Languages, grammars, automata

English sources:

[1] B. Melichar, J. Holub, T. Polcar: **Text Search Algorithms**
http://cw.felk.cvut.cz/lib/exe/fetch.php/courses/a4m33pal/melichar-tsa-lectures-1.pdf
Chapters 1.4 and 1.5, it is probably reasonably short, there is nothing to skip.

[2] J. E. Hopcroft, R. Motwani, J. D. Ullman: **Introduction to Automata Theory**
folow the link at http://cw.felk.cvut.cz/doku.php/courses/a4m33pal/literatura_odkazy
Chapters 1., 2., 3., there is a lot to skip, consult the teacher preferably.

Czech instant sources:

[3] M. Demlová: **A4B01JAG**
http://math.feld.cvut.cz/demlova/teaching/jag/
Pages 1-27, in PAL, you may wish to skip: Proofs, chapters 2.4, 2.6, 2.8.

[4] I. Černá, M. Křetínský, A. Kučera: **Automaty a formální jazyky I**
http://is.muni.cz/do/1499/el/estud/fi/js06/ib005/Formalni_jazyky_a_automaty_I.pdf
Chapters 1 and 2, skip same parts as in [1].

For more references see PAL links pages
https://cw.fel.cvut.cz/wiki/courses/be4m33pal/references (EN)
http://cw.felk.cvut.cz/doku.php/courses/b4m33pal/odkazy-zdroje (CZ)

Alphabet

<u>Alphabet</u> … finite (unempty) set of symbols
|A| … <u>size</u> of alphabet A

Examples: A = { 'A', 'D', 'G', 'O', 'U'}, |A| = 5
A = {0,1}, |A| = 2
A = {○, □, △}, |A| = 3

word

<u>Word</u> (over alphabet A) … finite (maybe empty) sequence
also <u>string</u> of symbols of alhabet (A)
|w| … <u>length</u> of word w

Examples: w = OUAGADOUGOU, |w| = 11
w = 1001, |w| = 4
w = □△○△□, |w| = 5

Language

<u>Language</u> … set of words (=strings)
(not necessarily finite, can be empty)
|L| … <u>cardinality</u> of language L

---

1 Language specification -- List of all words of the language (only for finite language!)

Examples: A_1 = {‘A’, ‘D’, ‘G’, ‘O’, ‘U’}

L_1 = {ADA, DOG, GOUDA, D, GAG}, $|L_1| = 5$

$A_2 = \{0,1\}$

L_2 = {0, 1, 00, 01, 10, 11}, $|L_2| = 6$

A_3 = { ○, □, △}

L_3 = { △△, ○□○, □□△○}, $|L_2| = 3$

2 **Language specification** -- Informal (but unambiguous) description in natural human language (usually for infinite language)

Examples: A_1 = {'A', 'D', 'G', 'O', 'U'}

L_1: Set of all words over A_1, which begin with DA, end with G a and do not contain subsequence AA.

L_1 = {DAG, DADG, DAGG, DAOG, DAUG, DADAG, DADDG… }

$|L_1| = \infty$

A_2 = {0,1}

L_2: Set of all words over A_2, which contain more 1s than 0s and each 0 is followed by at least two 1s.

L_2 = { 1, 11, 011, 0111, 1011, 1111, … , 011011, 011111, … }

$|L_2| = \infty$

3 Language specification -- By finite automaton

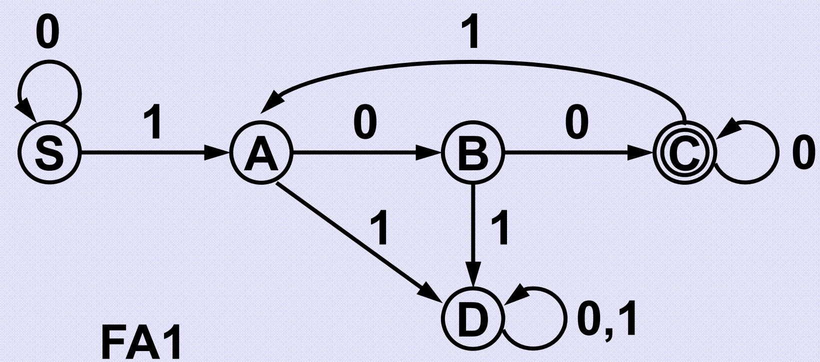
Finite automaton
is a five-tuple (A, Q, σ, S_0, Q_F), where:

A … <u>alphabet</u> … finite set of symbols
  $|A|$ ... size of alphabet
Q ... set of <u>states</u> (often numbered) (what is „a state“ ?)
σ ... <u>transition function</u> ... $\sigma: Q \times A \rightarrow Q$
S_0 ... <u>start state</u> $S_0 \in Q$
Q_F ... unempty set of <u>final states</u> $\emptyset \neq Q_F \subseteq Q$

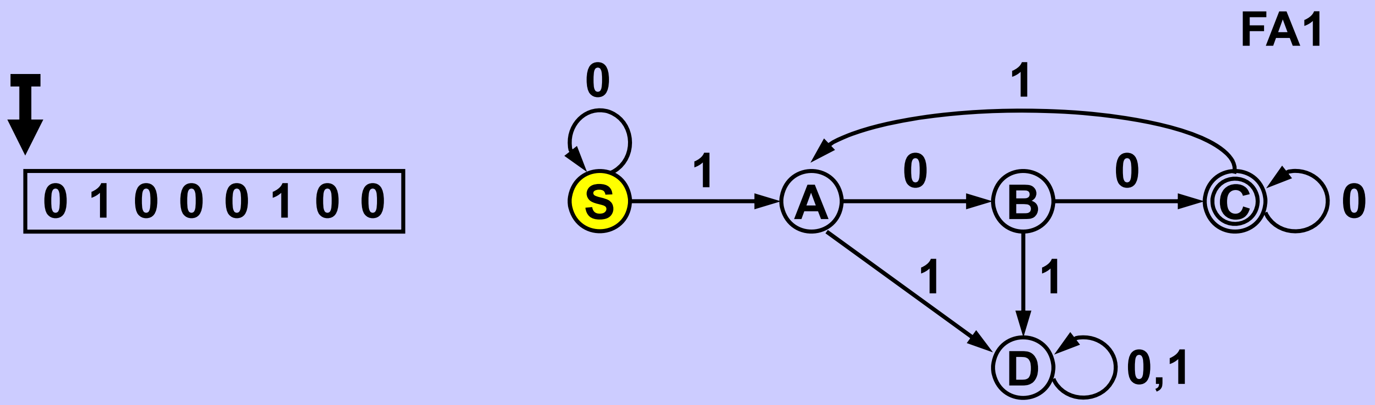
Automaton FA1:

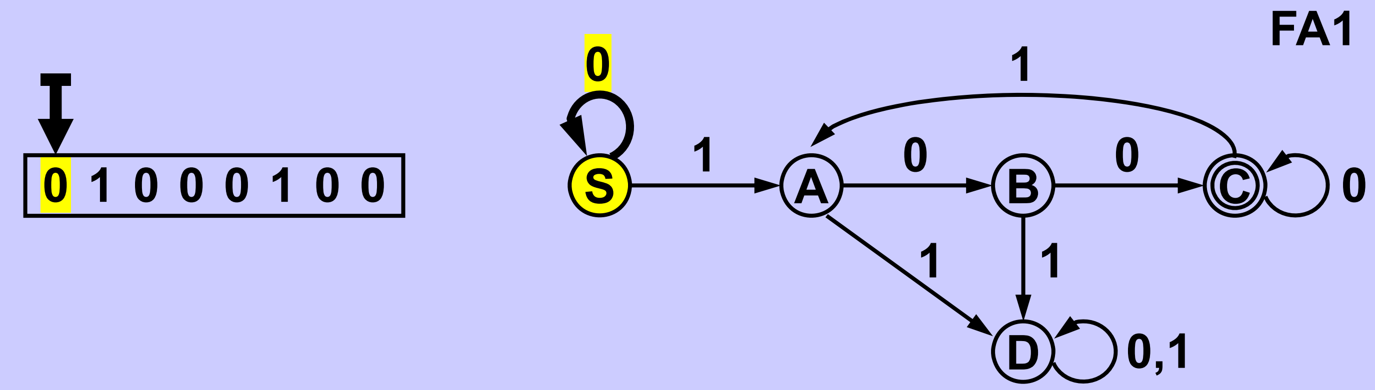
A … <u>alphabet</u> … {0,1}, $|A| = 2$
Q ... set of <u>states</u> {S, A, B, C, D }
σ ... <u>transition function</u> ... $\sigma: Q \times A \rightarrow Q$: {
$\sigma(S,0) = S$, $\sigma(A,0) = B$, $\sigma(B,0) = C$, $\sigma(C,0) = C$, $\sigma(D,0) = D$,
$\sigma(S,1) = A$, $\sigma(A,1) = D$, $\sigma(B,1) = D$, $\sigma(C,1) = A$, $\sigma(D,1) = D$ }
S_0 ... <u>start state</u> $S \in Q$
Q_F … unempty set of <u>final states</u> $\emptyset \neq \{ C \} \subseteq Q$

