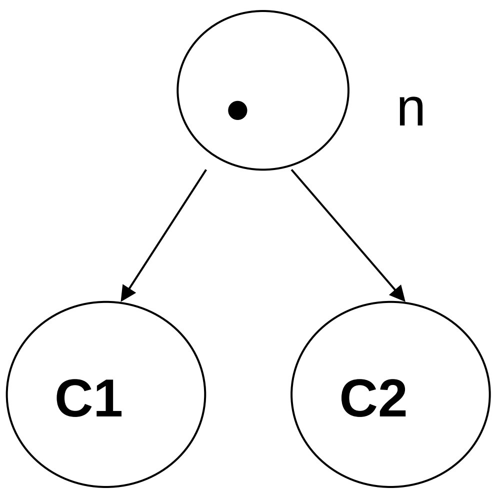
WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Function nullable

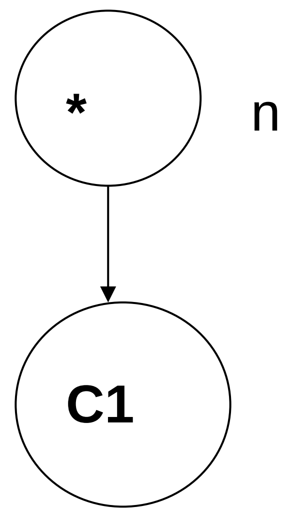
node *n*	nullable(*n*)
1. *n* – leaf with **ε**	true
2. *n* – leaf with position ***i***	false
3. nulable(C1) **or** nulable(C2)	



4.nulable(C1) **and** nulable(C2)

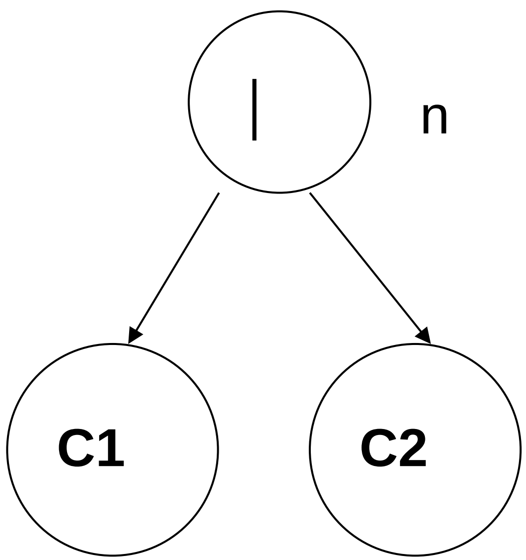


5. true



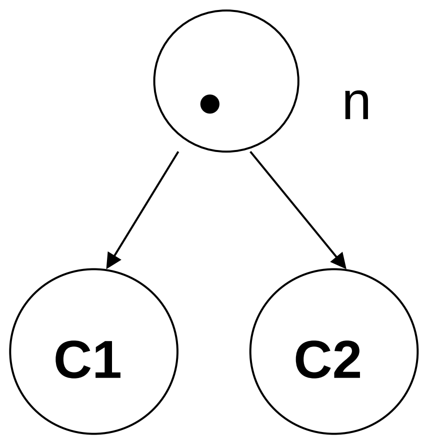
Functions first and last

node ***n***	**first(*n*)**	**last(*n*)**
1. *n* – leaf with **ε**	ϕ	ϕ
2. *n* – leaf with position ***i***	{i}	{i}
3.	first(C1) ∪ first(C2)	last(C2) ∪ last(C1)



node ***n*** — **first(*n*)** — **last(*n*)**

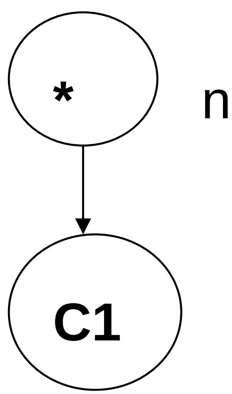
4.



first(*n*): if nullable(C1) then first(C1) ∪ first(C2) else first(C1)

last(*n*): if nullable(C2) then last(C2) ∪ last(C1) else last(C2)

5.



