RAJIV GANDHI INSTITUTE OF PETROLEUM TECHNOLOGY, JAIS, AMETHI

Department of Computer Science and Engineering

Compiler Design (CS312)

By

Dr. Kalka Dubey

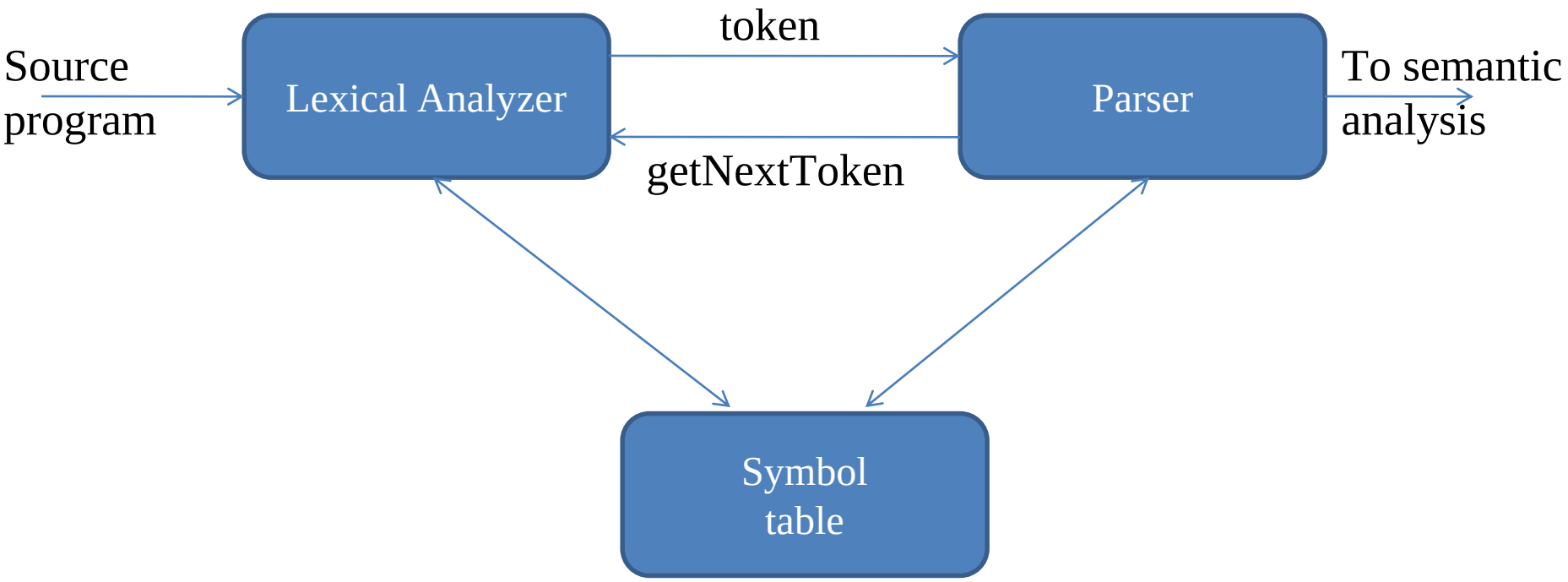
Lexical analyzer

Interaction of scanner & parser

- Upon receiving a “Get next token” command from parser, the lexical analyzer reads the input character until it can identify the next token.

- Lexical analyzer also stripping out comments and white space in the form of blanks, tabs, and newline characters from the source program.

The role of lexical analyzer



Why to separate Lexical analysis and parsing

1. Simplicity of design
2. Improving compiler efficiency
3. Enhancing compiler portability

Tokens

- Sequence of character having a collective meaning is known as token.
- Categories of Tokens:

✓ Identifier
✓ Keyword
✓ Operator
✓ Special symbol
✓ Constant

Pattern

- The set of rules called pattern associated with a token.
- Example: “non-empty sequence of digits”, “letter followed by letters and digits”

Lexemes

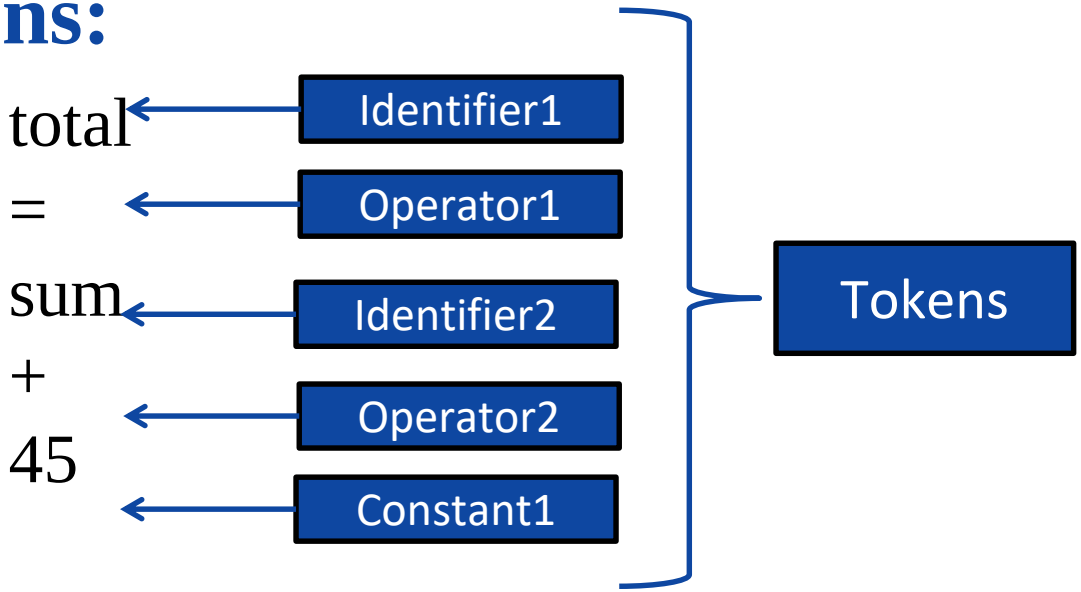
- The sequence of character in a source program matched with a pattern for a token is called lexeme.

- Example: Rate, DIET, count, Flag

Example: Token, Pattern & Lexemes

Example: total = sum + 45

Tokens:



Lexemes

Lexemes of identifier: total, sum

Lexemes of operator: =, +

Lexemes of constant: 45

Attributes for tokens

E = M * C ** 2

<id, pointer to symbol table entry for E>

<assign-op>

<id, pointer to symbol table entry for M>

<mult-op>

<id, pointer to symbol table entry for C>

<exp-op>

<number, integer value 2>

Exercise

Identify the number of tokens in the following C++ code.

```
int x = a+++b;
```

Solution

Tokens:
int (keyword)
x (identifier)
= (operator)
a (identifier)
++ (operator)
+ (operator)
b (identifier)
; (delimiter)

Total tokens = 8

Exercise

Identify the number of tokens in the following C++ code.

```
float y = (x>=10)?x--:++x;
```

Solution

Total tokens = 15

Exercise

Identify the number of tokens in the following C++ code.

```
cout<<"Token\tCount"<<endl;
```

Solution

Total tokens = 6

Exercise

Identify the number of tokens in the following C++ code.

```
x = y = z += 10;;
```

Solution

Total tokens = 8

Exercise

Identify the number of tokens in the following C++ code.

```
#define MAX(a,b) ((a)>(b)?(a):(b))
```

Solution

Total tokens = 25

Input buffering

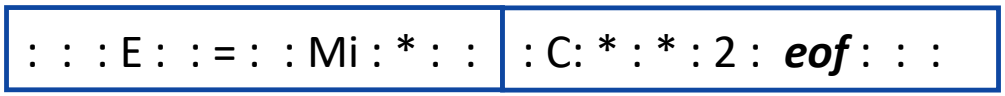
- There are mainly two techniques for input buffering:

✓ Buffer pairs

✓ Sentinels

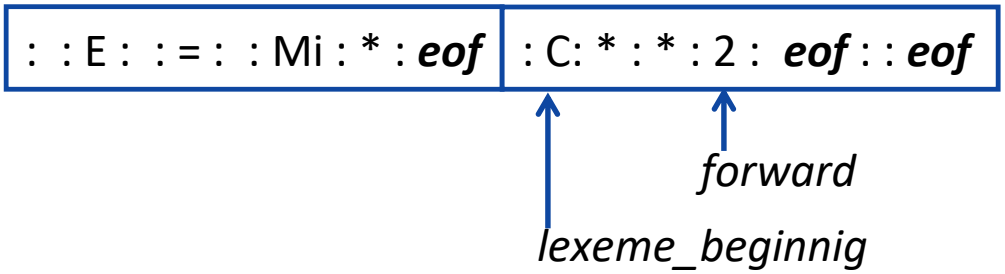
Buffer Pair

- The lexical analysis scans the input string from left to right one character at a time.
- Buffer divided into two N-character halves, where N is the number of character on one disk block.



Sentinels

- In buffer pairs we must check, each time we move the forward pointer that we have not moved off one of the buffers.
- Thus, for each character read, we make two tests.
- We can combine the buffer-end test with the test for the current character.
- We can reduce the two tests to one if we extend each buffer to hold a sentinel character at the end.
- The sentinel is a special character that cannot be part of the source program, and a natural choice is the character EOF.



Sentinels

```
forward  := forward + 1;
    if forward  = eof then begin
            if forward at end of first half then begin
                    reload second half;
                    forward  := forward + 1;
            end
            else if forward at the second half then begin
                    reload first half;
                    move forward to beginning of first half;
            end
            else terminate lexical analysis;
    end
```

Lexical errors

- Lexical errors are the first type of errors detected by a compiler, occurring during the lexical analysis phase.

- A lexical error occurs when the compiler cannot recognize a sequence of characters as a valid token in the programming language.

- Such errors are recognized when no pattern for tokens matches a character sequence.

Common Causes of Lexical Errors

- Invalid or illegal characters
- Malformed identifiers
- Unterminated string or character literals
- Invalid numeric constants
- Misspelled keywords

Error recovery

- **Panic mode:** successive characters are ignored until we reach to a well formed token.
- Delete one character from the remaining input.
- Insert a missing character into the remaining input.
- Replace a character with another character.
- Transpose two adjacent characters

Strings and languages

Term	Definition
Prefix of s	A string obtained by removing **zero or more trailing symbol** of string S. e.g., ban is prefix of banana.
Suffix of S	A string obtained by removing **zero or more leading symbol** of string S. e.g., nana is suffix of banana.
Sub string of S	A string obtained by **removing prefix and suffix** from S. e.g., nan is substring of banana
Proper prefix, suffix and substring of S	Any nonempty string t that is respectively proper prefix, suffix or substring of S, such that **s≠t**.
Subsequence of S	A string obtained by removing **zero or more not necessarily contiguous symbol** from S. e.g., baaa is subsequence of banana.

Operations on languages

Operation	Definition
Union of L and M Written L U M	$L\ U\ M\ = \{s \mid s\ is\ in\ L\ or\ s\ is\ in\ M\}$
Concatenation of L and M *Written LM*	$LM\ = \{st \mid s\ is\ in\ L\ and\ t\ is\ in\ M\}$
Kleene closure of L Written L^*	L^* *denotes "zero or more concatenation of"* L.
Positive closure of L *Written* L^+	L^+ *denotes "one or more concatenation of"* L.

Regular expression

- A regular expression is a sequence of characters that define a pattern.