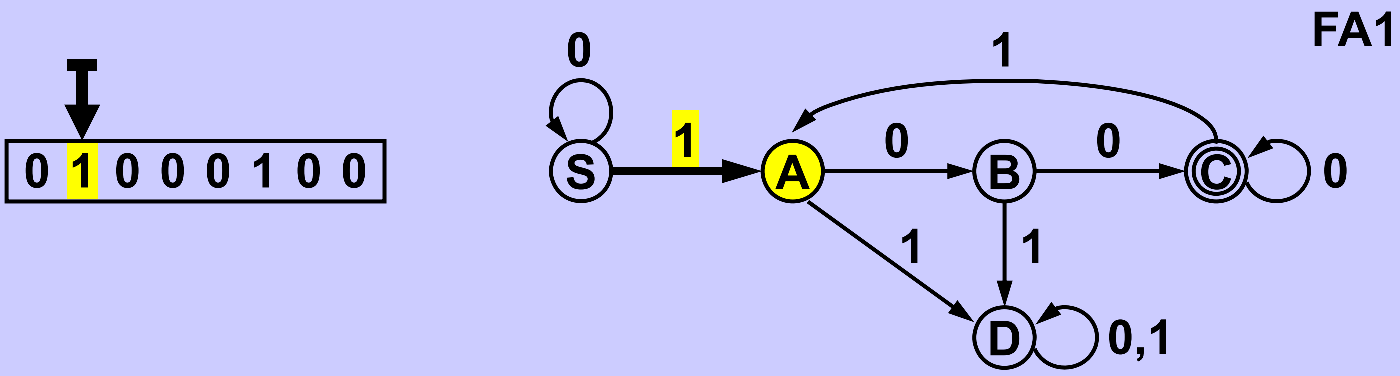
Transition diagram
of the automaton FA1

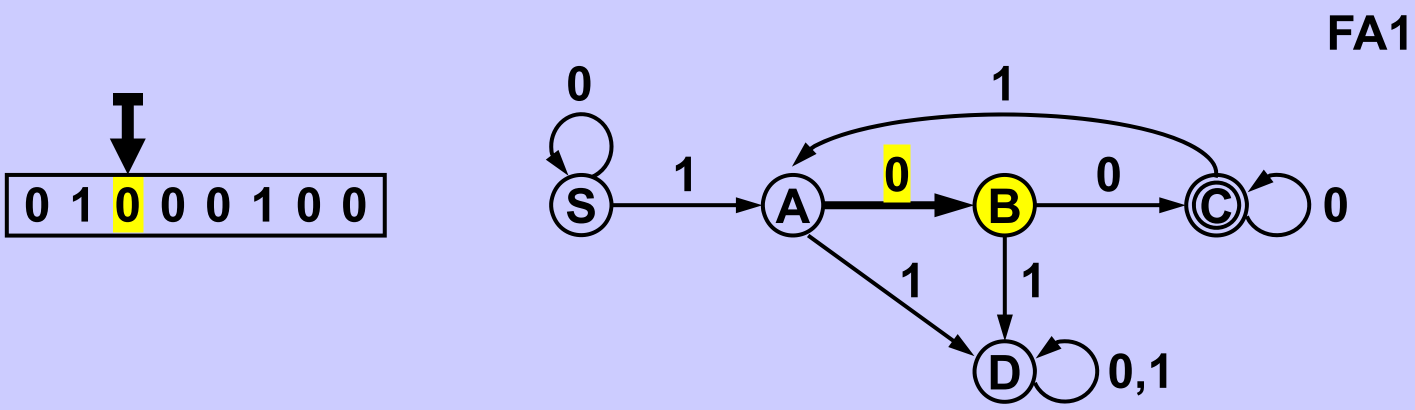


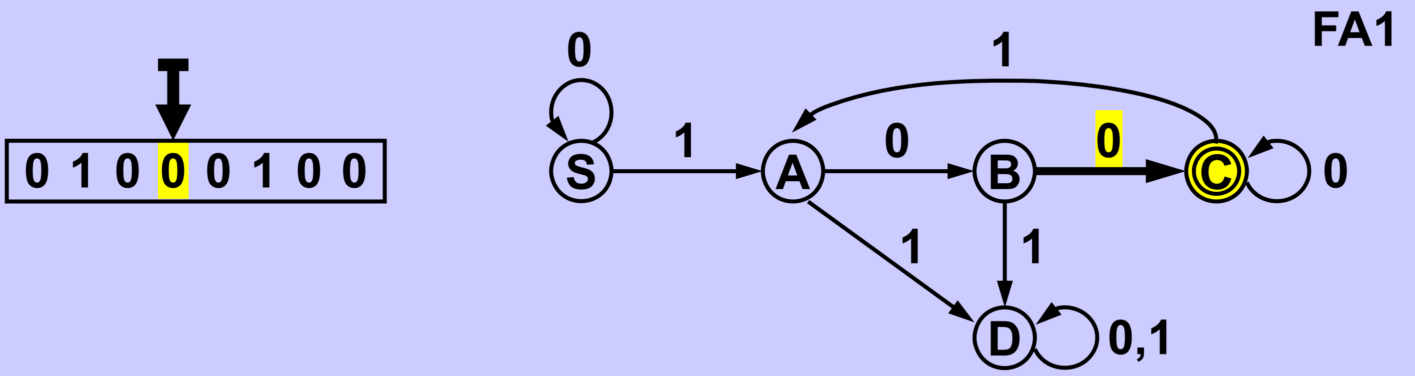
FA1

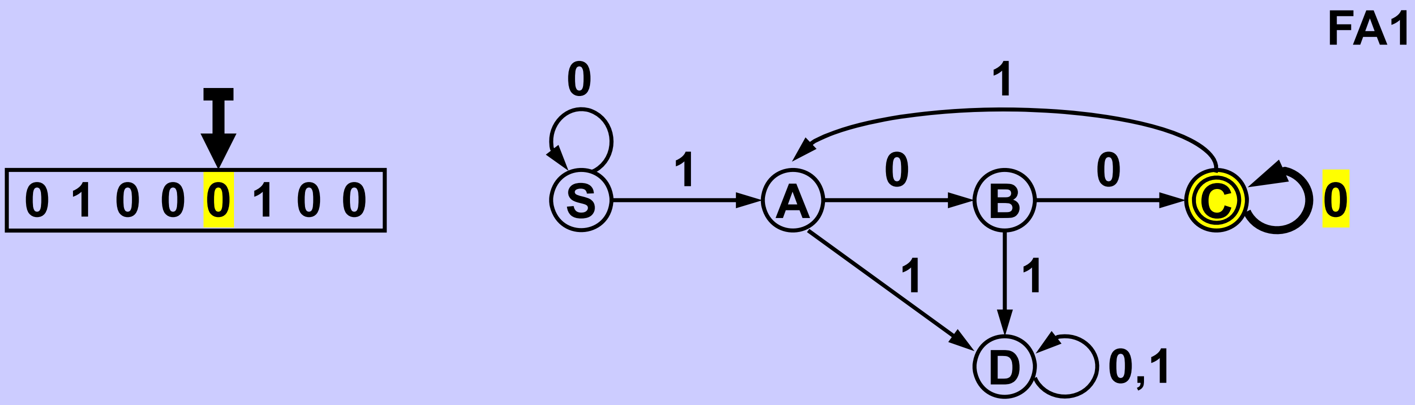

FA1
0 1 0 0 0 1 0 0
S
A
B
C
D
0
1
0
0
1
1
1
0
0,1

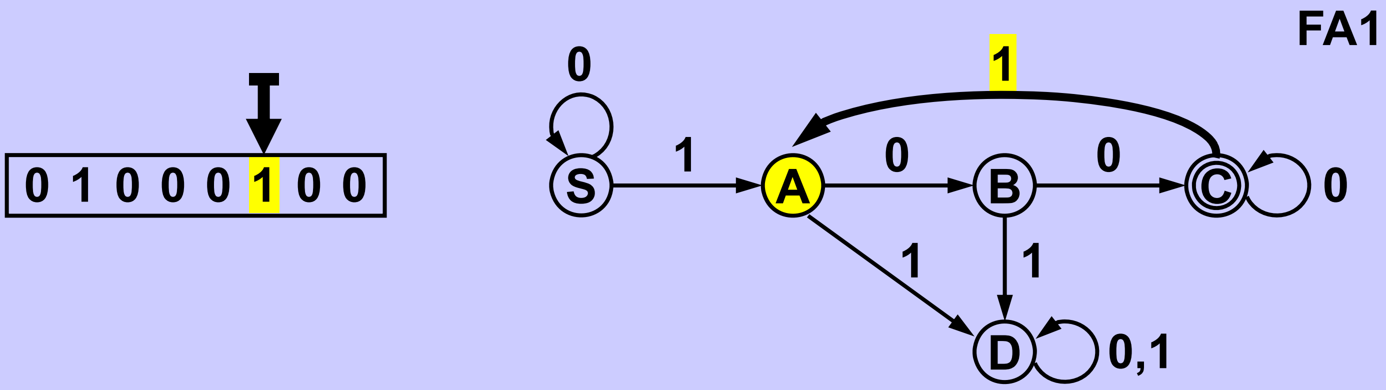

FA1
0 1 0 0 0 1 0 0
S
A
B
C
D
0
1
0
0
1
1
1
0
0,1

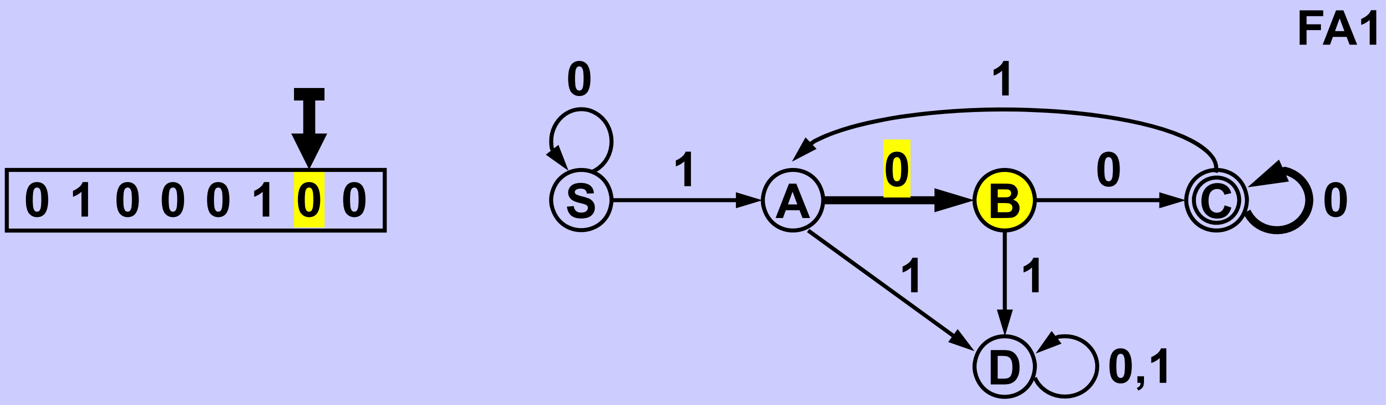

0 1 0 0 0 1 0 0
FA1
0
1
1
0
0
1
1
1
0
0,1
S
A
B
C
D

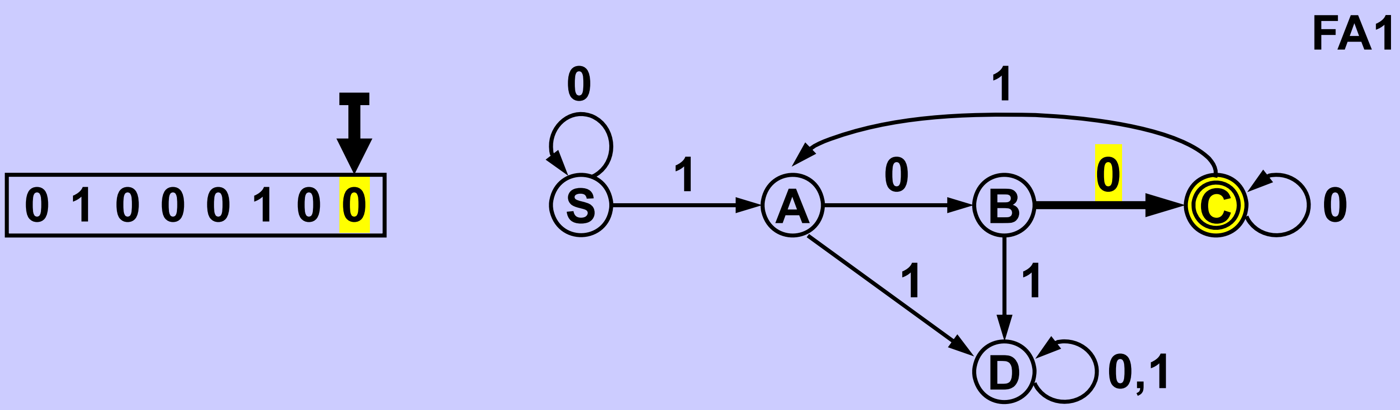

0 1 0 0 0 1 0 0
FA1
0
1
1
0
0