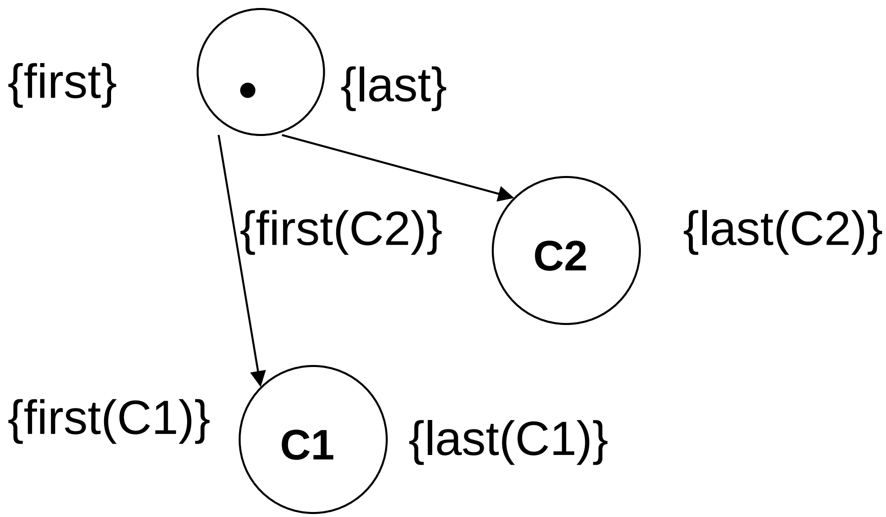
first(*n*): first(C1)

last(*n*): last(C1)

Function follow(*i*)

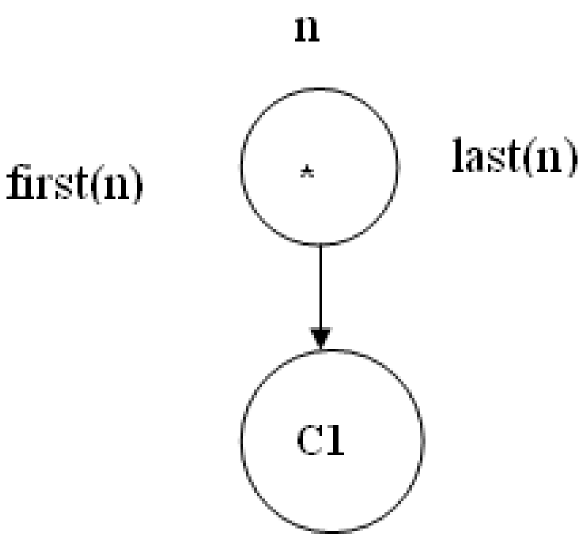
Tells what position can follow position ***i*** in the syntax tree.

1. concatenation node



{i} ∈ last(C1), then all positions in first(C2) are in follow(i)