Notational shorthand's

- ✓ One or more instances: +
- ✓ Zero or more instances: *
- ✓ Alphabets: Σ

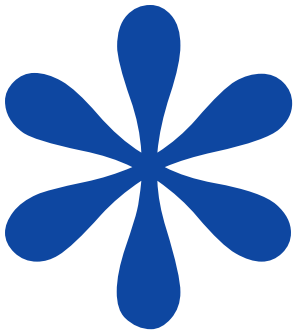
Regular expression

- $\in$ is a regular expression that denotes $\{\in\}$, the set containing empty string.
- If a is a symbol in Σ then a is a regular expression, $L(a)=\{a\}$
- Suppose r and s are regular expression denoting the languages $L(r)$ and $L(s)$. Then,
- (r)|(s) is a regular expression denoting L(r) U L(s)
- (r)(s) is a regular expression denoting L(r)L(s)
- (r)* is a regular expression denoting $(L(r))*$
- (r) is a regular expression denoting L((r))

- The language denoted by regular expression is said to be a regular set.

Regular expression

L = Zero or More Occurrences of a = a*



ϵ

a

aa

aaa

aaaa

aaaaa

.....

Infinite

.....

Regular expression

L = One or More Occurrences of a = a^+

a

aa

aaa

aaaa

aaaaa

…..

Infinite

…..

Precedence and associativity of operators

Operator	Precedence	Associative
Kleene *	1	left
Concatenation	2	left
Union |	3	left

Regular expression examples

1. 0 or 1

   Strings: 0, 1

   R. E. = 0 | 1

2. 0 or 11 or 111

   Strings: 0, 11, 111

   R. E. = 0 | 11 | 111

3. String having zero or more *a*.

   Strings: ϵ, a, aa, aaa, aaaa ……

   R. E. = a^*

4. String having one or more *a*.

   Strings: a, aa, aaa, aaaa ……

   R. E. = a^+

5. Regular expression over $\Sigma = \{a, b, c\}$ that represent all string of length 3.

   Strings: abc, bca, bbb, cab, aba ……

   R. E. = (a|b|c) (a|b|c) (a|b|c)

6. All binary string

   Strings: 0, 11, 101, 10101, 1111 …

   R. E. = $(0 \mid 1)^+$

Regular expression examples

7. 0 or more occurrence of either a or b or both
   $\boldsymbol{Strings}: \epsilon, \boldsymbol{a}, \boldsymbol{aa}, \boldsymbol{abab}, \boldsymbol{bab}$ ...

   $\boldsymbol{R.E.} = (\boldsymbol{a} \mid \boldsymbol{b}) *$

8. 1 or more occurrence of either a or b or both
   $\boldsymbol{Strings}: \boldsymbol{a}, \boldsymbol{aa}, \boldsymbol{abab}, \boldsymbol{bab}, \boldsymbol{bbbaaa}$ ...

   $\boldsymbol{R.E.} = (\boldsymbol{a} \mid \boldsymbol{b})^{+}$

9. Binary no. ends with 0
   $\boldsymbol{Strings}: \mathbf{0}, \mathbf{10}, \mathbf{100}, \mathbf{1010}, \mathbf{11110}$ ...

   $\boldsymbol{R.E.} = (\mathbf{0} \mid \mathbf{1})^{*}\ \mathbf{0}$

10. Binary no. ends with 1
    $\boldsymbol{Strings}: \mathbf{1}, \mathbf{101}, \mathbf{1001}, \mathbf{10101}, \ldots$

    $\boldsymbol{R.E.} = (\mathbf{0} \mid \mathbf{1}) * \mathbf{1}$

11. Binary no. starts and ends with 1
    $\boldsymbol{Strings}: \mathbf{11}, \mathbf{101}, \mathbf{1001}, \mathbf{10101}, \ldots$

    $\boldsymbol{R.E.} = \mathbf{1}\ (\mathbf{0} \mid \mathbf{1}) * \mathbf{1}$

12. String starts and ends with same character
    $\boldsymbol{Strings}: \mathbf{00}, \mathbf{101}, \boldsymbol{aba}, \boldsymbol{baab}$ ...

$$\boldsymbol{R.E.} = \mathbf{1}\ (\mathbf{0} \mid \mathbf{1}) * \mathbf{1} \ \text{ or } \ \mathbf{0}\ (\mathbf{0} \mid \mathbf{1}) * \mathbf{0}$$
$$\boldsymbol{a}\ (\boldsymbol{a} \mid \boldsymbol{b})^{*}\ \boldsymbol{a} \ \text{ or } \ \boldsymbol{b}\ (\boldsymbol{a} \mid \boldsymbol{b})^{*}\ \boldsymbol{b}$$

Regular expression examples

13. All string of a and b starting with a

$R.E. = a(a \mid b)^*$

$Strings: a, ab, aab, abb \ldots$

14. String of 0 and 1 ends with 00

$R.E. = (0 \mid 1) * 00$

$Strings: 00, 100, 000, 1000, 1100 \ldots$

15. String ends with abb

$R.E. = (a \mid b) * abb$

$Strings: abb, babb, ababb \ldots$

16. String starts with 1 and ends with 0

$R.E. = 1(0 \mid 1) * 0$

$Strings: 10, 100, 110, 1000, 1100 \ldots$

17. All binary string with at least 3 characters and 3rd character should be zero

$Strings: 000, 100, 1100, 1001 \ldots$

$R.E. = (0|1)(0|1)0(0 \mid 1) *$

18. Language which consist of exactly two b's over the set $\Sigma = \{a, b\}$

$Strings: bb, bab, aabb, abba \ldots$

$R.E. = a^* b a^* b a^*$

Regular expression examples

19. The language with $\Sigma = \{a, b\}$ such that 3rd character from right end of the string is always a.

$\boldsymbol{Strings: aaa, aba, aaba, abb\ldots}$ $\qquad \boldsymbol{R.E. = (a \mid b) * a(a|b)(a|b)}$

20. Any no. of a followed by any no. of b followed by any no. of c

$\boldsymbol{Strings: \epsilon, abc, aabbcc, aabc, abb\ldots}$ $\qquad \boldsymbol{R.E. = a^{*}\, b^{*}\, c^{*}}$

21. String should contain at least three 1

$\boldsymbol{Strings: 111, 01101, 0101110\ldots.}$ $\boldsymbol{R.E. = (0|1)^{*}1\ (0|1)^{*}1\ (0|1)^{*}1\ (0|1)^{*}}$

22. String should contain exactly two 1

$\boldsymbol{Strings: 11, 0101, 1100, 010010, 100100\ldots.}$ $\qquad \boldsymbol{R.E. = 0^{*}10^{*}10^{*}}$

23. Length of string should be at least 1 and at most 3

$\boldsymbol{Strings: 0, 1, 11, 01, 111, 010, 100\ldots.}$

$$\boldsymbol{R.E. = (0|1) \mid (0|1)(0|1) \mid (0|1)(0|1)(0|1)}$$

24. No. of zero should be multiple of 3

$\boldsymbol{Strings: 000, 010101, 110100, 000000, 100010010\ldots.}$

$$\boldsymbol{R.E. = (1^{*}01^{*}01^{*}01^{*})^{*}}$$

Regular expression examples

25. The language with $\Sigma = \{a, b, c\}$ where a should be multiple of 3

$\boldsymbol{Strings: aaa, baaa, bacaba, aaaaaa..}$

$$\boldsymbol{R.E. = ((b|c)^* a (b|c)^* a (b|c)^* a (b|c)^*)^*}$$

26. Even no. of 0

$\boldsymbol{Strings: 00, 0101, 0000, 100100....}$

$$\boldsymbol{R.E. = (1^* 01^* 01^*)^*}$$

27. String should have odd length

$\boldsymbol{Strings: 0, 010, 110, 000, 10010....}$

$$\boldsymbol{R.E. = (0|1)\ ((0|1)(0|1))^*}$$

28. String should have even length

$\boldsymbol{Strings: 00, 0101, 0000, 100100....}$

$$\boldsymbol{R.E. = ((0|1)(0|1))^*}$$

29. String start with 0 and has odd length

$\boldsymbol{Strings: 0, 010, 010, 000, 00010....}$

$$\boldsymbol{R.E. = (0)\ ((0|1)(0|1))^*}$$

30. String start with 1 and has even length

$\boldsymbol{Strings: 10, 1100, 1000, 100100....}$

$$\boldsymbol{R.E. = 1(0|1)((0|1)(0|1))^*}$$

Regular definition

- A regular definition gives names to certain regular expressions and uses those names in other regular expressions.
- Regular definition is a sequence of definitions of the form:

$$d1 \rightarrow r1$$
$$d2 \rightarrow r2$$
$$\ldots\ldots$$
$$dn \rightarrow rn$$

Where di is a distinct name & ri is a regular expression.

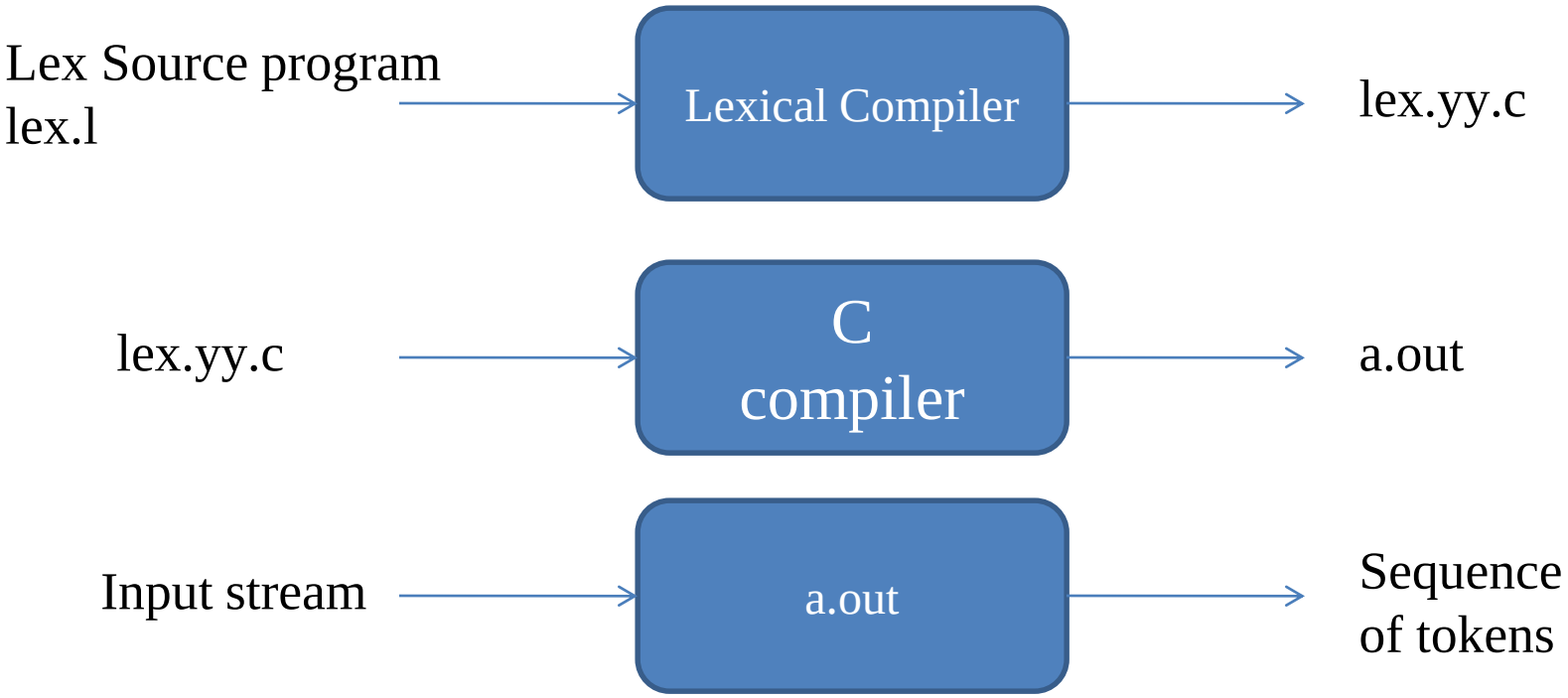
- Example: Regular definition for identifier

letter → A|B|C|……..|Z|a|b|……..|z

digit → 0|1|…….|9|

id → letter (letter | digit)*

Lexical Analyzer Generator - Lex



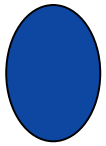
Finite Automata

- Finite Automata are recognizers.
  - ✓ FA simply say “Yes” or “No” about each possible input string.