1
1
1
0
0,1
S
A
B
C
D


FA1
0 1 0 0 0 1 0 0
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


FA1
0 1 0 0 0 1 0 0
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


FA1
0 1 0 0 0 1 0 0
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D

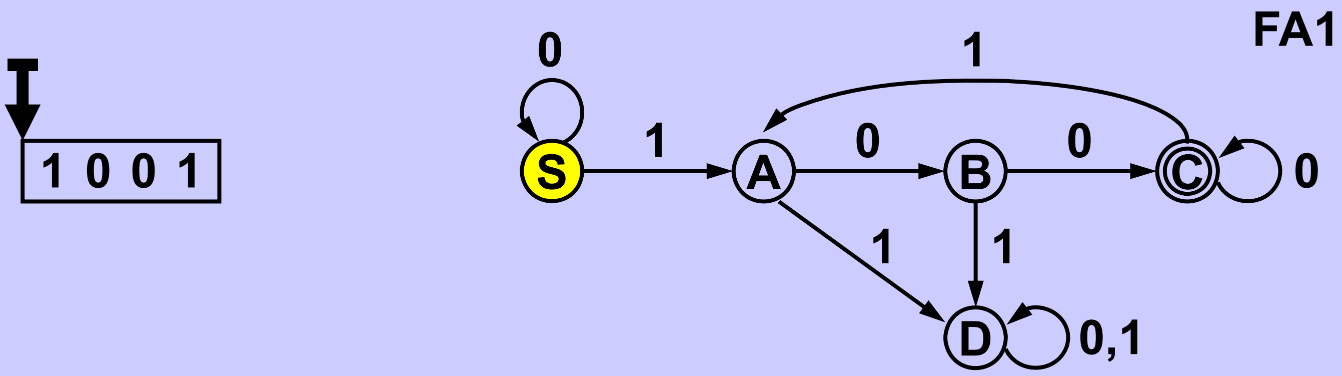

FA1
0 1 0 0 0 1 0 0
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D

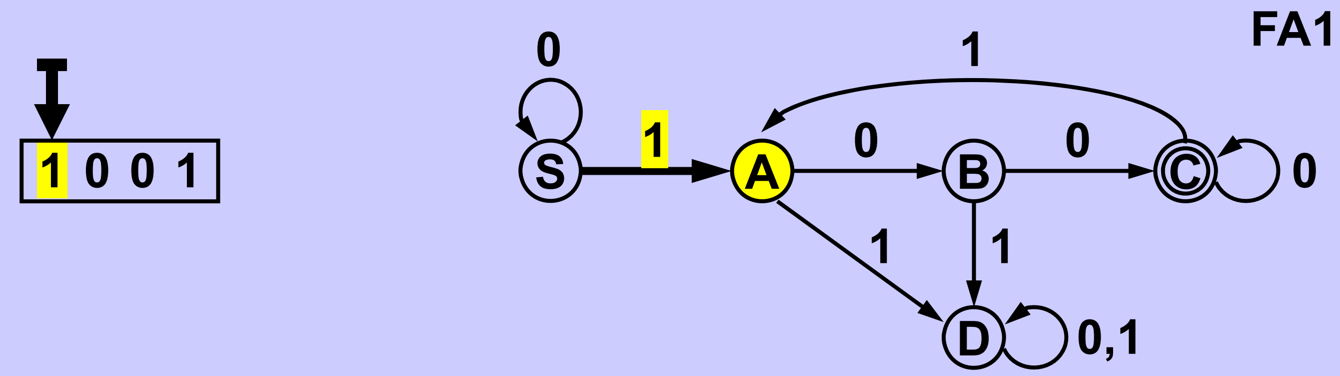


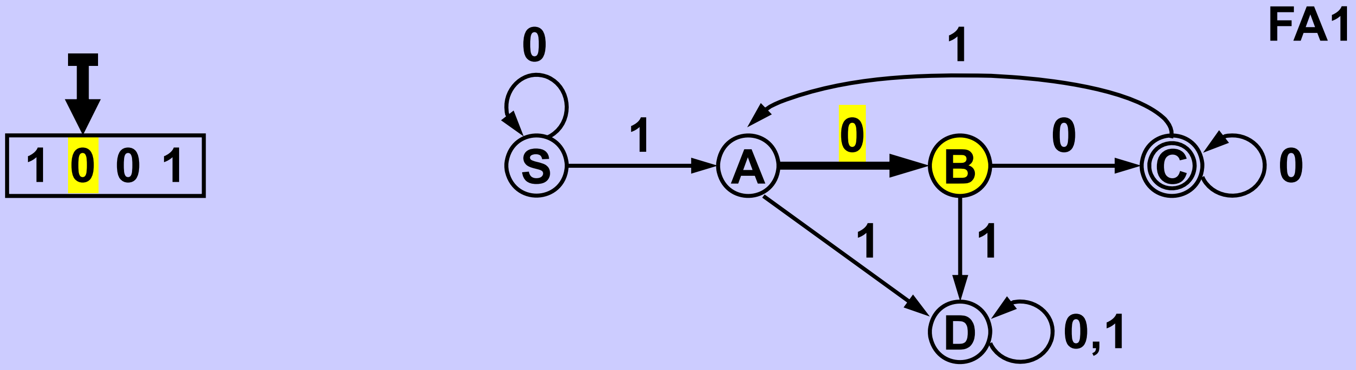
When the last word symbol is read automaton FA1 is in final state ◎