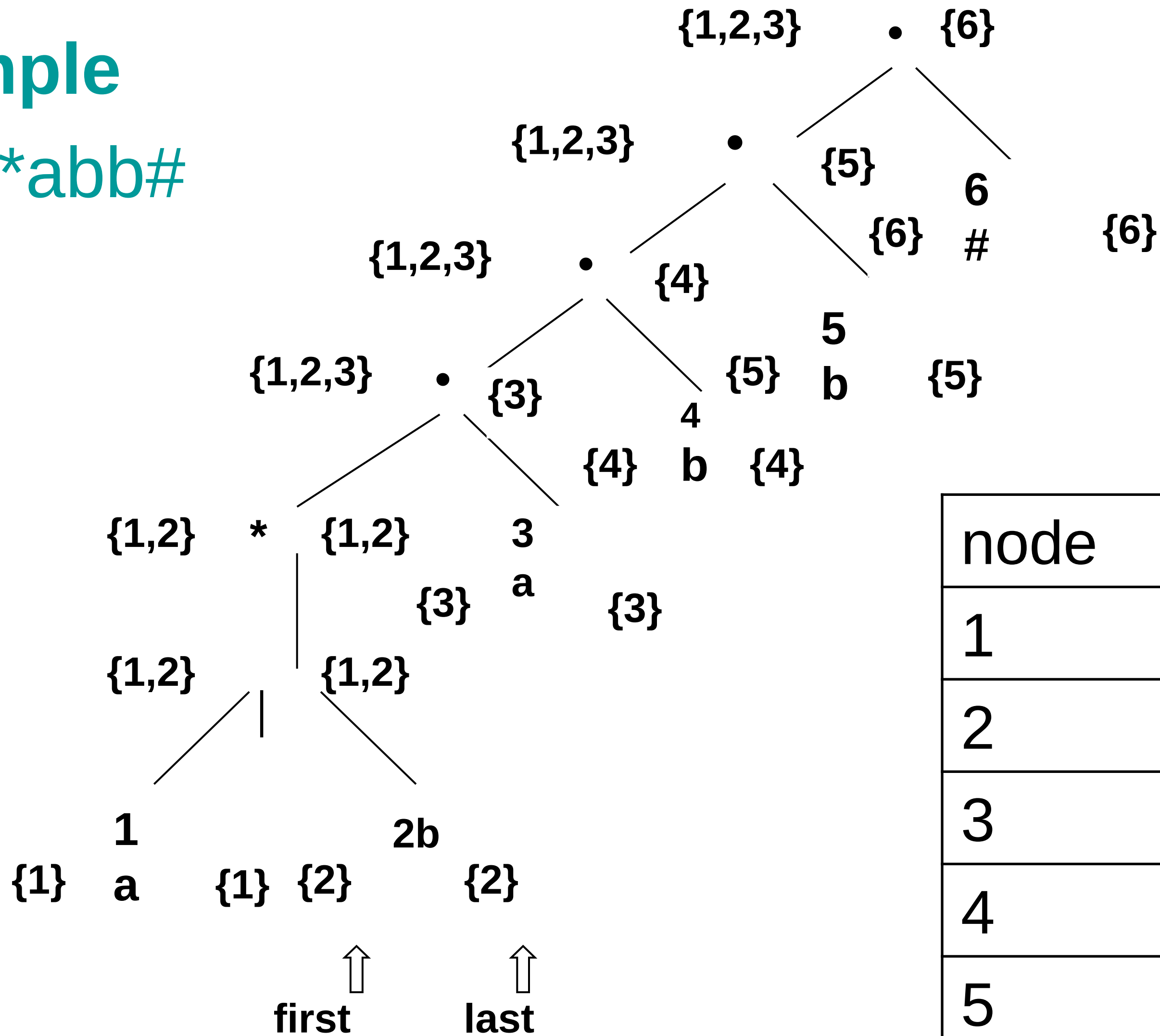
2. star node



{i} ∈ last(n), then all positions in first(n) are in follow(i)