- Finite Automata is a mathematical model consist of:
  1. Set of states ***S***
  2. Set of input symbol ***Σ***
  3. A transition function ***move***
  4. Initial state S_0
  5. Final states or accepting states **F**

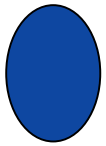
Transition diagrams

A stylized flowchart is called transition diagram.



is a state

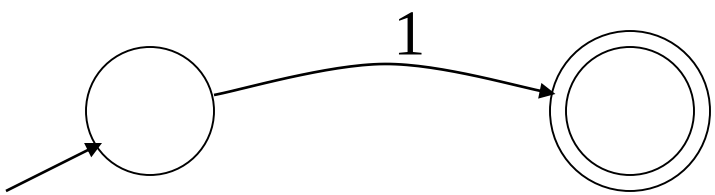
is a transition



is a start state

is a final state

A Simple Example

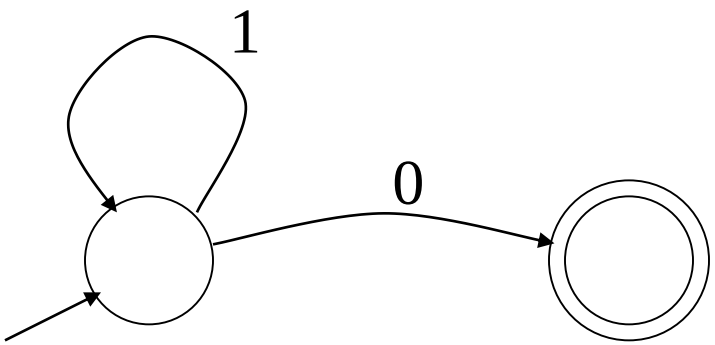


- A finite automaton that accepts only “1”
- A finite automaton accepts a string if we can follow transitions labeled with the characters in the string from the start to some accepting state

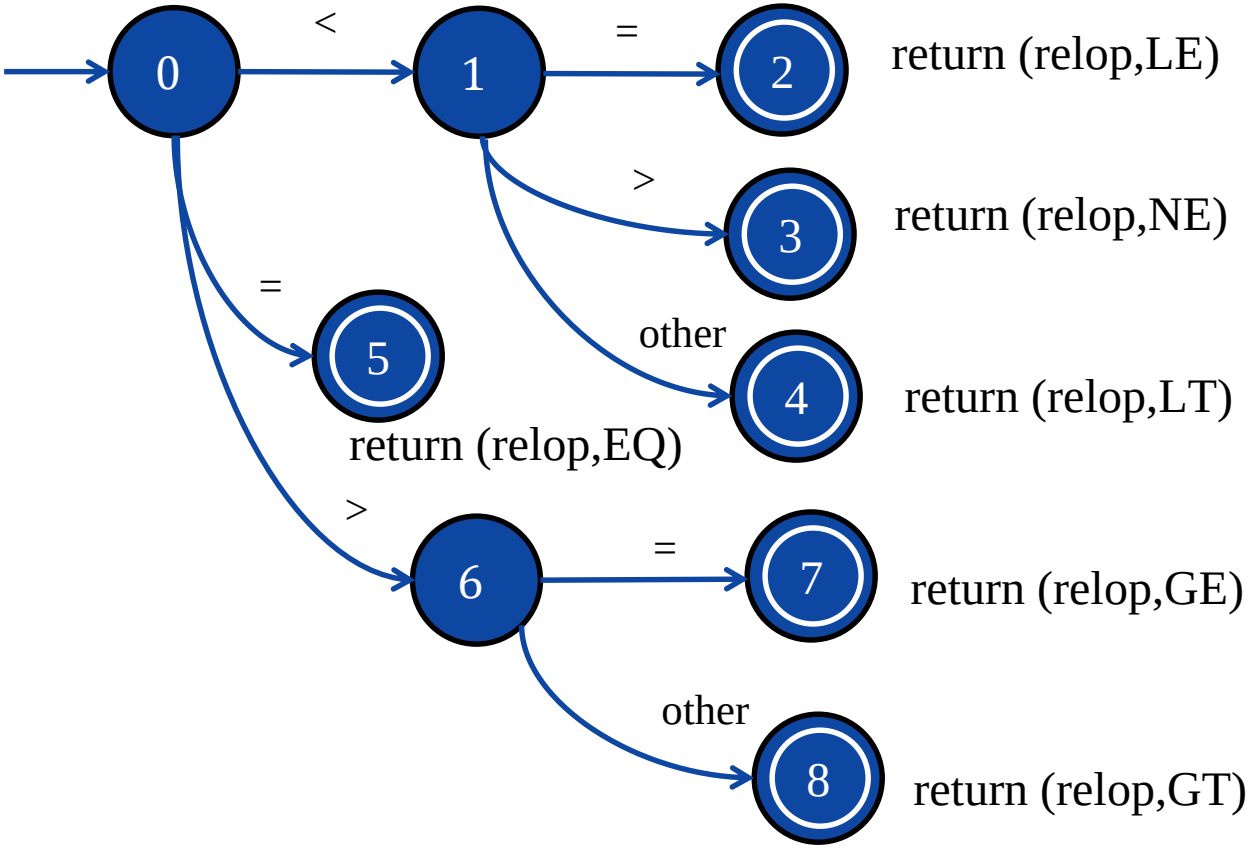
Another Simple Example

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

Alphabet: {0,1}

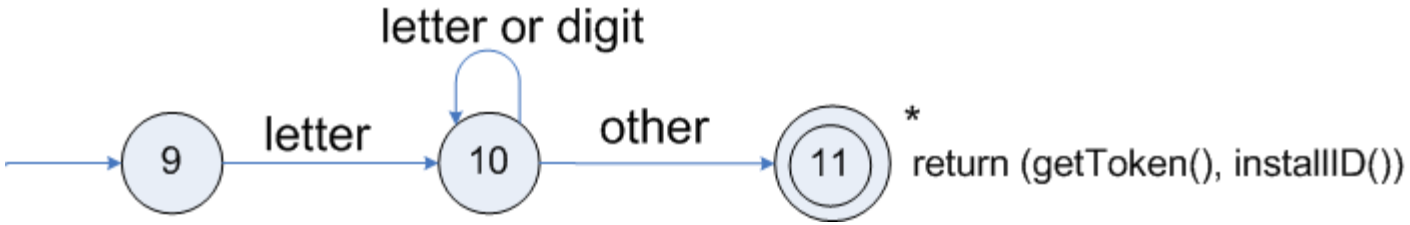


Relational operator Example



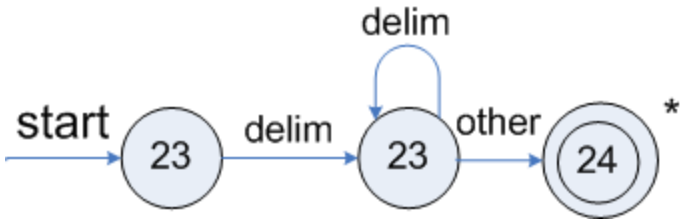
Transition diagrams (cont.)

Transition diagram for reserved words and identifiers



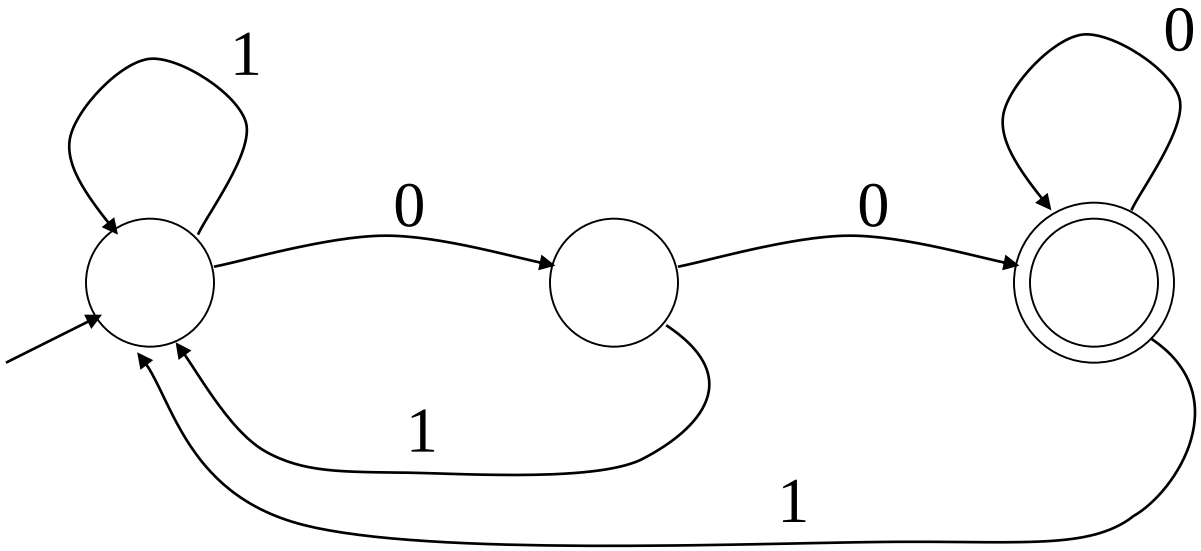
Transition diagrams (cont.)

Transition diagram for whitespace



Example

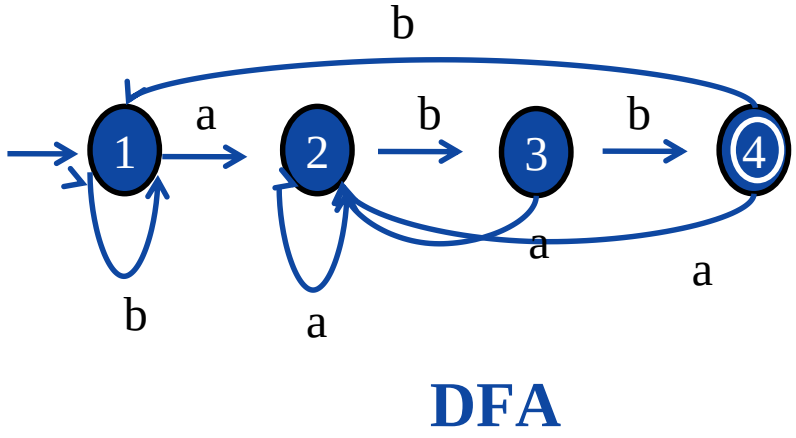
Alphabet {0,1}
What language does this recognize?



Types of finite automata

DFA

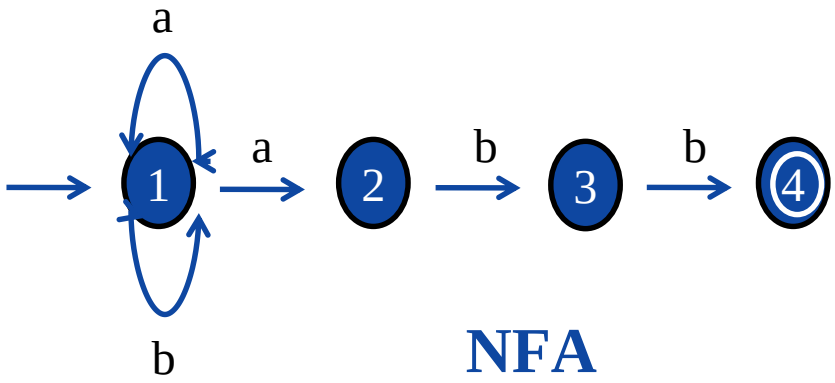
- Deterministic finite automata (DFA): have for each state exactly one edge leaving out for each symbol.



DFA

NFA

- Nondeterministic finite automata (NFA): There are no restrictions on the edges leaving a state. There can be several with the same symbol as label and some edges can be labeled with ϵ.



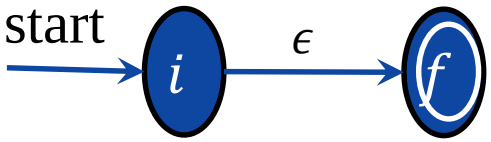
NFA

Regular expression to NFA using Thompson's rule

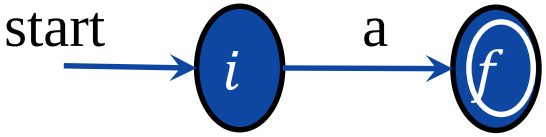
- **Thompson's rule:** Break the regular expression into small parts and **build small NFAs**, then **join them using rules**.

Regular expression to NFA using Thompson's rule

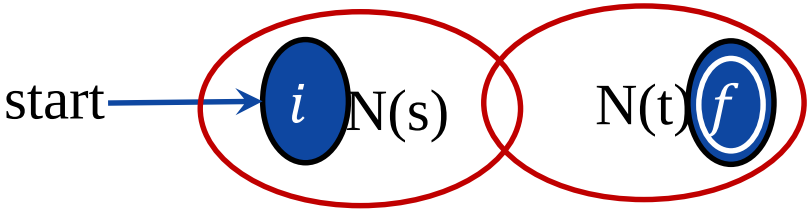
1. For ∈ , construct the NFA



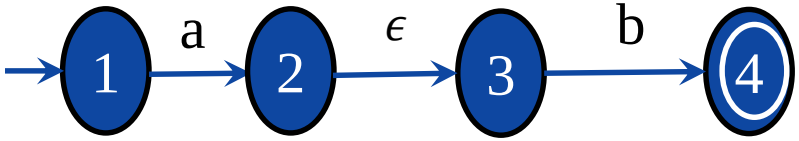
2. For a in Σ, construct the NFA



3. For regular expression st

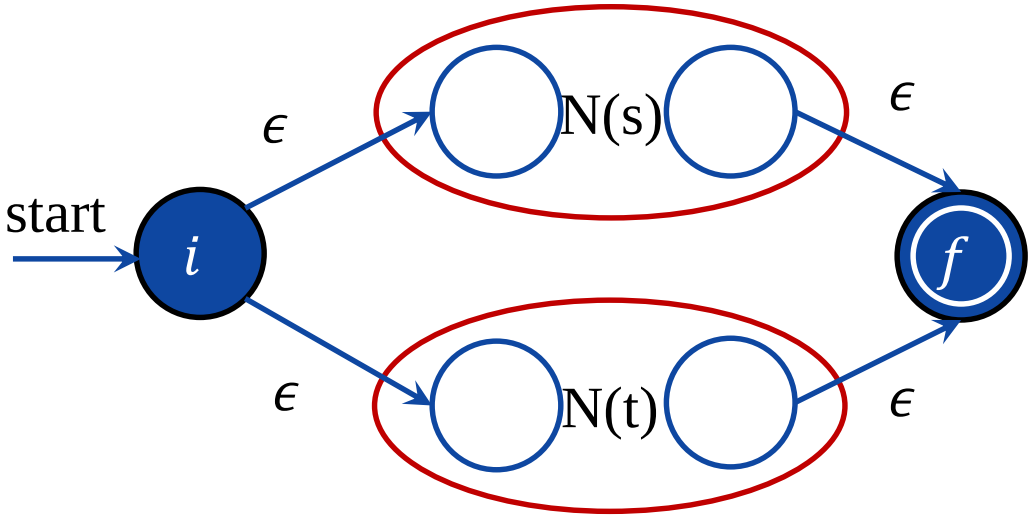


Ex: ab

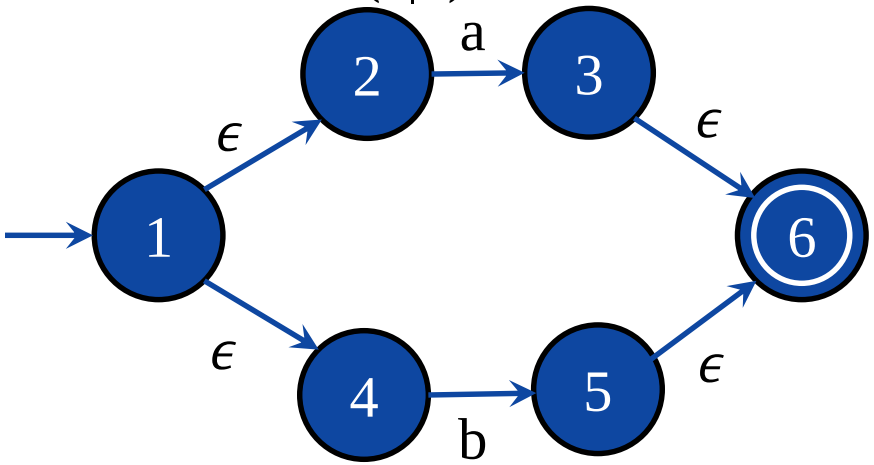


Regular expression to NFA using Thompson's rule

4. For regular expression $s|t$

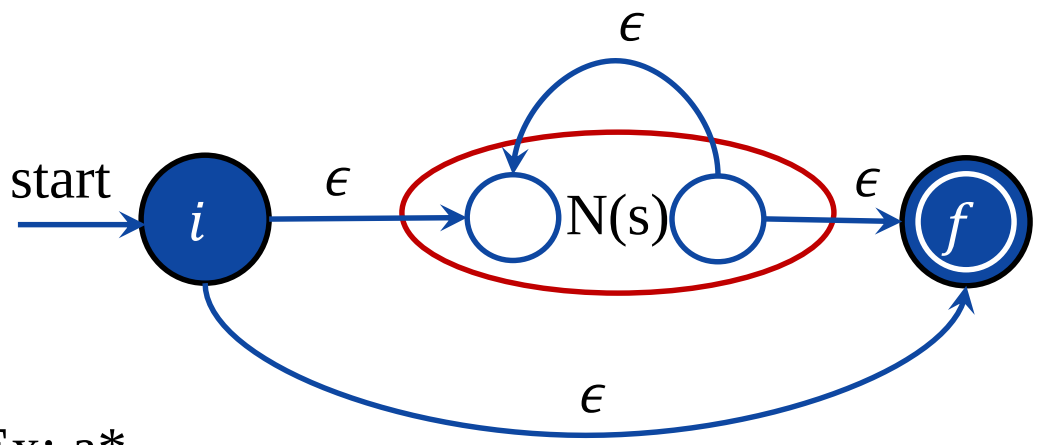


Ex: (a|b)

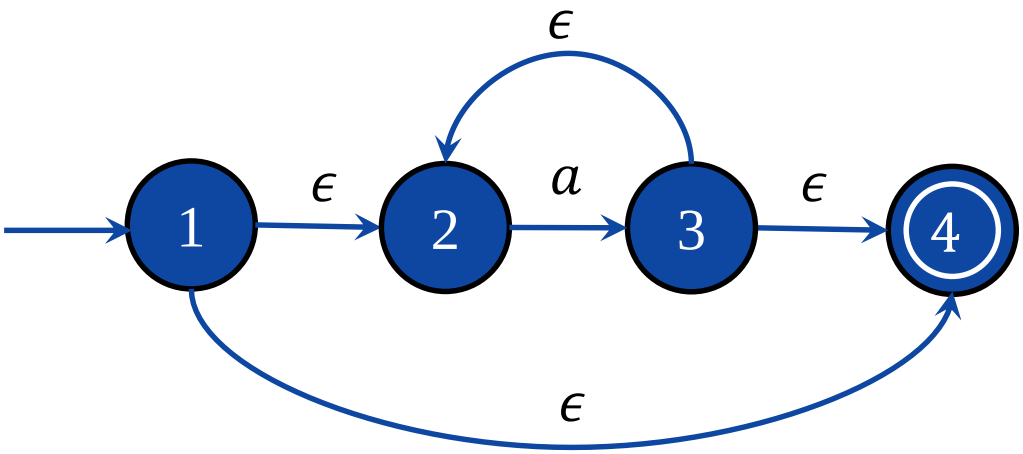


Regular expression to NFA using Thompson's rule

5. For regular expression *s**

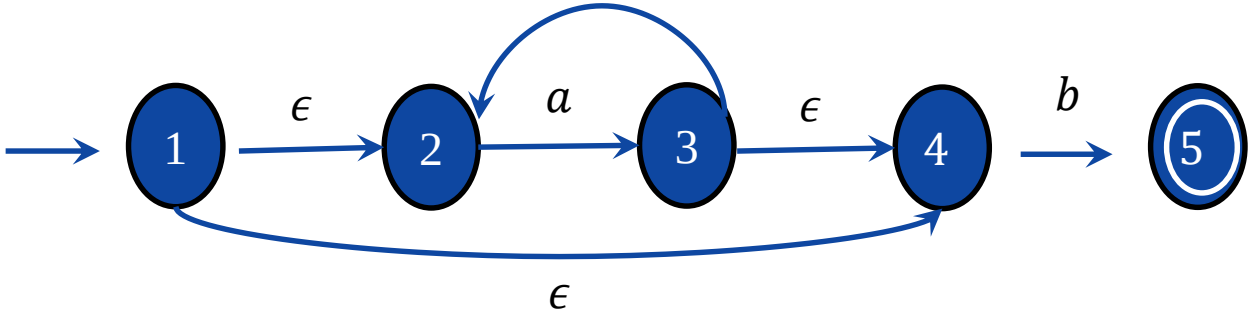


Ex: a*

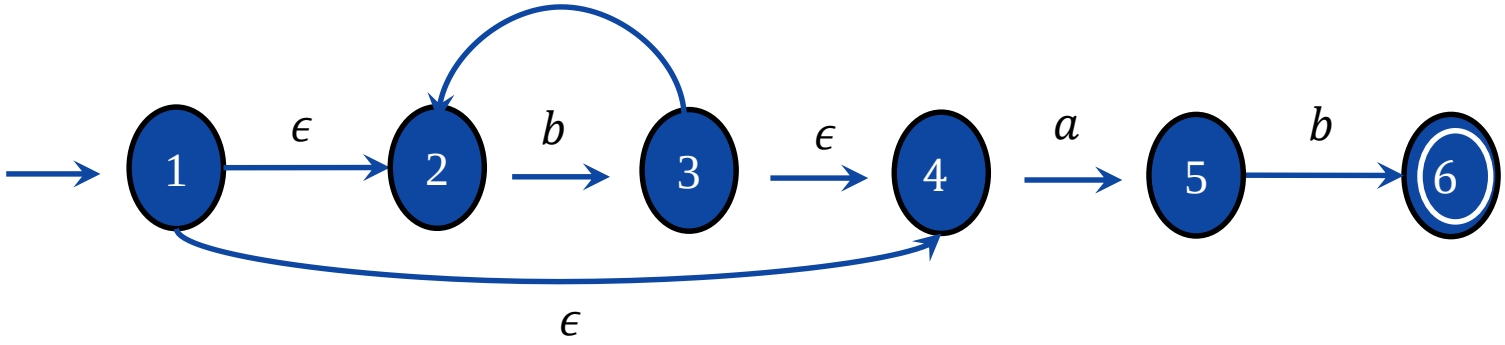


Regular expression to NFA using Thompson's rule

a*b

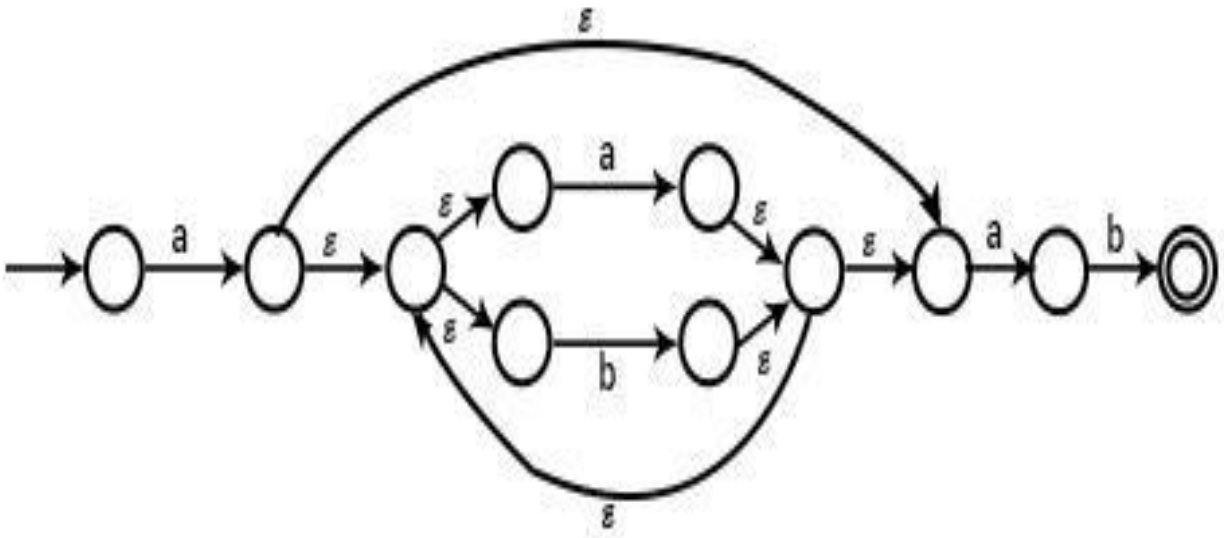


b*ab



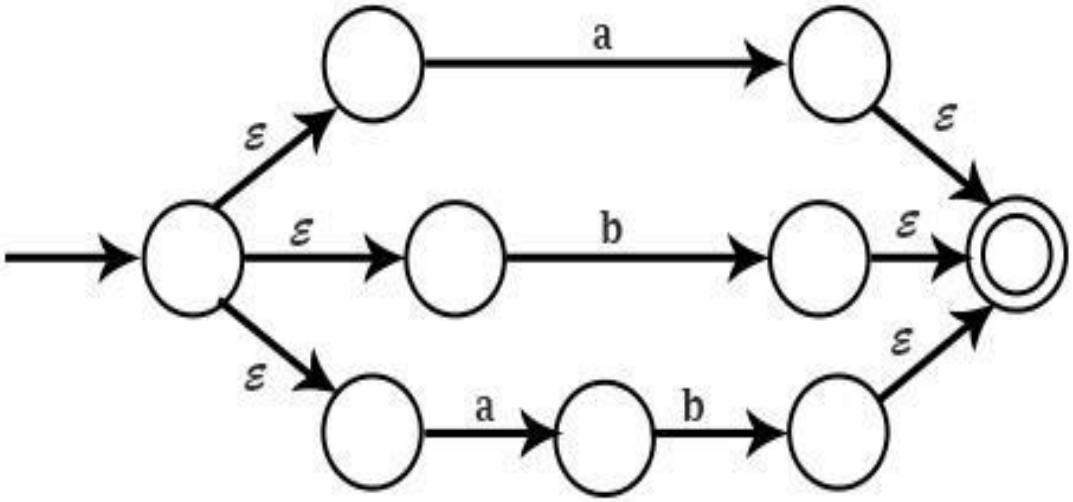
Regular expression to NFA using Thompson's rule

- Draw NFA for the Regular Expression a(a+b)*ab



Regular expression to NFA using Thompson's rule

- Draw NFA for the Regular Expression a + b + ab



Assignment 1

Regular expression to NFA using Thompson's rule

Convert following regular expression to NFA:

1. abba
2. bb(a)*
3. (a|b)*
4. a* | b*
5. a(a)*ab
6. aa*+ bb*
7. (a+b)*abb
8. 10(0+1)*1
9. (a+b)*a(a+b)
10. (0+1)*010(0+1)*
11. (010+00)*(10)*
12. 100(1)*00(0+1)*

Non-Deterministic Finite Automata (NFA)

- Non-Deterministic Finite Automata is defined by the quintuple-

$$\mathbf{M = (Q, \sum, \delta, q0, F)}$$

where-

- Q = finite set of states
- $\sum$ = non-empty finite set of symbols called as input alphabets
- $\delta : Q \times \sum \rightarrow 2Q$ is a total function called as transition function
- $q0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of final states

Deterministic Finite Automata (NFA)

- A DFA is a collection of 5-tuples same as we described in the definition of FA.

$$\mathbf{M = (Q, \sum, \delta, q0, F)}$$

Q: finite set of states

$\sum$: finite set of the input symbol

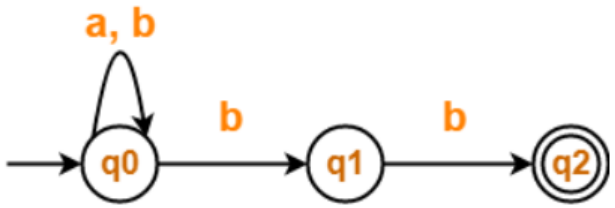
q0: initial state

F: final state

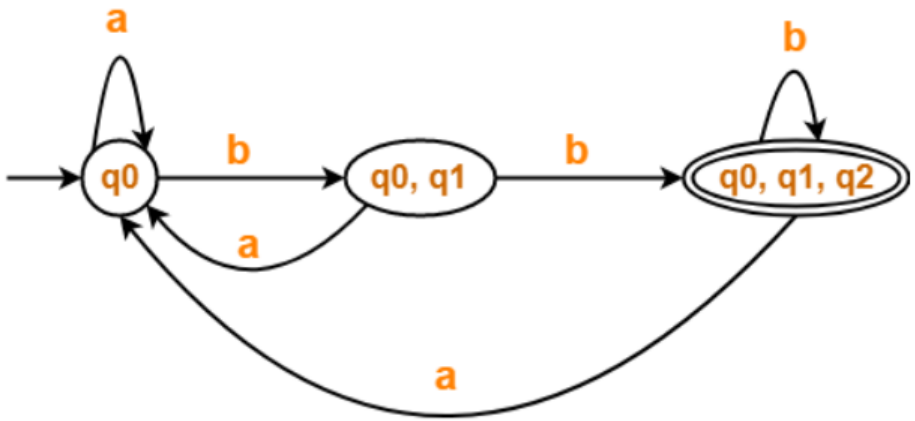
δ: Transition function δ: Q x $\sum \rightarrow$ Q

Example 1 NFA to DFA Conversion

NFA



DFA

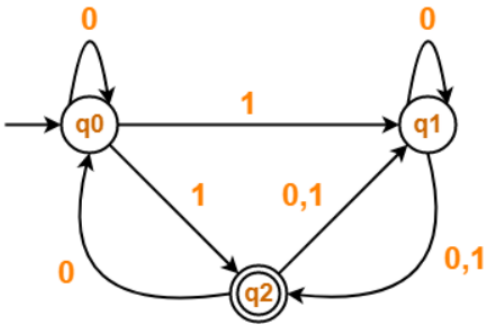


State / Alphabet	a	b
→q0	q0	q0, q1
q1	–	*q2
*q2	–	–

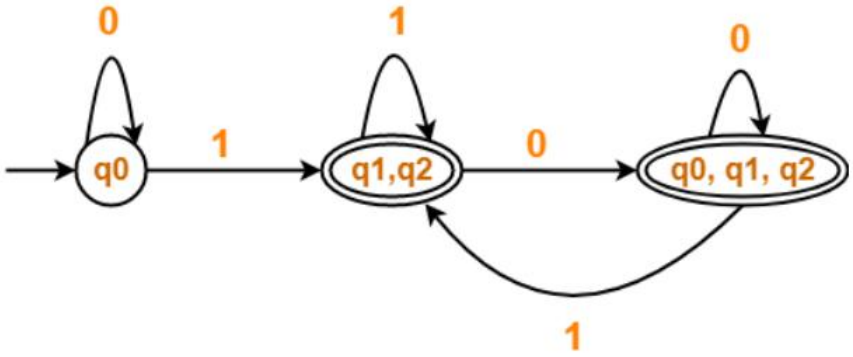
State / Alphabet	a	b
→q0	q0	{q0, q1}
{q0, q1}	q0	*{q0, q1, q2}
*{q0, q1, q2}	q0	*{q0, q1, q2}

Example 2 NFA to DFA conversion

NFA



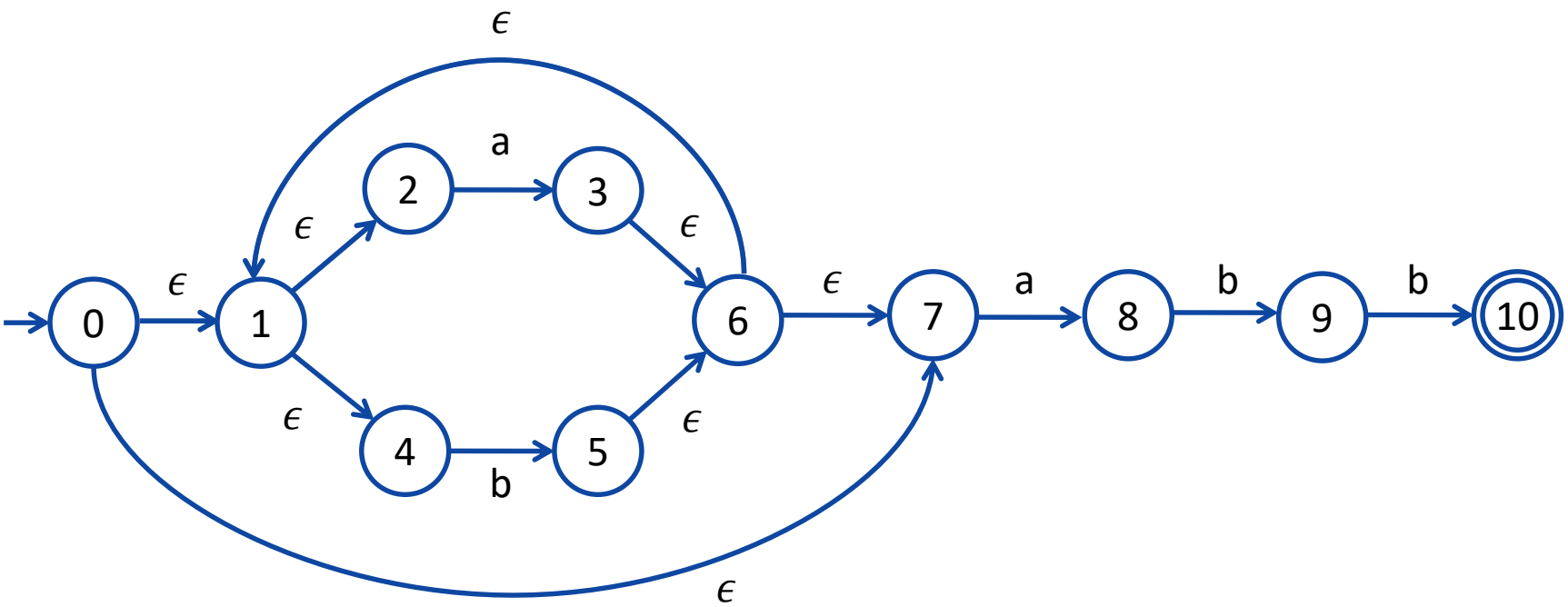
DFA



State / Alphabet	0	1
→**q0**	q0	q1, *q2
q1	q1, *q2	*q2
***q2**	q0, q1	q1

State / Alphabet	0	1
→**q0**	q0	*{q1, q2}
***{q1, q2}**	*{q0, q1, q2}	*{q1, q2}
***{q0, q1, q2}**	*{q0, q1, q2}	*{q1, q2}

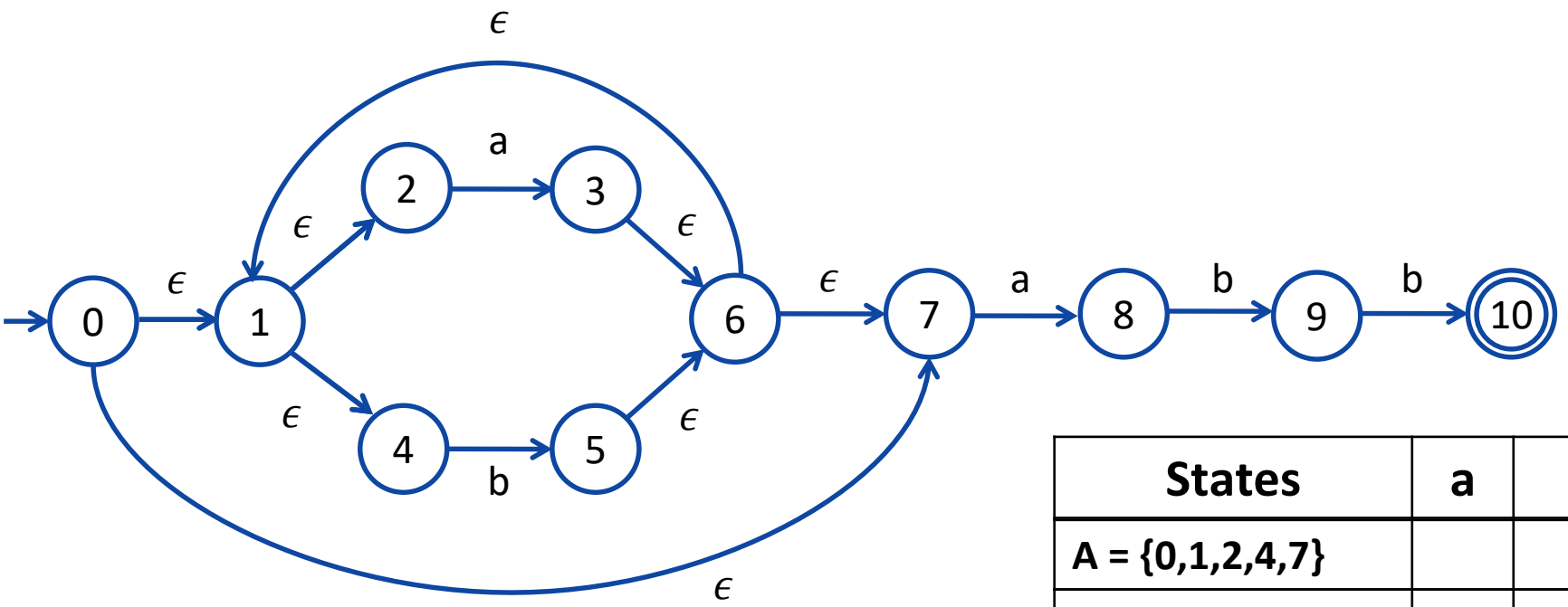
Conversion from NFA to DFA



ϵ- Closure(0)=

= {0,1,2,4,7} **---- A**

Conversion from NFA to DFA



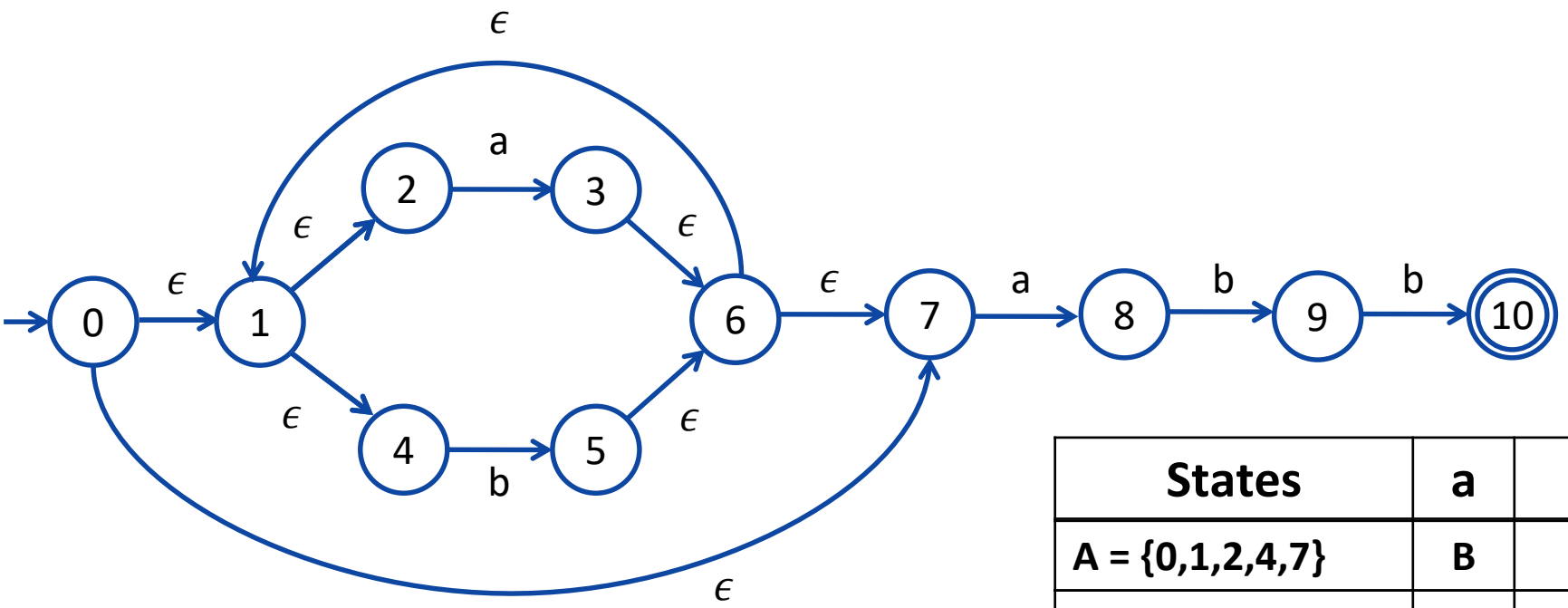
States	a	b
A = {0,1,2,4,7}		
B = {1,2,3,4,6,7,8}		

A= {0, 1, 2, 4, 7}

Move(A,a) = {3,8}

ϵ- Closure(Move(A,a)) = {1,2,3,4,6,7,8} **---- B**

Conversion from NFA to DFA



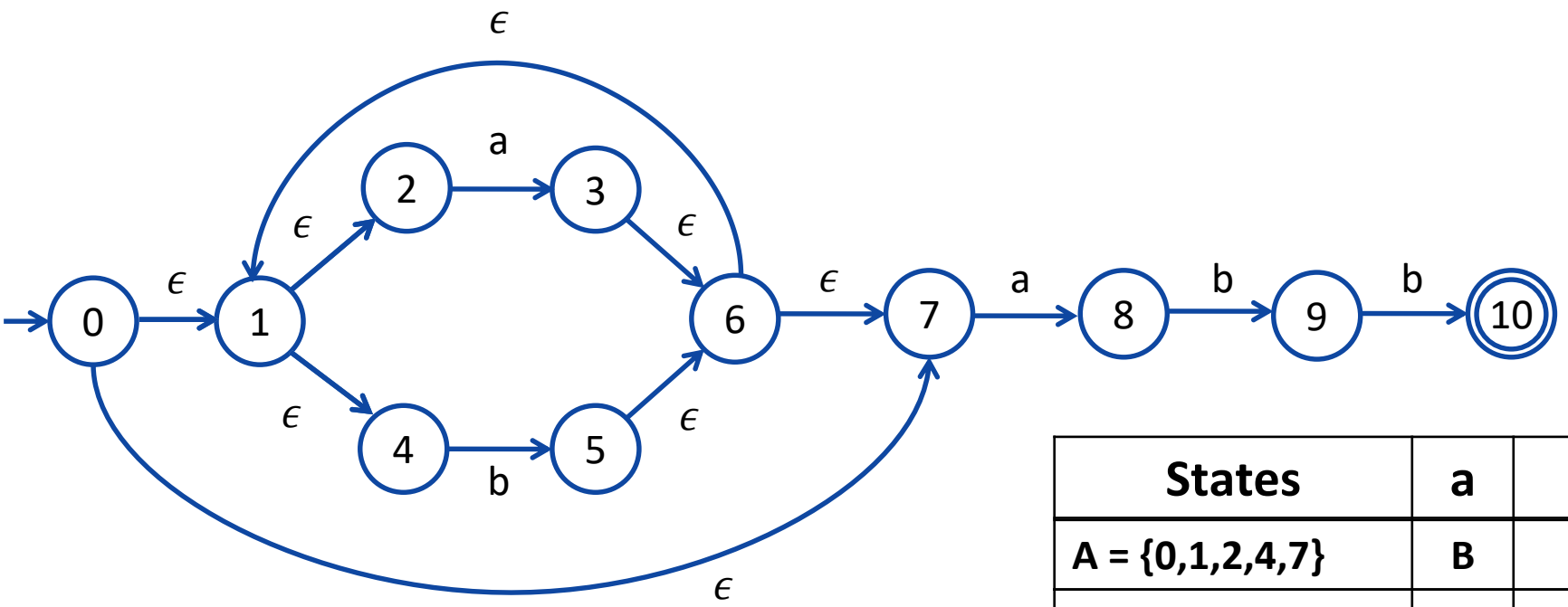
States	a	b
A = {0,1,2,4,7}	B	
B = {1,2,3,4,6,7,8}		
C = {1,2,4,5,6,7}		

A= {0, 1, 2, 4, 7}

Move(A,b) = {5,6}

ϵ- Closure(Move(A,b)) = {1,2,4,5,6,7} **---- C**

Conversion from NFA to DFA



States	a	b
A = {0,1,2,4,7}	B	C
B = {1,2,3,4,6,7,8}		
C = {1,2,4,5,6,7}		

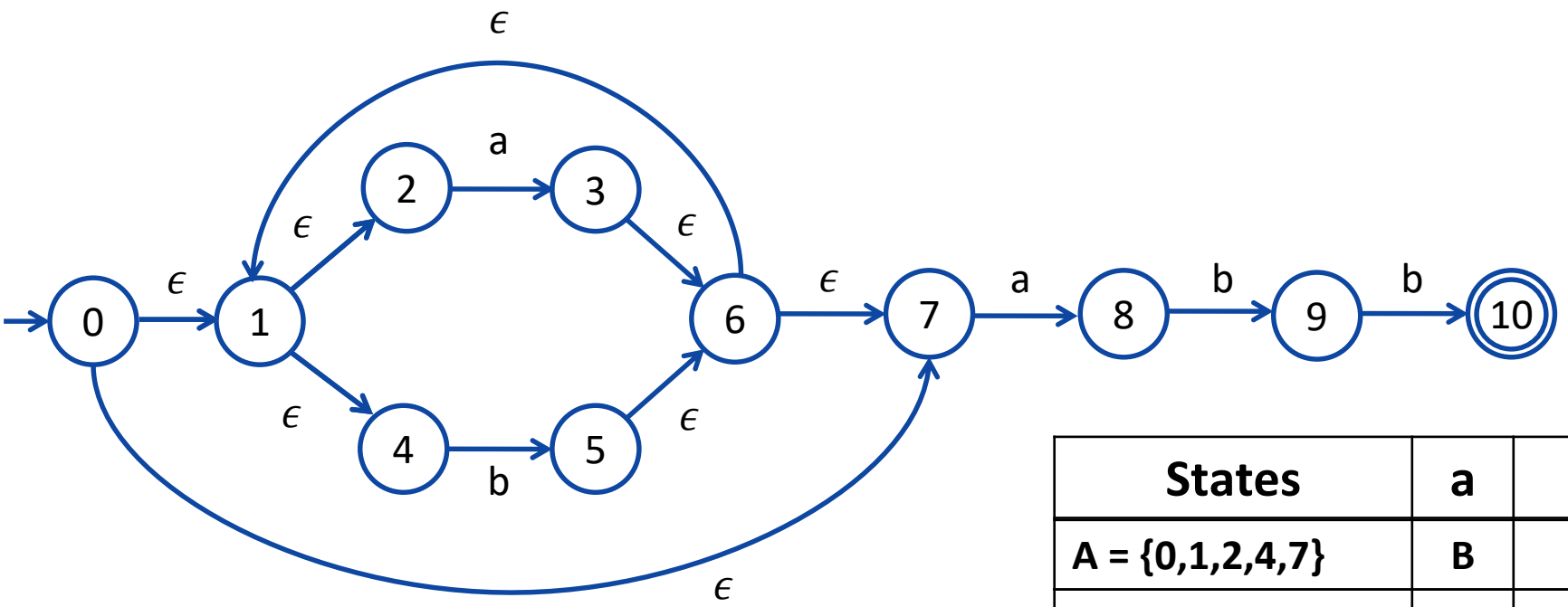
B = {1, 2, 3, 4, 6, 7, 8}

Move(B,a) = {3,8}

ϵ- Closure(Move(B,a))

= {1,2,3,4,6,7,8} **---- B**

Conversion from NFA to DFA



B = {1, 2, 3, 4, 6, 7, 8}

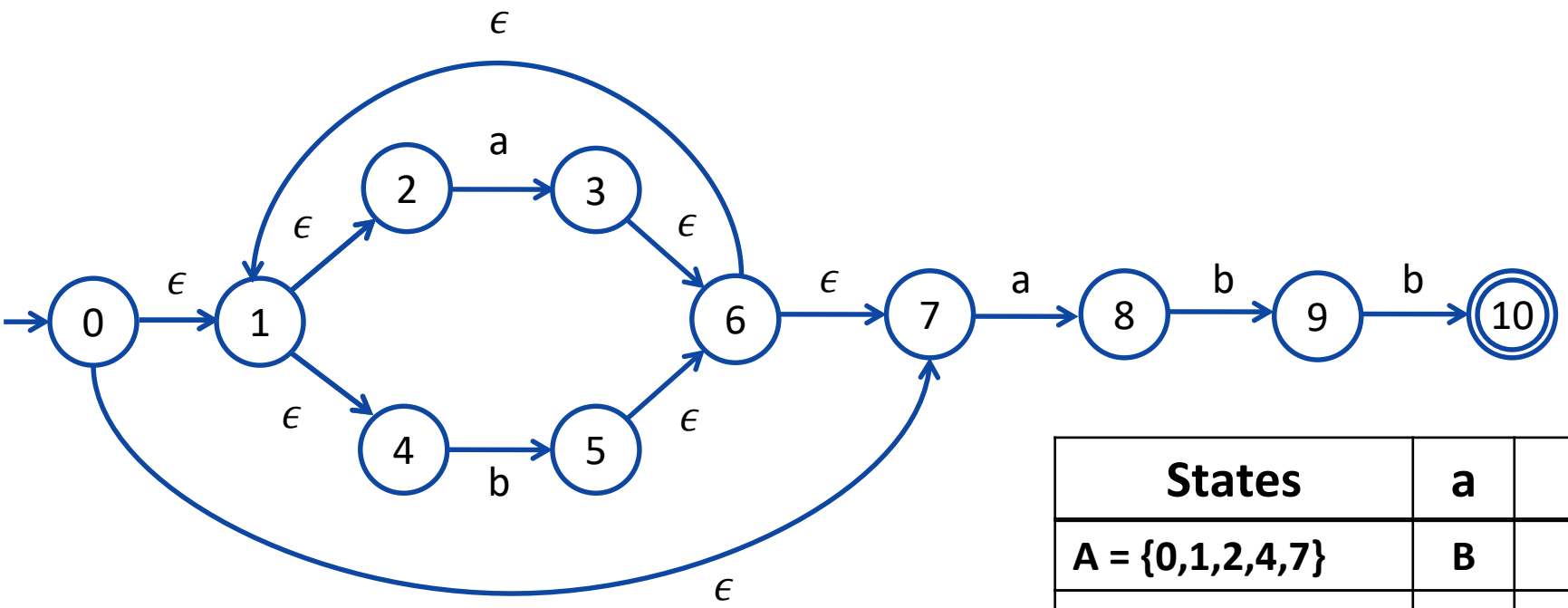
Move(B,b) = {5,9}

ϵ- Closure(Move(B,b))

= {1,2,4,5,6,7,9} **---- D**

States	a	b
A = {0,1,2,4,7}	B	C
B = {1,2,3,4,6,7,8}	B	
C = {1,2,4,5,6,7}		
D = {1,2,4,5,6,7,9}		

Conversion from NFA to DFA



States	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}		
D = {1,2,4,5,6,7,9}		

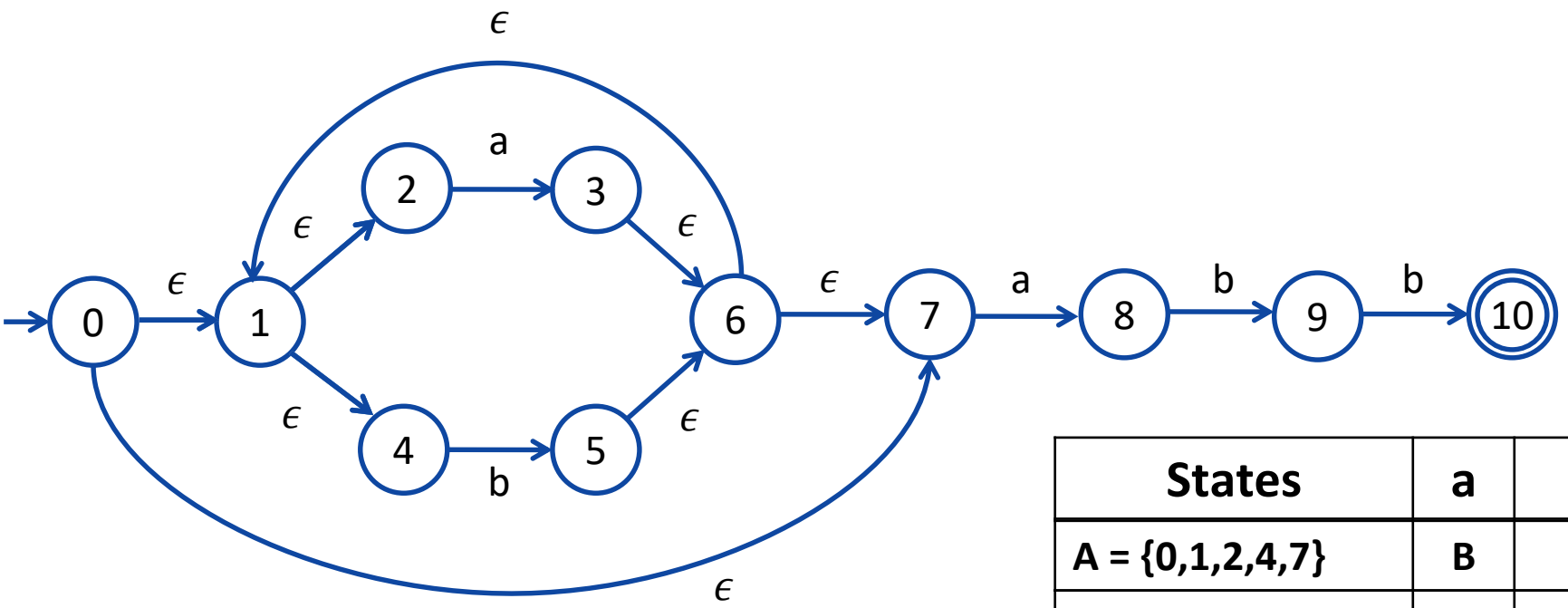
C= {1, 2, 4, 5, 6 ,7}

Move(C,a) = {3,8}

ϵ- Closure(Move(C,a))

= {1,2,3,4,6,7,8} **---- B**

Conversion from NFA to DFA



States	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	
D = {1,2,4,5,6,7,9}		

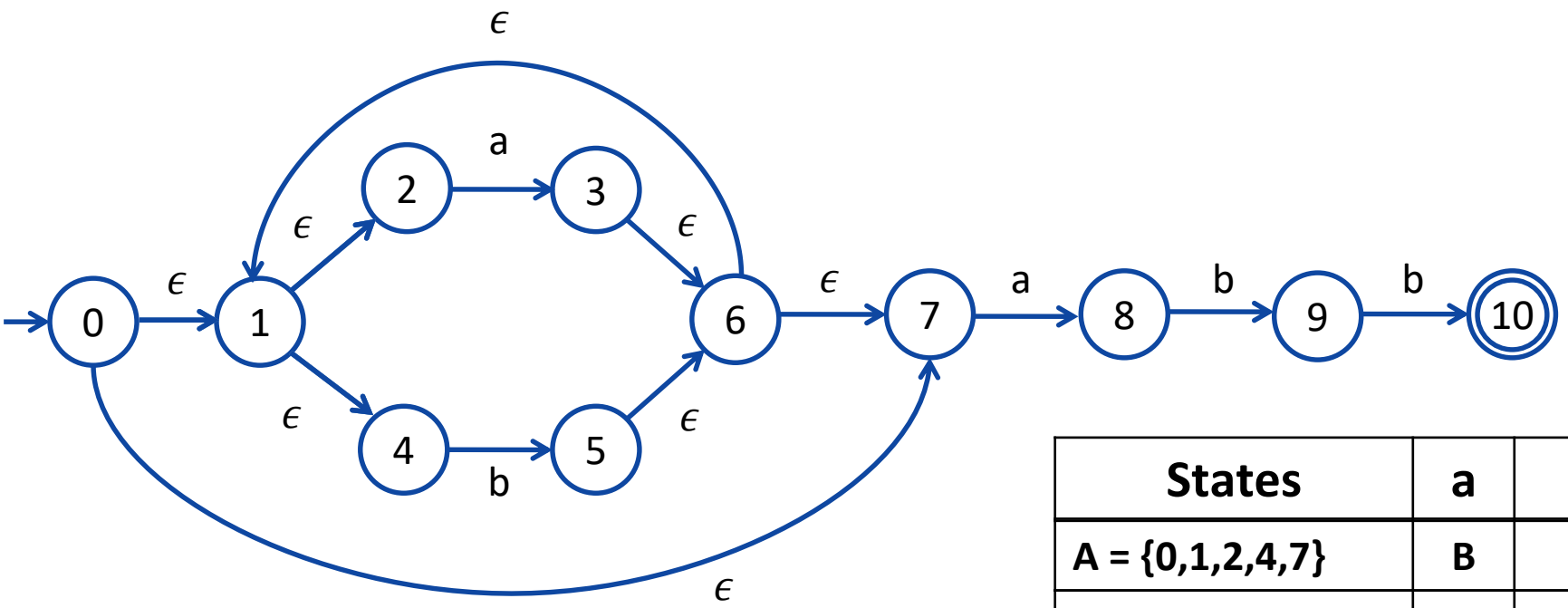
C= {1, 2, 4, 5, 6 ,7}

Move(C,a) =

ϵ- Closure(Move(C,b))

= {1,2,4,5,6,7} **---- C**

Conversion from NFA to DFA



States	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}		

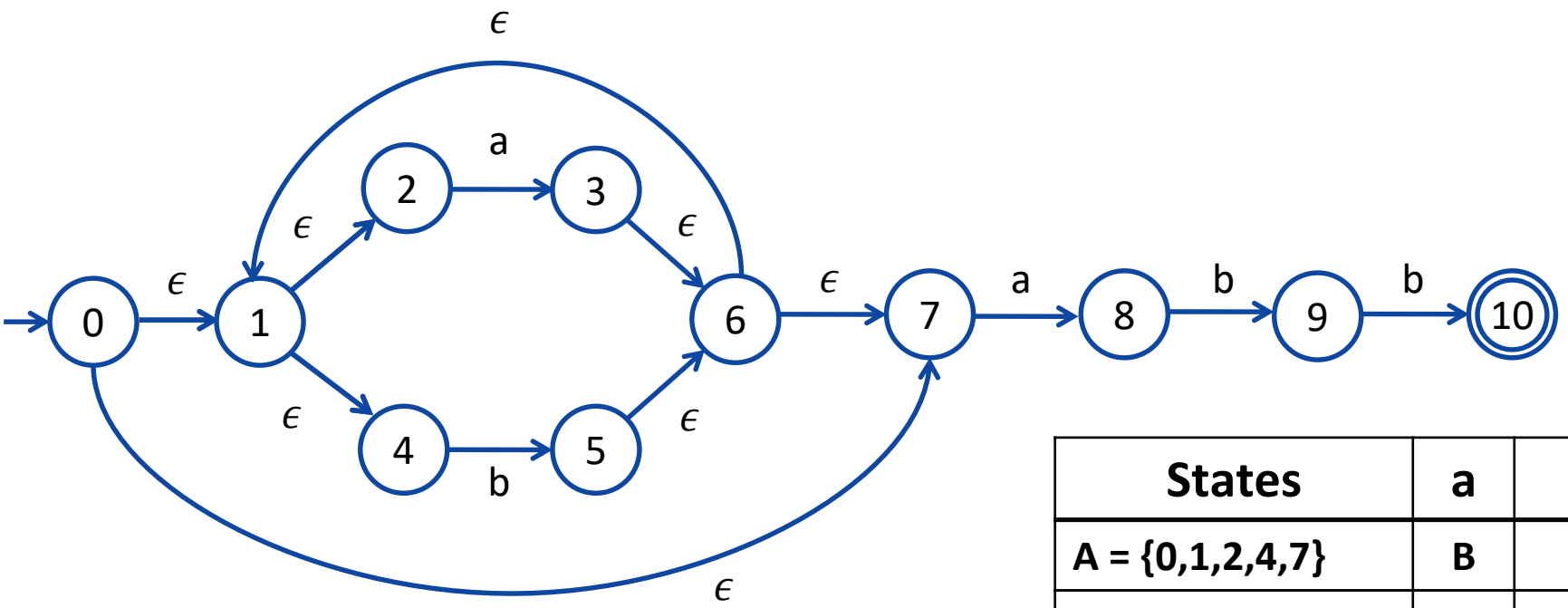
D= {1, 2, 4, 5, 6, 7, 9}

Move(D,a) = {3,8}

ϵ- Closure(Move(D,a))

= {1,2,3,4,6,7,8} **---- B**

Conversion from NFA to DFA



States	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 = {1,2,4,5,6,7,10}		

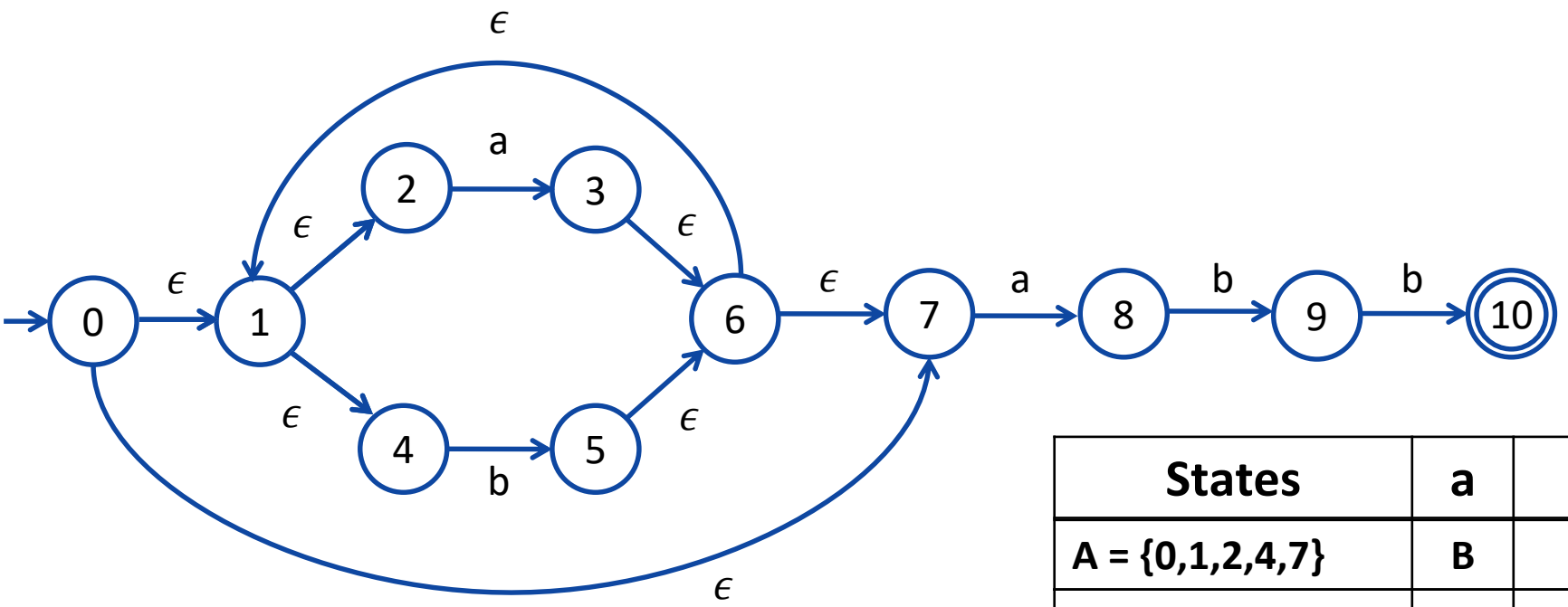
D= {1, 2, 4, 5, 6, 7, 9}

Move(D,b) = {5,10}

ϵ- Closure(Move(D,b))

= {1,2,4,5,6,7,10} **---- E**

Conversion from NFA to DFA



E= {1, 2, 4, 5, 6, 7, 10}

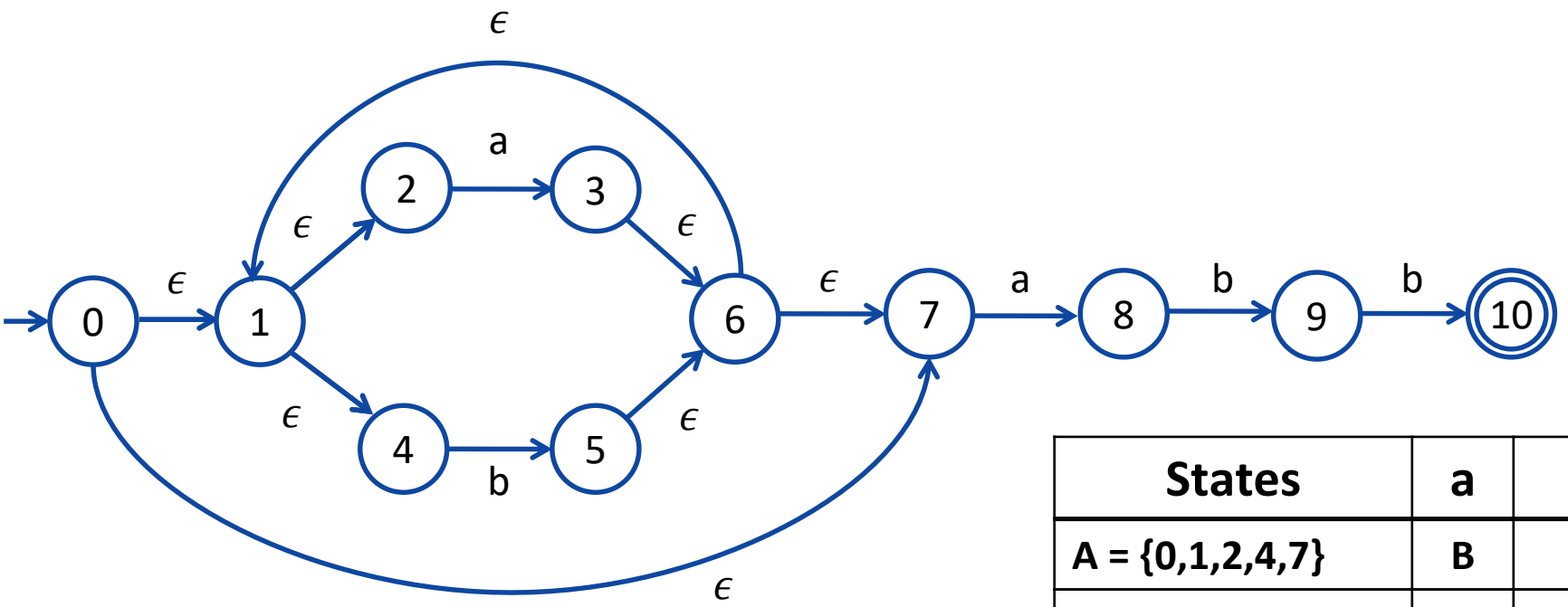
Move(E,a) = {3,8}

ϵ- Closure(Move(E,a))

= {1,2,3,4,6,7,8} **---- B**

States	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}		

Conversion from NFA to DFA



E= {1, 2, 4, 5, 6, 7, 10}

Move(E,b) =

ϵ- Closure(Move(E,b))

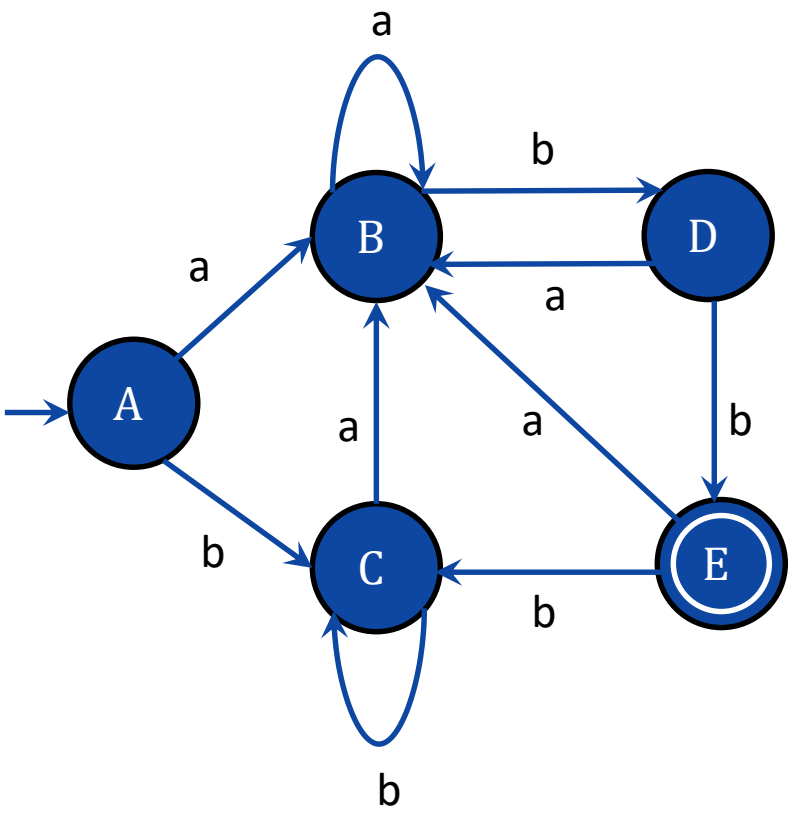
= {1,2,4,5,6,7} **---- C**

States	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}	B	

Conversion from NFA to DFA

States	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}	B	C

Transition Table



DFA

Note:

- **Accepting state in NFA is 10**
- **10 is element of E**
- **So, E is acceptance state in DFA**

End of Unit 2

Thank You