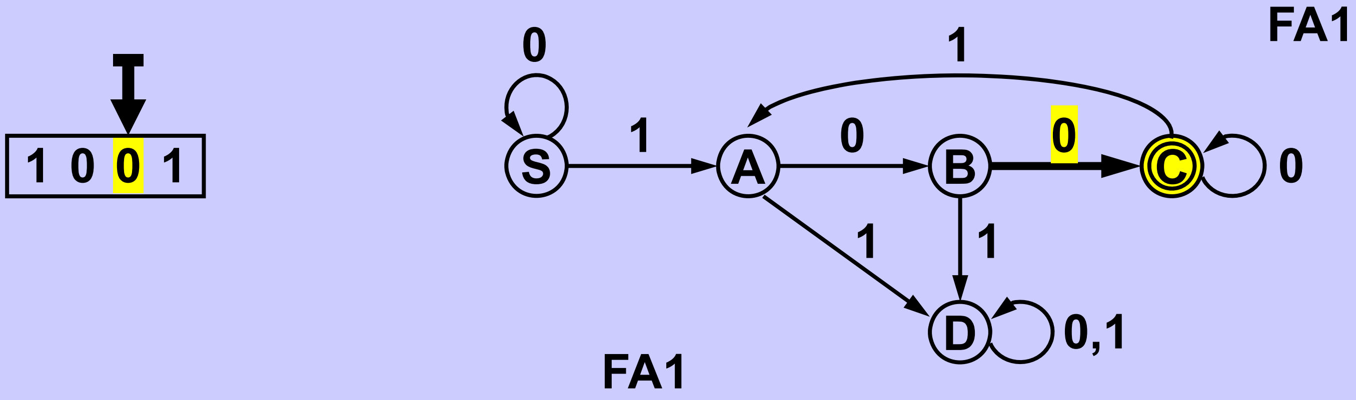
$\Rightarrow$

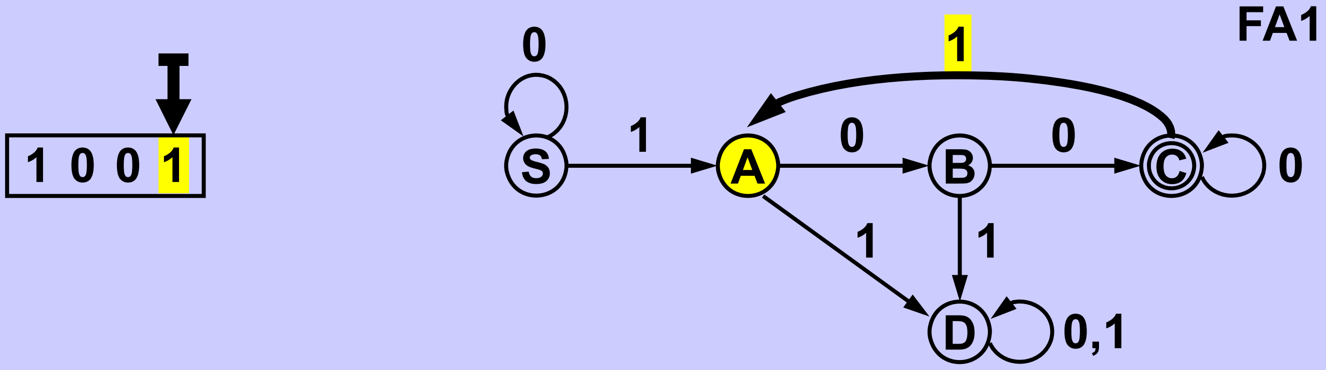
Word 0 1 0 0 0 1 0 0 **is** <u>**accepted**</u> **by automaton FA1**


1 0 0 1
FA1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


1 0 0 1
FA1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


FA1
1 0 0 1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D

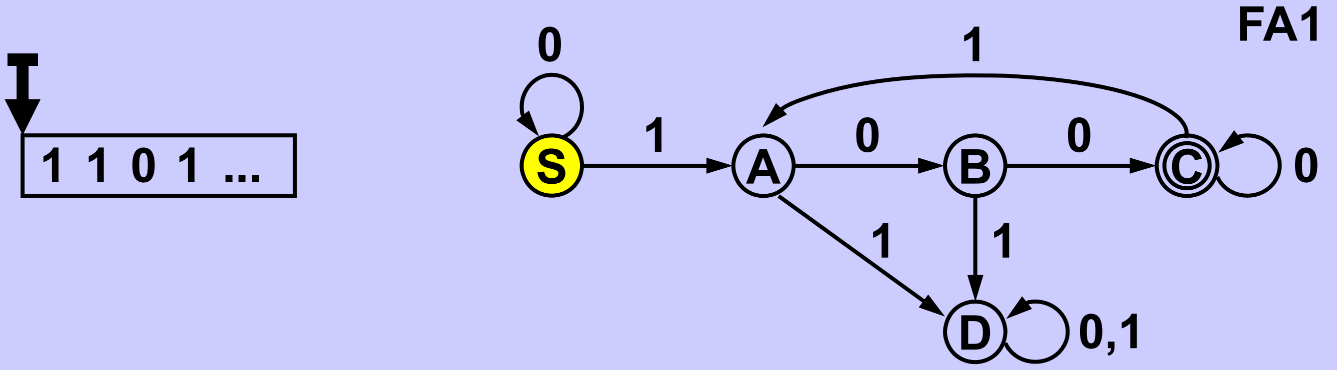

FA1
1 0 0 1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D
FA1

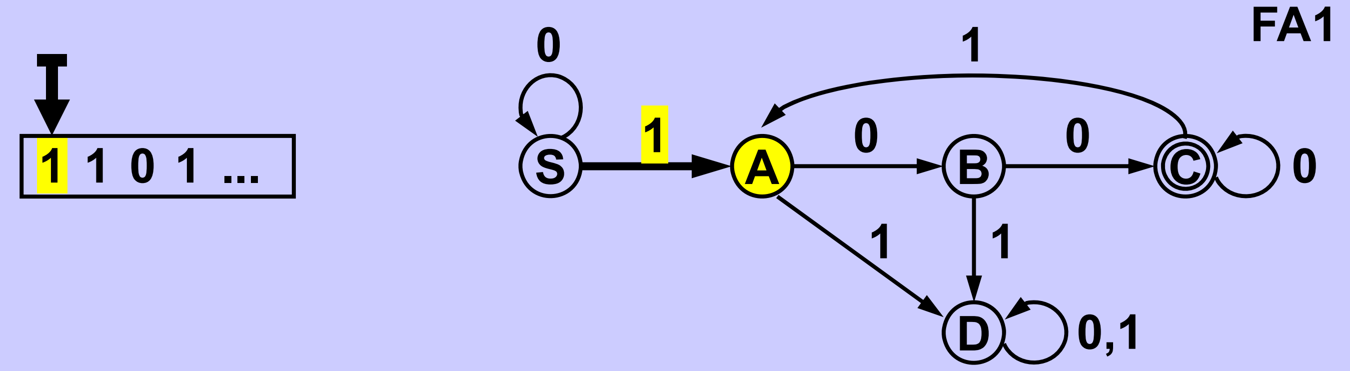


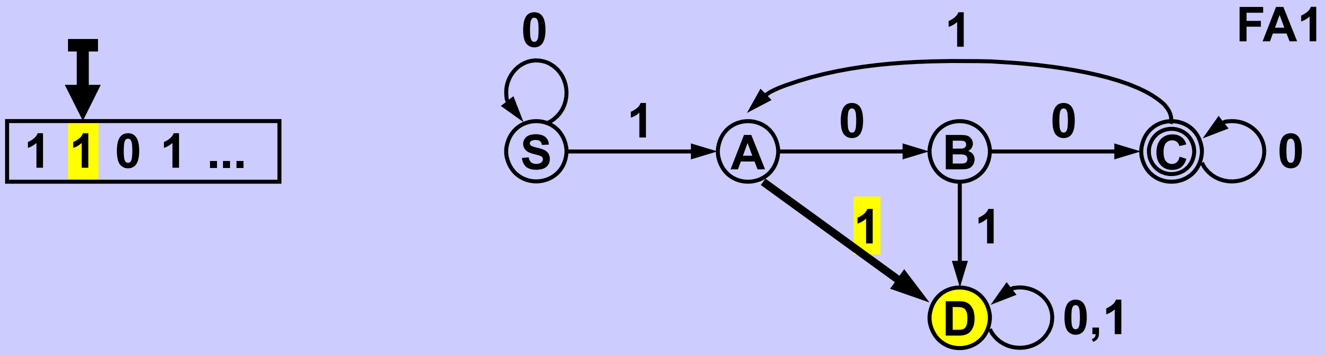
When the last word symbol is read automaton FA1 is in a state which is not final ○

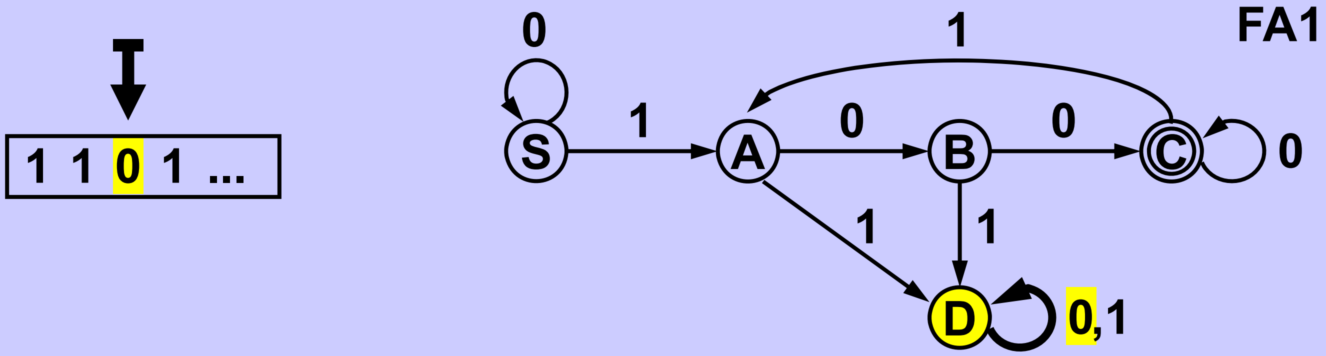
⇒

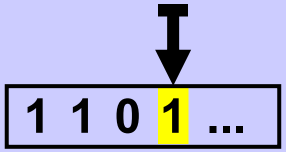
Word 1 0 0 1 is not accepted by automaton FA1

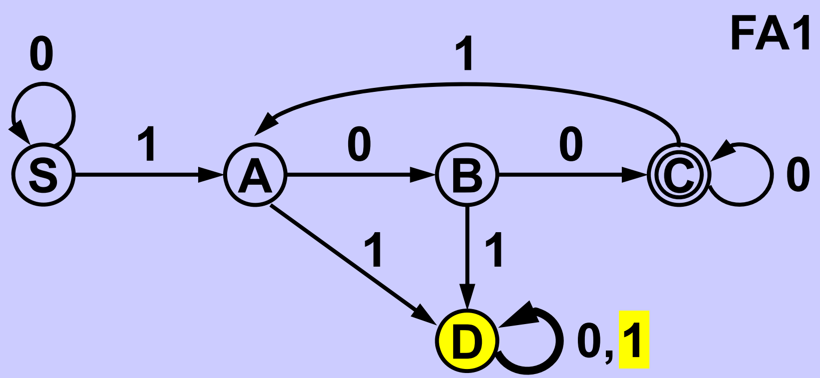

1 1 0 1 ...
FA1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


1 1 0 1 ...
FA1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


1 1 0 1 ...
FA1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D


1 1 0 1 ...
FA1
0
1
1
0
0
0
1
1
0,1
S
A
B
C
D





No word starting with 1 1 ... is accepted by automaton FA1

No word containing ... 1 1 ... is accepted by automaton FA1

No word containing ... 1 0 1 ... is accepted by automaton FA1

Automaton FA1 accepts only words -- containing at least one 1
-- containing at least two 0s after each 1

Language accepted by automaton = set of all words accepted by automaton

Automaton activity:

At the begining the automaton is in the start state.

Next it reads the input word symbol by symbol and transits

to other states according to the transition function.

When the word is read the automaton is again in some state.

If it is in a final state, we say that it <u>accepts</u> the word,

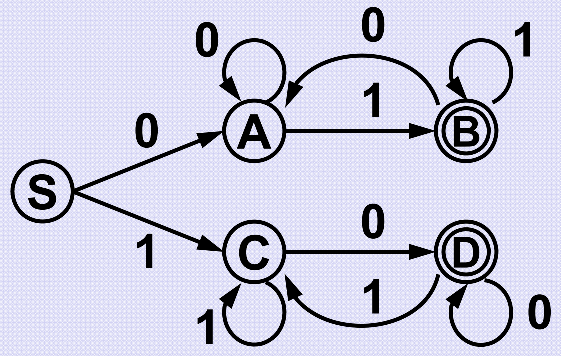
if it is not in a final state, we say that it <u>does not accept</u> the word.

All words accepted by the automaton represent

<u>a language accepted (or recognized) by the automaton.</u>

Language over alphabet {0,1} :

If the word starts with 0, it ends with 1,
If the word starts with 1, it ends with 0.



Automat FA2

Example of analysis of different words by FA2:

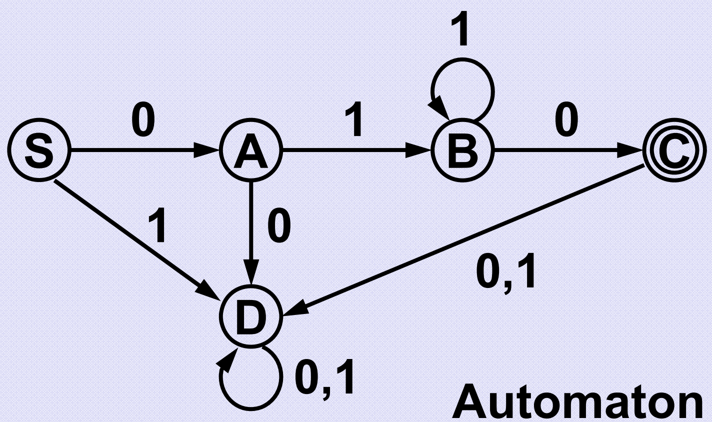
0 1 0 1 0 : (S),0 → (A),1 → (B),0 → (A),1 → (B),0 → (A)

(A) is not a final state, word 0 1 0 1 0 is rejected by FA2.

1 0 1 1 0 : (S),1 → (C),0 → (D),1 → (C),1 → (C),0 → (D)

(D) is a final state, word 1 0 1 1 0 is accepted by FA2.

Language:
{
0 1 0,
0 1 1 0,
0 1 1 1 0,
0 1 1 1 1 0,
0 1 1 1 1 1 0,
... }



Automaton FA3

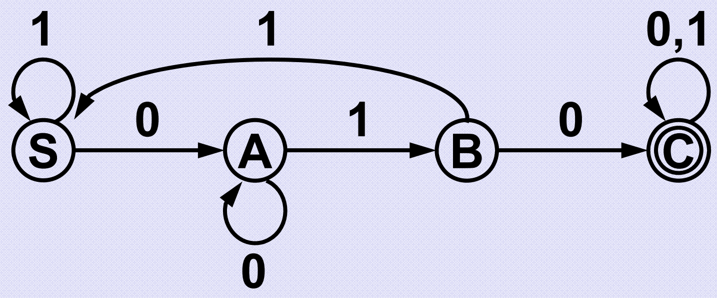
Example of analysis of different words by FA3:

0 1 0 1 0 : (S),0 → (A),1 → (B),0 → (C),1 → (D),0 → (D)

(D) is not a final state, word 0 1 0 1 0 is rejected by FA3.

0 1 1 1 0 : (S),0 → (A),1 → (B),1 → (B),1 → (B),0 → (C)

(C) is a final state, word 0 1 1 1 0 is accepted by FA3.



Automat FA4

Automaton FA4 accepts each word over the alphabet {0,1} which contains subsequence $\boxed{\text{... 0 1 0 ...}}$

Example of analysis of different words by FA4:

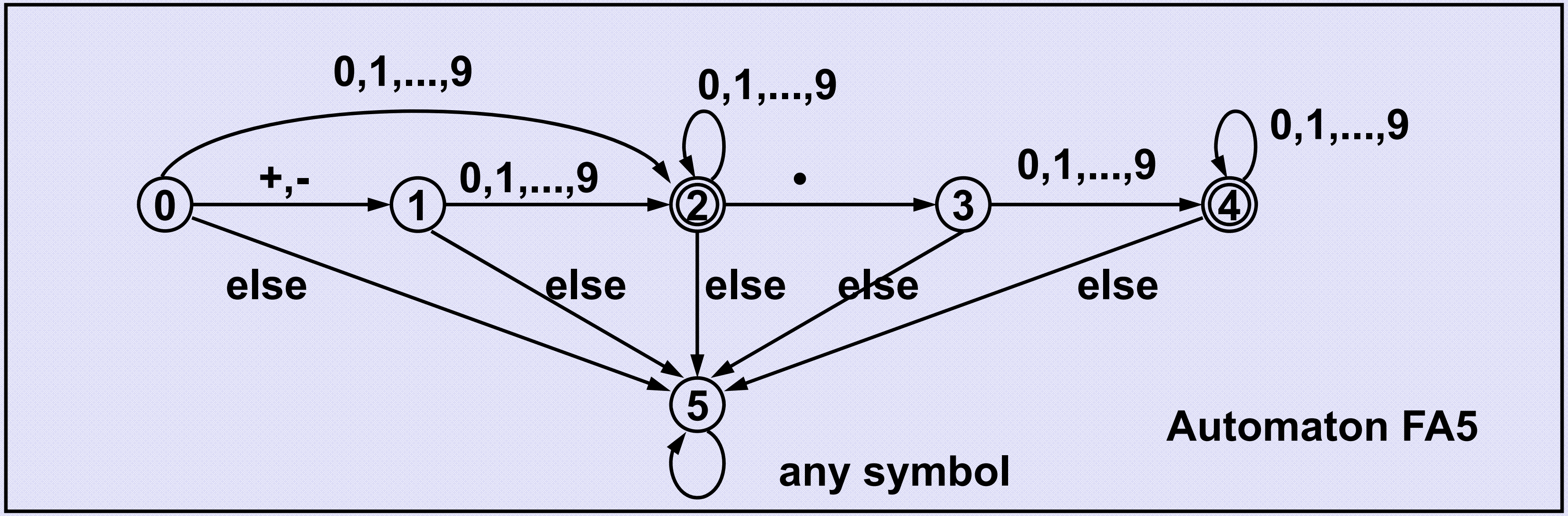
0 0 1 0 1 : (S),0 → (A),0 → (A),1 → (B),0 → (C),1 → (C)

(C) is a final state, word 0 0 1 0 1 is accepted by FA4.

0 1 1 1 0 : (S),0 → (A),1 → (B),1 → (S),1 → (S),0 → (A)

(A) is not a final state, word 0 1 1 1 0 is rejected by FA4.

Language over the alphabet { +, - , . , 0, 1, …, 8, 9, … } whose words represent a decimal numbers



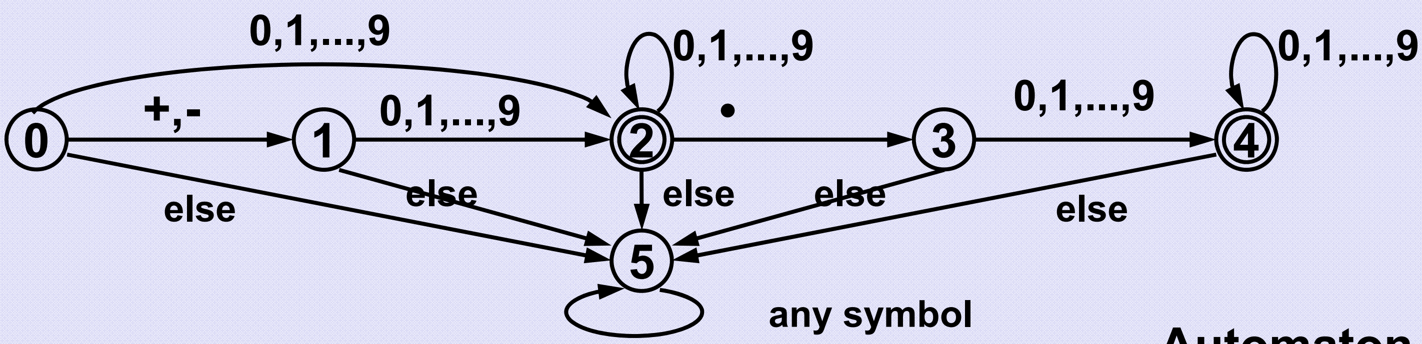
Example of word analysis

+87.09: (0),+ → (1),8 → (2),7 → (2), . → (3),0 → (4),9 → (4)

(4) is a final state, word +87.05 is accepted by FA5.

76+2: (0),7 → (2),6 → (2),+ → (5),2 → (5)

(5) is not a final state, word 76+2 is not accepted by FA5.



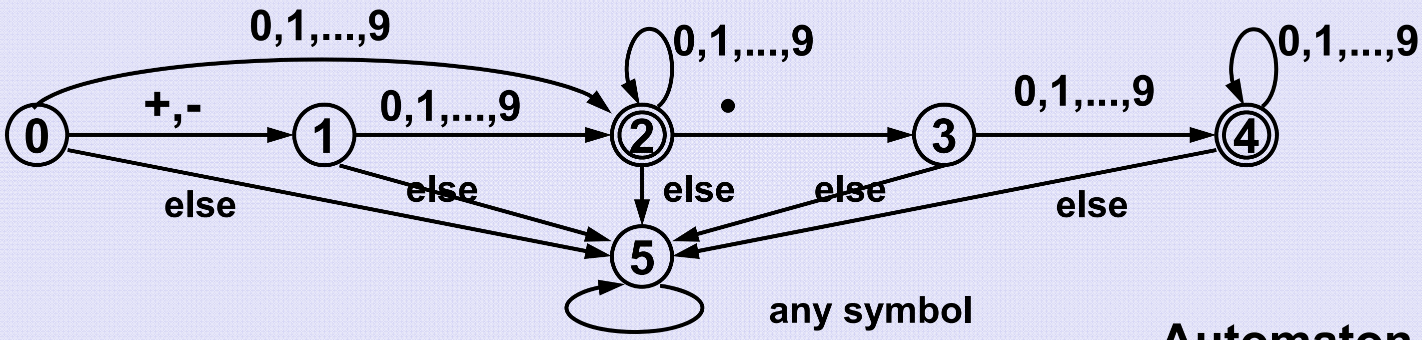
Automaton FA5

Code of the finite automaton

(The word which is being read is stored in the array text[ ]):

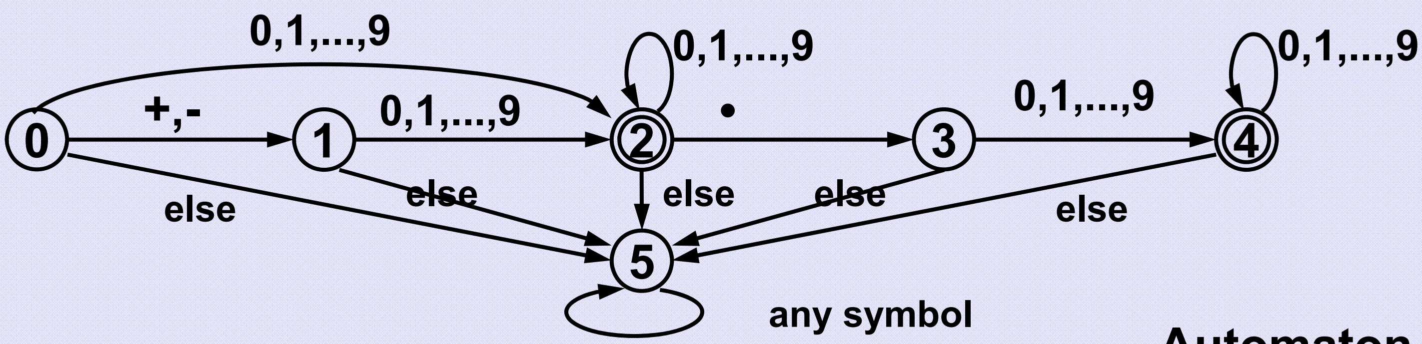
```
boolean isDecimal( char text[] ) {
int state = 0;

for(int i = 0; i < text.length; i++) {  // check each symbol
  switch (state) {
  ...
```



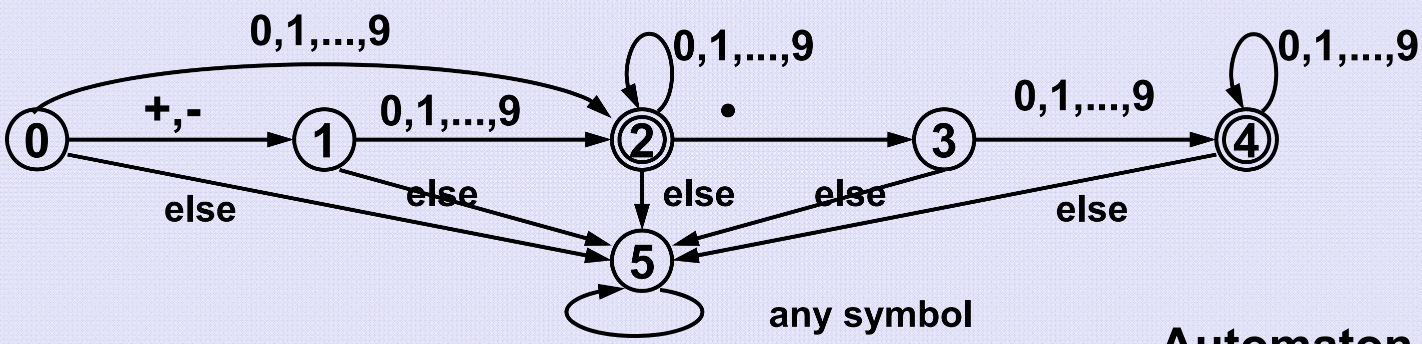
Automaton FA5

```
0 case 0:
    if ((text[i] == '+') || (text[i] == '-')) state = 1;
    else
    if ((text[i] >= '0') && (text[i] <= '9')) state = 2;
    else state = 5;
    break;
1 case 1:
    if ((text[i] >= '0') && (text[i] <= '9')) state = 2;
    else state = 5;
    break;
```



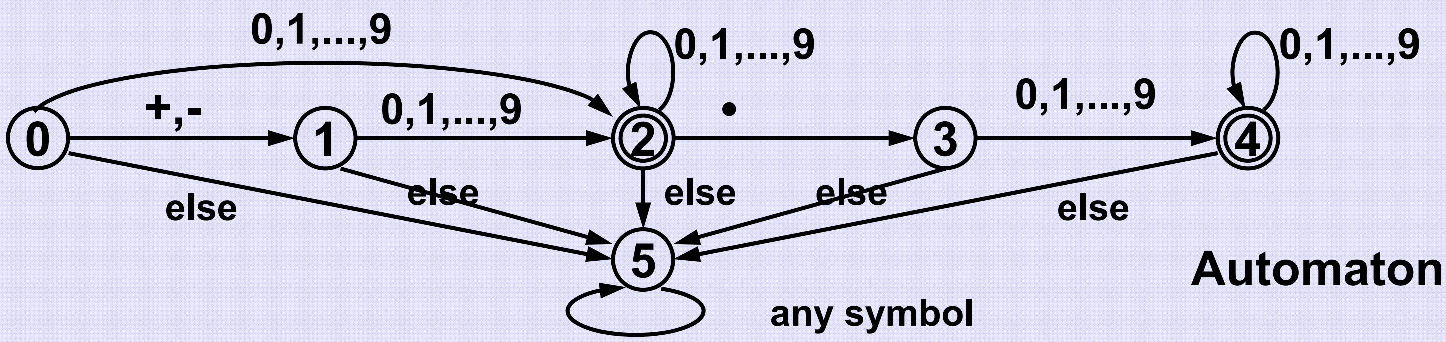
Automaton FA5

```
(2) case 2:
      if ((text[i] >= '0') && (text[i] <= '9')) state = 2;
      else
      if (text[i] == '.') state = 3;
      else state = 5;
      break;
(3) case 3:
      if ((text[i] >= '0') && (text[i] <= '9')) state = 4;
      else state = 5;
      break;
```



Automaton FA5

```
④  case 4:
     if ((text[i] >= '0') && (text[i] <= '9')) state = 4;
     else state = 5;
     break;
⑤  case 5: break;                    // no need to react anyhow
    default : break;
    } // end  switch
  } // end for
return (state == 2) || (state == 4);   // final states
}
```



Automaton FA5

Transition table of automaton FA5

alphabet

states	0	1	2	3	4	5	6	7	8	9	.	+	–	%	=	...	}	final
0	2	2	2	2	2	2	2	2	2	2	5	1	1	5	5	...	5	0
1	2	2	2	2	2	2	2	2	2	2	5	5	5	5	5	...	5	0
2	2	2	2	2	2	2	2	2	2	2	3	5	5	5	5	...	5	1
3	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	...	5	0
4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	...	5	1
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	...	5	0

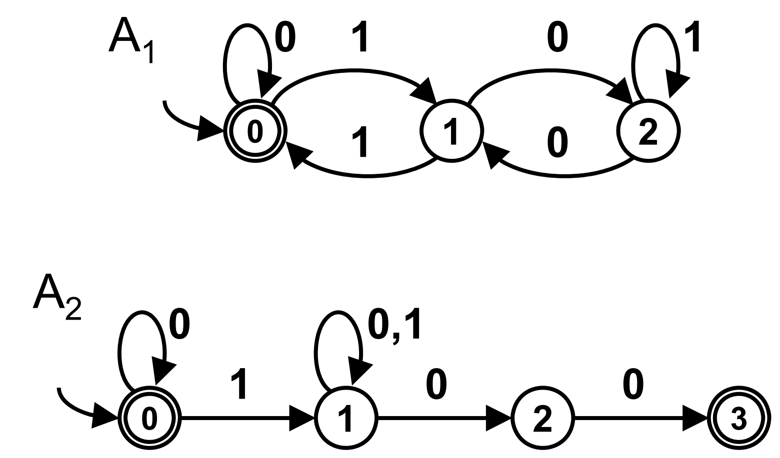
```
boolean isAccepted( char [] text, int [][] TT, boolean [] F ){
  int state = 0;   // start state
  for( char c: text ){
    state = TT[state][Integer.valueOf(c)];
  }
  return F[state];
}
```

Tables TT and F specify the automaton completely (provided start state is typically 0) , their construction is problem/implementation dependent and should not influence the operation(s) of the automaton.

alphabet	0	1	2	3	4	5	6	7	8	9	.	+	–	%	=	...	}	final
states 0	2	2	2	2	2	2	2	2	2	2	5	1	1	5	5	...	5	0
1	2	2	2	2	2	2	2	2	2	2	5	5	5	5	5	...	5	0
2	2	2	2	2	2	2	2	2	2	2	3	5	5	5	5	...	5	1
3	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	...	5	0
4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	...	5	1
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	...	5	0

Deterministic Finite Automaton (DFA)
Nondeterministic Finite Automaton (NFA)

Both DFA nd NFA consist of:
  Finite input alphabet Σ.
  Finite set of internal states Q.
  One starting state $q_0 \in Q$.
  Nonempty set of accept states $F \subseteq Q$.
  Transition function δ.

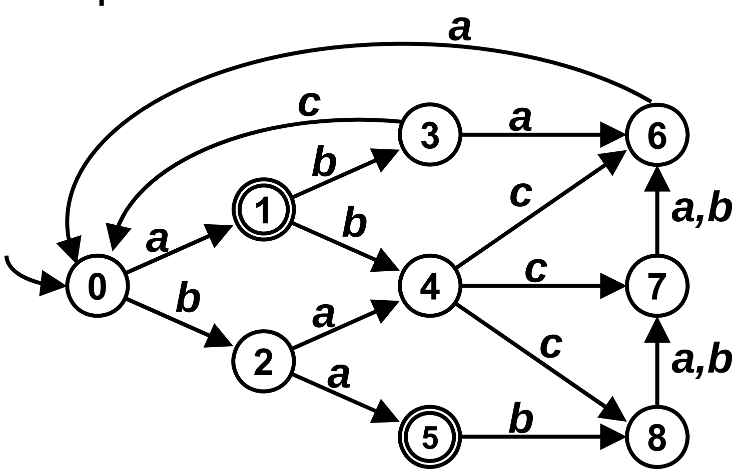


DFA transition function is $\delta: Q \times \Sigma \rightarrow Q$.
DFA is always in one of its states $q \in Q$.
DFA transits from current state to another state depending on the current input symbol.

NFA transition function is $\delta: \ Q \times \Sigma \rightarrow P(Q)$ ($P(Q)$ is the powerset of Q)
NFA is always (simultaneously) in a set of some number of its states.
NFA transits from a set of states to another set of states
 depending on the current input symbol.

Nondeterministic finite automaton (NFA) A1,
its transition diagram and its transition table

A_1

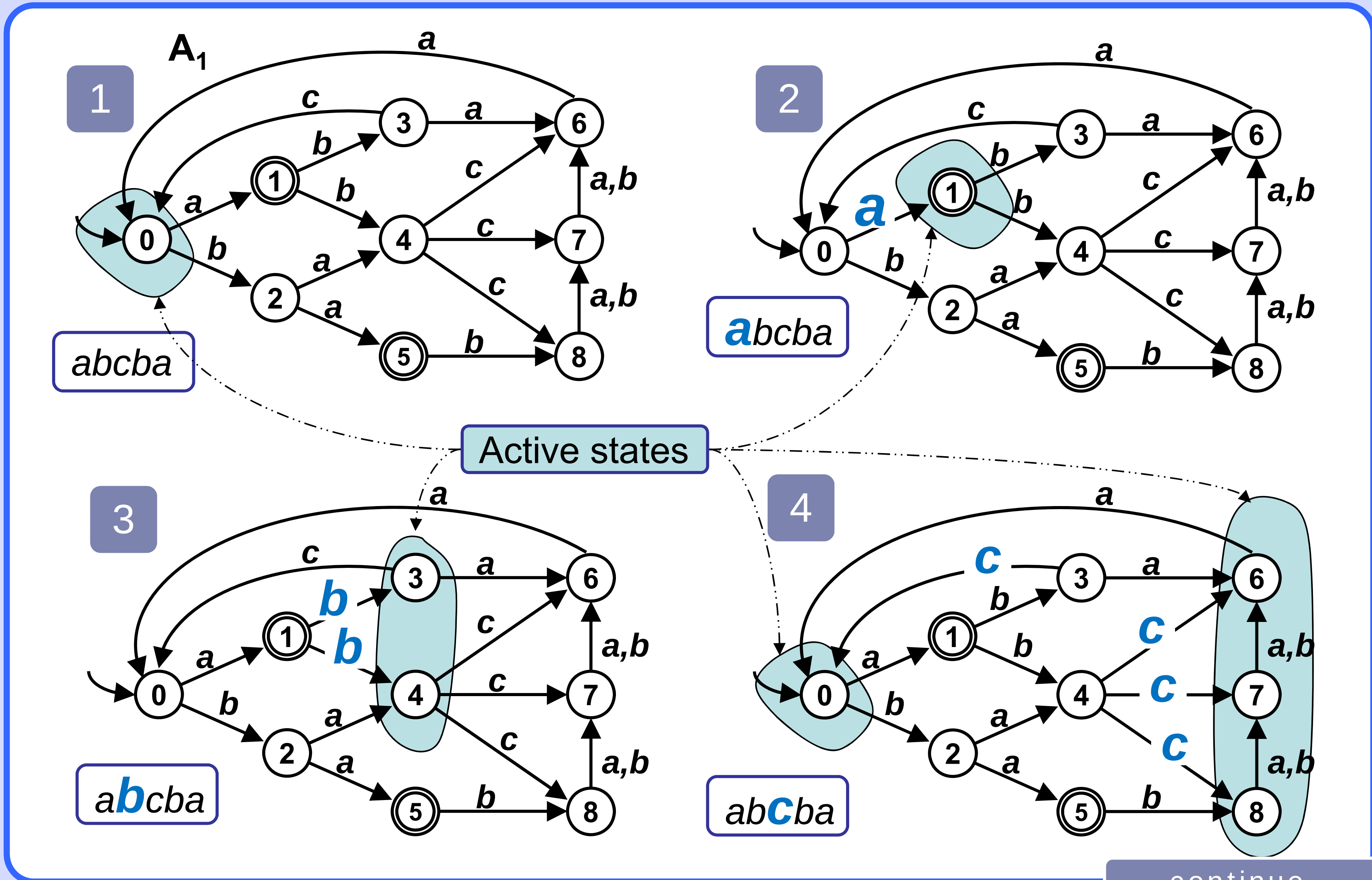


alphabet

states	a	b	c	
0	1	2		
1		3,4		F
2	4,5			
3	6		0	
4			6,7,8	
5		8		F
6	0			
7	6	6		
8	7	7		

accept states marked

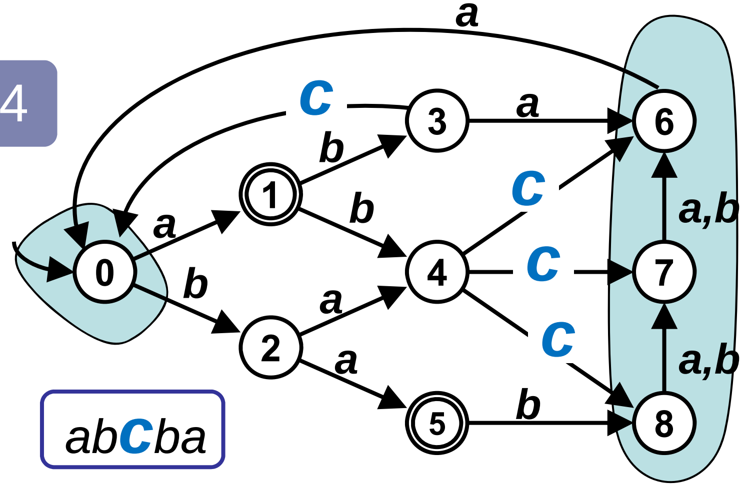
NFA **A**$_1$ processing input word *abcba*



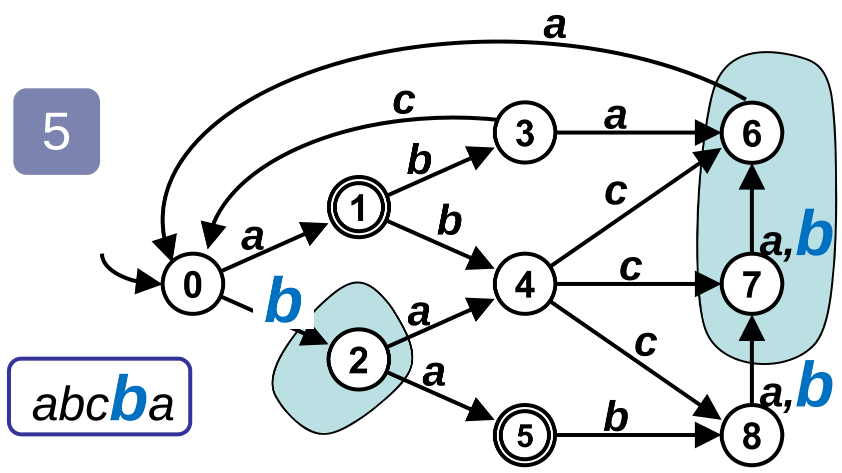
continue...

...continued

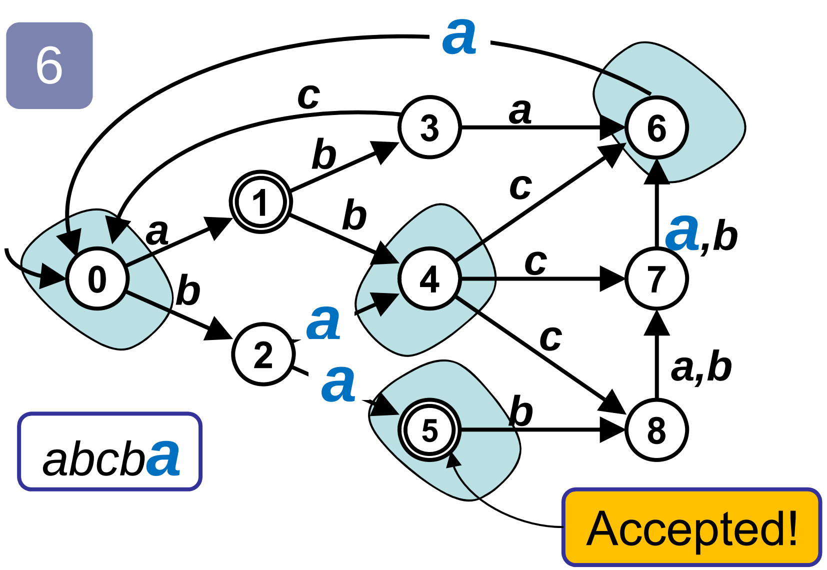
4



5



6



NFA $\mathbf{A_1}$ has processed the word *abcba* and went through the input characters and respective sets(!) of states

{0} → a → {1} → b → {3, 4} → c →
→ {0, 6, 7, 8} → b → {2, 6, 7} → a →
→ {0, 4, 5, 6}.

NFA simulation without transform to DFA

Idea:
Register all states to which you have just arrived. In the next step, read the input symbol ***x*** and move SIMULTANEOUSLY to ALL states to which you can get from ALL active states along transitions marked by ***x***.

Input: NFA , text in array **t**

```
SetOfStates S = {q0}, S_tmp;

i = 1;
while( (i <= t.length) && (!S.isEmpty()) ) {
  S_tmp = Set.emptySet();
  for( q in S )                  // for each state in S
    S_tmp.union( delta(q, t[i]) );
  S = S_tmp;
  i++;
}
return S.containsFinalState();   // true or false
```