Example

(a | b)*abb#



node	follow
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	Φ

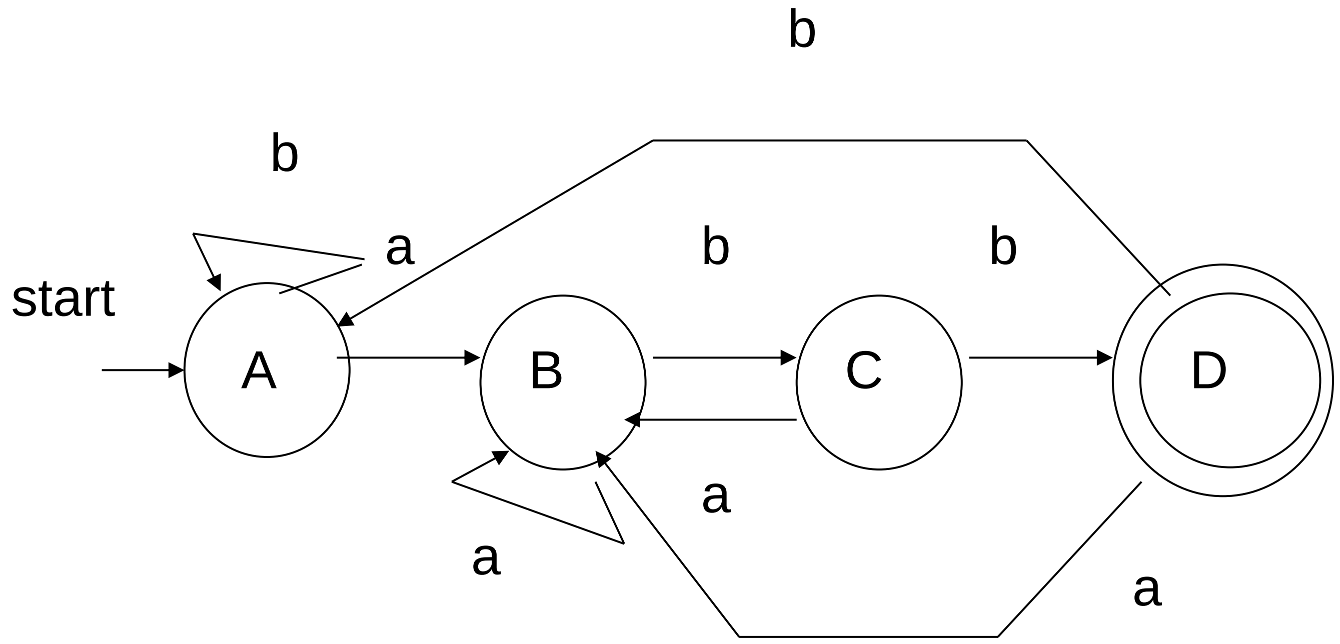
Initial state: A = first(root) = {1,2,3}

Transition table:

- Dtran[A,a] = follow(1) ∪ follow(3) = {1,2,3,4}=B
- Dtran[A,b] = follow(2) = {1,2,3} = A
- Dtran[B,a] = follow(1) ∪ follow(3) = {1,2,3,4}=B
- Dtran[B,b] = follow(2) ∪ follow(4) = {1,2,3,5}=C
- Dtran[C,a] = follow(1) ∪ follow(3) = {1,2,3,4}=B
- Dtran[C,b] = follow(2) ∪ follow(5) = {1,2,3,6}=D
- Dtran[D,a] = follow(1) ∪ follow(3) = {1,2,3,4}=B
- Dtran[D,b] = follow(2) = {1,2,3} = A

Final state D – position 6 (#)

DFA



Methods:

- minimising the number of states
- table compression

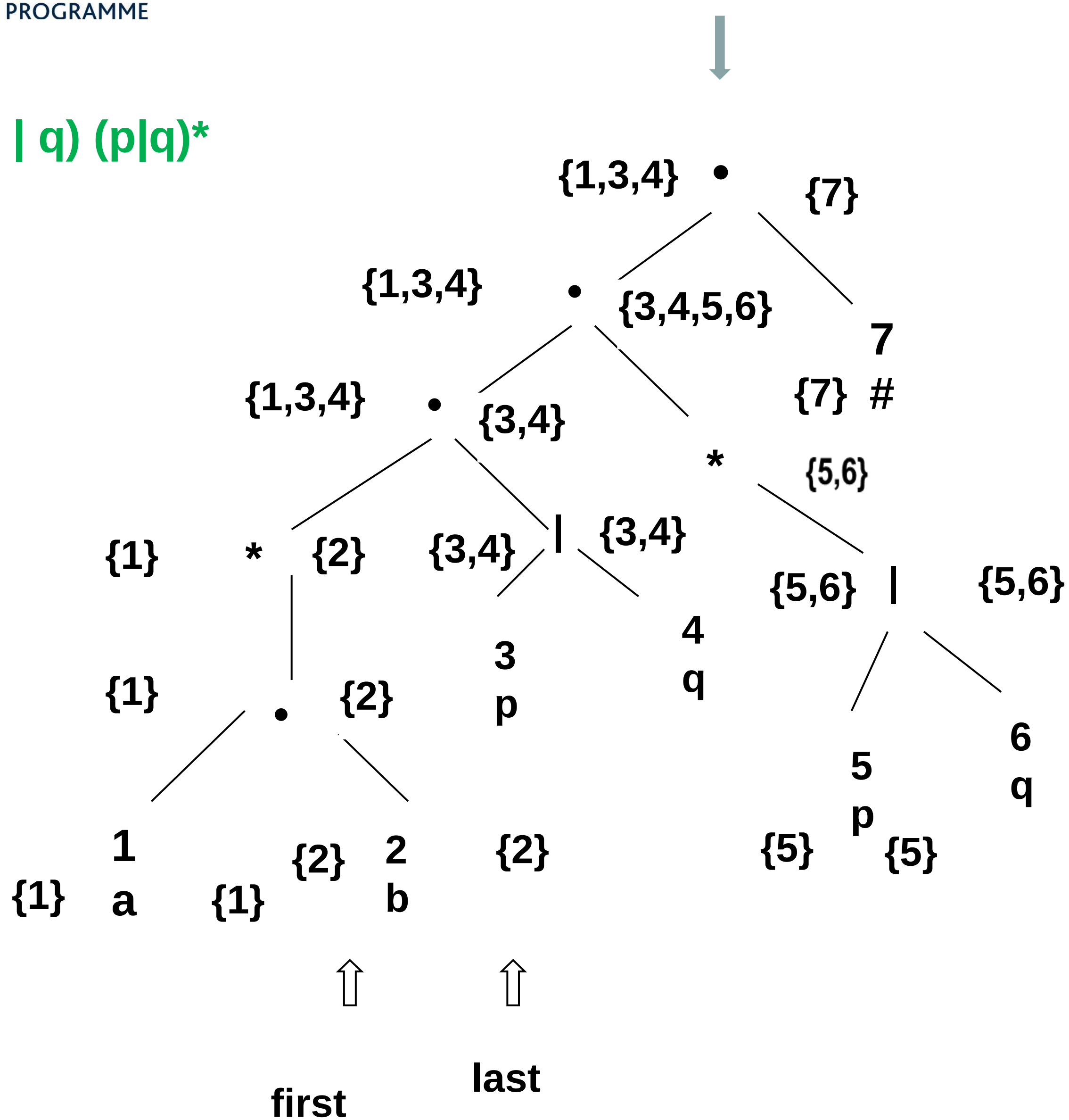
RE into DFA

- Find DFA for regular expression (with syntax tree):

(ab)* (p | q)+

(ab)* (p | q) (p|q)*

(ab)* (p | q) (p|q)*


{1,3,4}
•
{7}
{1,3,4}
•
{3,4,5,6}
7
{7}
#
{1,3,4}
•
{3,4}
*
{5,6}
{1}
*
{2}
{3,4}
|
{3,4}
{5,6}
|
{5,6}
3
p
4
q
{1}
•
{2}
5
p
6
q
1
a
{1}
{2}
2
b
{1}
{2}
{5}
{5}
first
last

Follow

1. {2}
2. {1, 3,4]
3. {5,6,7}
4. {5,6,7}
5. {5,6,7}
6. {5,6,7}
7. Φ

Transitions

Initial A= {1,3,4}

- Dtran [A,a] = follow(1) = {2} = B
- Dtran [A,p] = follow(3) = {5,6,7}=C
- Dtran [A,q] = follow(4) = {5,6,7}=C
- Dtran [B,b] = follow(2) = {1,3,4} = A
- Dtran [C,p] = follow(5) = {5,6,7}=C
- Dtran [A,q] = follow(6) = {5,6,7}=C

C-FINAL

WARSAW UNIVERSITY OF TECHNOLOGY
DEVELOPMENT PROGRAMME

Compiling Techniques
ECOTE
part 3- Scanner

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund