Unit – 2 & 3

Regular Languages and Finite Automata

&

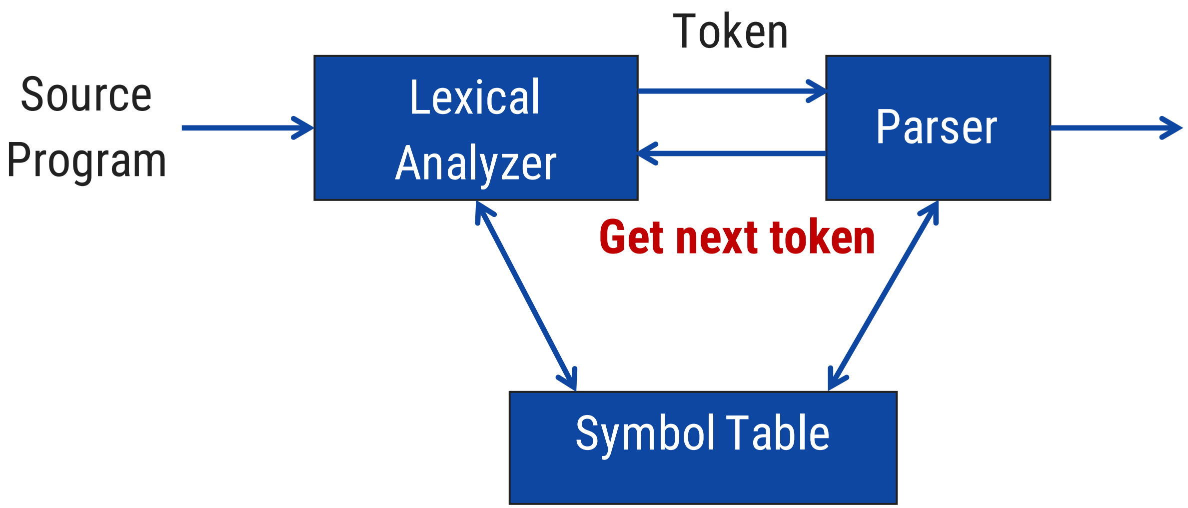
Lexical Analysis

Topics to be covered

- Interaction of scanner & parser
- Token, Pattern & Lexemes
- Input buffering
- Specification of tokens
- Regular expression & Regular definition
- Transition diagram
- Hard coding & automatic generation lexical analyzers
- Finite automata
- Regular expression to NFA using Thompson's rule
- Conversion from NFA to DFA using subset construction method
- DFA optimization
- Conversion from regular expression to DFA
- An Elementary Scanner Design & It’s Implementation

Interaction with Scanner & Parser

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.

Why to separate lexical analysis & parsing?

1. Simplicity in design.
2. Improves compiler efficiency.
3. Enhance compiler portability.

Token, Pattern & Lexemes

Token, Pattern & Lexemes

Token

A **token** is a pair consisting of a token name and an optional attribute value.

The token name is an abstract symbol representing a kind of lexical unit

Categories of Tokens:

1. Identifier
2. Keyword
3. Operator
4. Special symbol
5. Constant

Pattern

A **pattern** is a description of the form that the lexemes of a token may take.

Example: "*non-empty sequence of digits*", "*letter followed by letters and digits*"

Lexemes

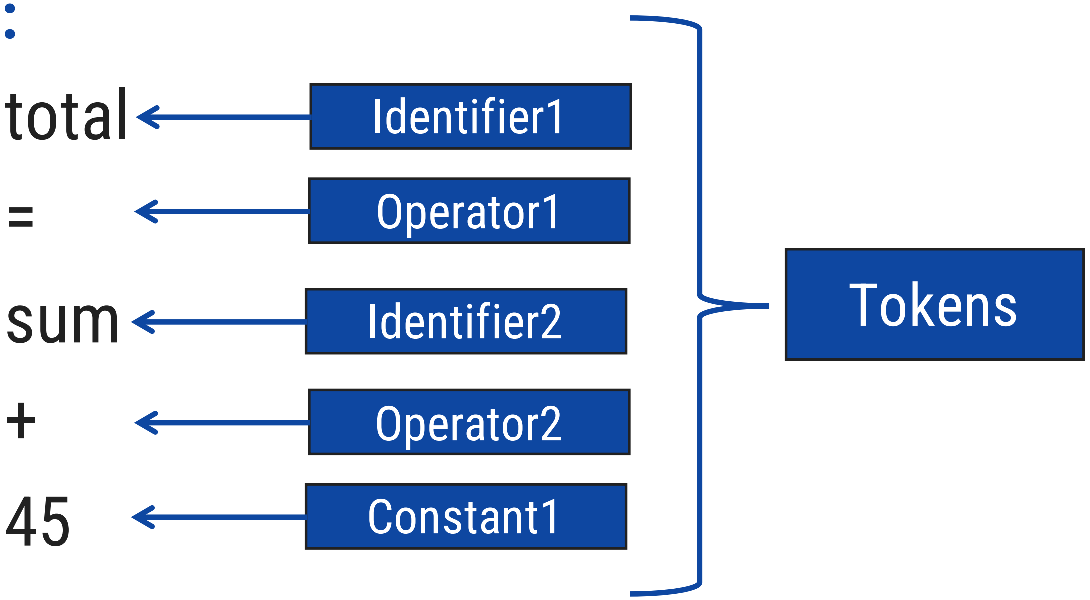
The sequence of character in a source program matched with a pattern for a token is called lexeme.

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

Example: Token, Pattern & Lexemes

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

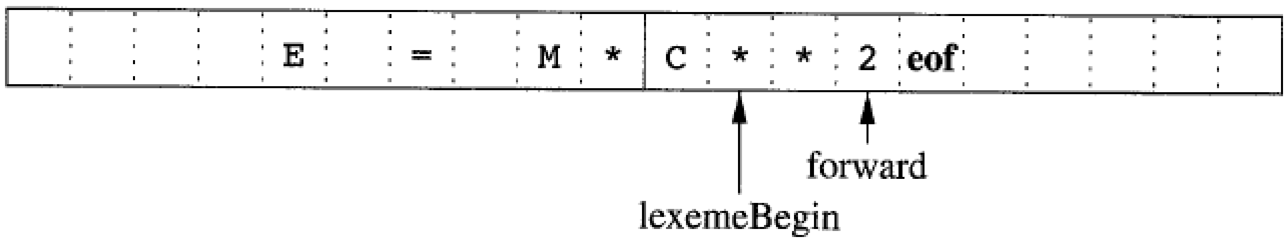
Input buffering

Input buffering

- There are mainly two techniques for input buffering:
  1. Buffer pairs
  2. Sentinels

Buffer Pair

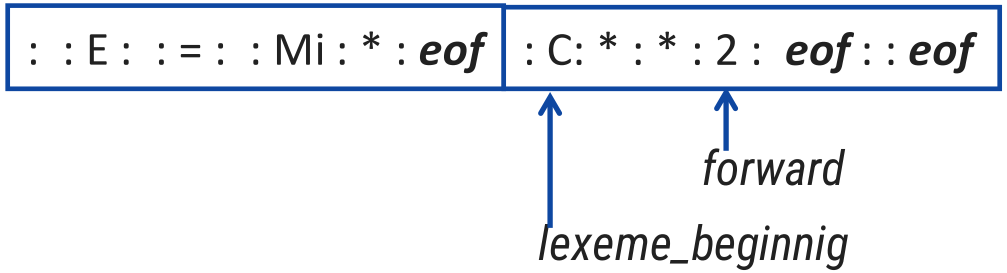
- The lexical analysis scans the input string from left to right one character at a time.
- Each buffer is of the same size N, and N is usually the size of a disk block, e.g., 4096 bytes.
- Using one system read command we can read N characters inio a buffer, rather than using one system call per character. If fewer than N characters remain in the input file, then a special character, represented by eof, marks the end of the source file.



Buffer pairs

- Two pointers to the input are maintained:
  - **Pointer lexemeBegin**, marks the beginning of the current lexeme, whose extent we are attempting to determine.
  - **Pointer forward** scans ahead until a pattern match is found

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**.

Specification of tokens

Strings and Languages

- Regular expressions are an important notation for specifying lexeme patterns.
- An **alphabet** is any finite set of symbols.
- Typical examples of symbols are letters, digits, and punctuation. The set {0,1) is the binary alphabet
- A **string** over an alphabet is a finite sequence of symbols drawn from that alphabet.
- The length of a string s, usually written Isl, is the number of occurrences of symbols in s.
- For example, banana is a string of length six.
- The empty string, denoted ∈, is the string of length zero.
- A **language** is any countable set of strings over some fixed alphabet.
- Abstract languages like Ø**,** the empty set, or {∈}, the set containing only the empty string, are languages under this definition.

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 x that is respectively proper prefix, suffix or substring of S, such that **s≠x**.
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	$L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$
Concatenation of L and M Written LM	$LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$
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 & Regular Definition

Regular expression

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

Notational shorthand's

1. One or more instances: +
2. Zero or more instances: *
3. Zero or one instances: ?
4. Alphabets: Σ

Rules to define regular expression

1. $\in$ is a regular expression that denotes $\{\in\}$, the set containing empty string.
2. If a is a symbol in Σ then a is a regular expression, $L(a) = \{a\}$
3. Suppose $r$and $s$are regular expression denoting the languages $L(r)$ and $L(s)$. Then,
   a. $(\mathrm{r})|(\mathrm{s})$ is a regular expression denoting $\mathrm{L(r) \cup L(s)}$
   b. $(\mathrm{r})(\mathrm{s})$ is a regular expression denoting $\mathrm{L(r)L(s)}$
   c. (r)* is a regular expression denoting $\left(\mathrm{L(r)}\right)^{*}$
   d. (r) is a regular expression denoting $\mathrm{L((r))}$

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

   $\mathbf{Strings:\ 0, 1}$ $\mathbf{R.E. = 0 \mid 1}$

2. 0 or 11 or 111

   $\mathbf{Strings:\ 0, 11, 111}$ $\mathbf{R.E. = 0 \mid 11 \mid 111}$

3. String having zero or more *a*.

   $\mathbf{Strings:\ \epsilon, a, aa, aaa, aaaa \ldots..}$ $\mathbf{R.E. = a^{*}}$

4. String having one or more *a*.

   $\mathbf{Strings: a, aa, aaa, aaaa \ldots..}$ $\mathbf{R.E. = a^{+}}$

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

   $\mathbf{Strings: abc, bca, bbb, cab, aba \ldots.}$ $\mathbf{R.E. = (a|b|c)\ (a|b|c)\ (a|b|c)}$

6. All binary string

   $\mathbf{Strings:\ 0, 11, 101, 10101, 1111 \ldots}$ $\mathbf{R.E. = (0 \mid 1)^{+}}$

Regular expression examples

7. 0 or more occurrence of either a or b or both

   $\boldsymbol{Strings: \epsilon, a, aa, abab, bab \ldots}$ $\boldsymbol{R.E. = (a \mid b) *}$

8. 1 or more occurrence of either a or b or both

   $\boldsymbol{Strings: a, aa, abab, bab, bbbaaa \ldots}$ $\boldsymbol{R.E. = (a \mid b)^{+}}$

9. Binary no. ends with 0

   $\boldsymbol{Strings: 0, 10, 100, 1010, 11110 \ldots}$ $\boldsymbol{R.E. = (0 \mid 1)^{*}\, 0}$

10. Binary no. ends with 1

    $\boldsymbol{Strings: 1, 101, 1001, 10101, \ldots}$ $\boldsymbol{R.E. = (0 \mid 1) * 1}$

11. Binary no. starts and ends with 1

    $\boldsymbol{Strings: 11, 101, 1001, 10101, \ldots}$ $\boldsymbol{R.E. = 1\,(0 \mid 1) * 1}$

12. String starts and ends with same character

    $\boldsymbol{Strings: 00, 101, aba, baab \ldots}$ $\boldsymbol{R.E. = 1\,(0 \mid 1) * 1 \mid 0\,(0 \mid 1) * 0}$

    $\boldsymbol{a\,(a \mid b)^{*}\, a \mid b\,(a \mid b)^{*}\, b}$

Regular expression examples

13. All string of a and b starting with a

    $Strings: a, ab, aab, abb...$ $R.E. = a(a \mid b)^*$

14. String of 0 and 1 ends with 00

    $Strings: 00, 100, 000, 1000, 1100...$ $R.E. = (0 \mid 1) * 00$

15. String ends with abb

    $Strings: abb, babb, ababb...$ $R.E. = (a \mid b) * abb$

16. String starts with 1 and ends with 0

    $Strings: 10, 100, 110, 1000, 1100...$ $R.E. = 1(0 \mid 1) * 0$

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

    $Strings: 000, 100, 1100, 1001...$ $R.E. = (0|1)(0|1)0(0 \mid 1) *$

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

    $Strings: bb, bab, aabb, abba...$ $R.E. = a^* \, b \, a^* \, b \, a^*$

Regular expression examples

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

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

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

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

21. String should contain at least three 1

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

22. String should contain exactly two 1

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

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

    $Strings: 0, 1, 11, 01, 111, 010, 100\ldots.$ $R.E. = (0|1) \mid (0|1)(0|1) \mid (0|1)(0|1)(0|1)$

24. No. of zero should be multiple of 3

    $Strings: 000, 010101, 110100, 000000, 100010010\ldots.$ $R.E. = (1^*01^*01^*01^*)^*$

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

    $\boldsymbol{Strings: 10, 1100, 1000, 100100....}$ $\boldsymbol{R.E. = 1(0|1)((0|1)(0|1))^*}$

31. All string begins or ends with 00 or 11

    $\boldsymbol{Strings:\ 00101, 10100, 110, 01011 \ldots}$ $\boldsymbol{R.E. = (00|11)(0\ |\ 1) * |\ (0|1)^*\ (00|11)}$

Regular expression examples

32. Language of all string containing both 11 and 00 as substring

$\boldsymbol{Strings}$: $\mathbf{0011, 1100, 100110, 010011} \dots$

$\boldsymbol{R.E.} = ((\mathbf{0}|\mathbf{1})^*\mathbf{00}(\mathbf{0}|\mathbf{1})^*\mathbf{11}(\mathbf{0}|\mathbf{1})^*) \;\; | \;\; ((\mathbf{0}|\mathbf{1})^*\mathbf{11}(\mathbf{0}|\mathbf{1})^*\mathbf{00}(\mathbf{0}|\mathbf{1})^*)$

33. String ending with 1 and not contain 00

$\boldsymbol{Strings}$: $\mathbf{011, 1101, 1011} \dots.$

$\boldsymbol{R.E.} = (\mathbf{1}|\mathbf{01})^+$

34. Language of identifier

$\boldsymbol{Strings}$: $\boldsymbol{area, i, redious, grade1} \dots.$

$\boldsymbol{R.E.} = (\_ + \boldsymbol{L})(\_ + \boldsymbol{L} + \boldsymbol{D})^*$

$\boldsymbol{where\ L\ is\ Letter}$ & $\mathbf{D\ is\ digit}$

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:

  $d_1 \rightarrow r_1$
  $d_2 \rightarrow r_2$

  ……

  $d_n \rightarrow rn$

  Where d_i is a **distinct name** & r_i 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)*

Regular definition example

- Example: Unsigned Pascal numbers

  3

  5280

  39.37

  6.336E4

  1.894E-4

  2.56E+7

Regular Definition

digit → 0|1|…..|9

digits → digit digit*

optional_fraction → .digits | ϵ

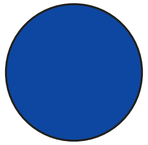
optional_exponent → (E(+|-|ϵ)digits)|ϵ

num → digits optional_fraction optional_exponent

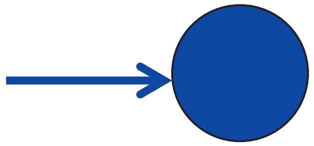
Transition Diagram

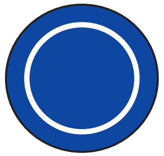
Transition Diagram

- A stylized flowchart is called transition diagram.

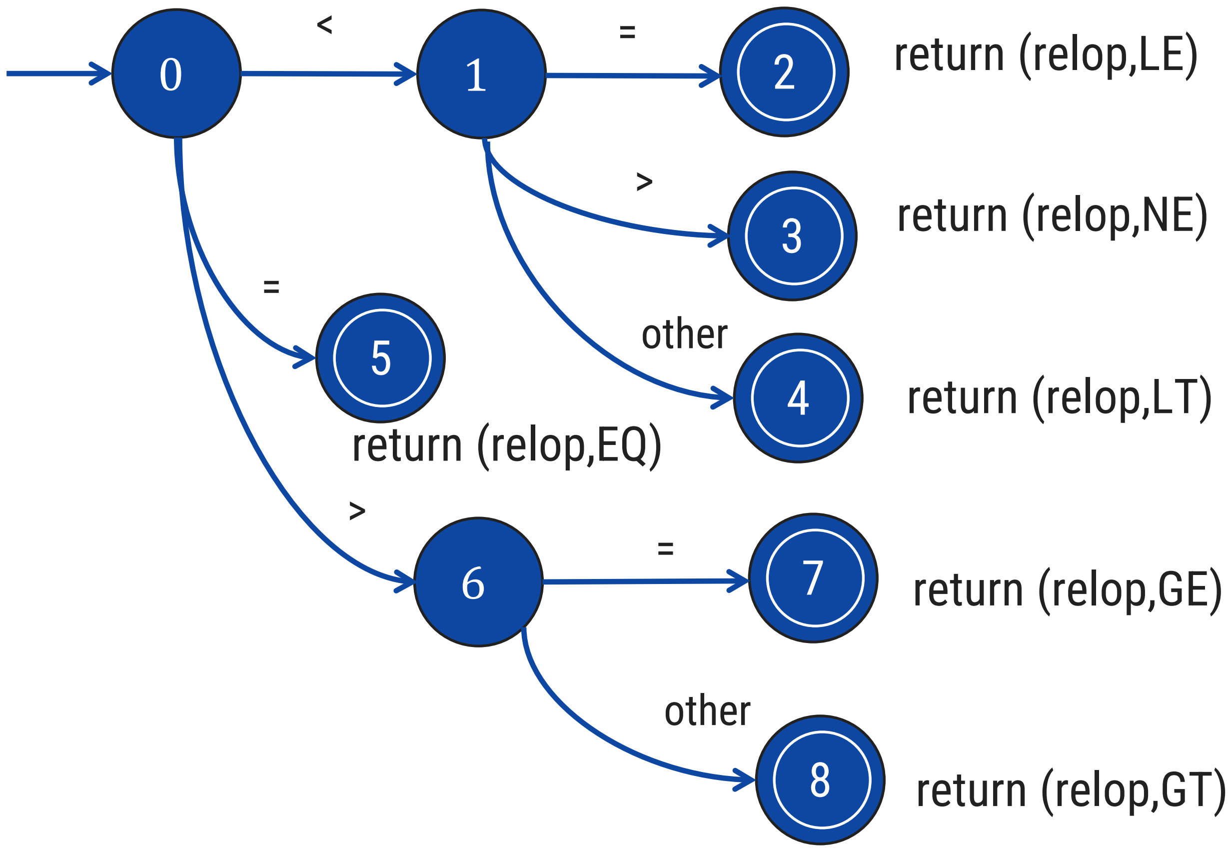

is a state


is a transition

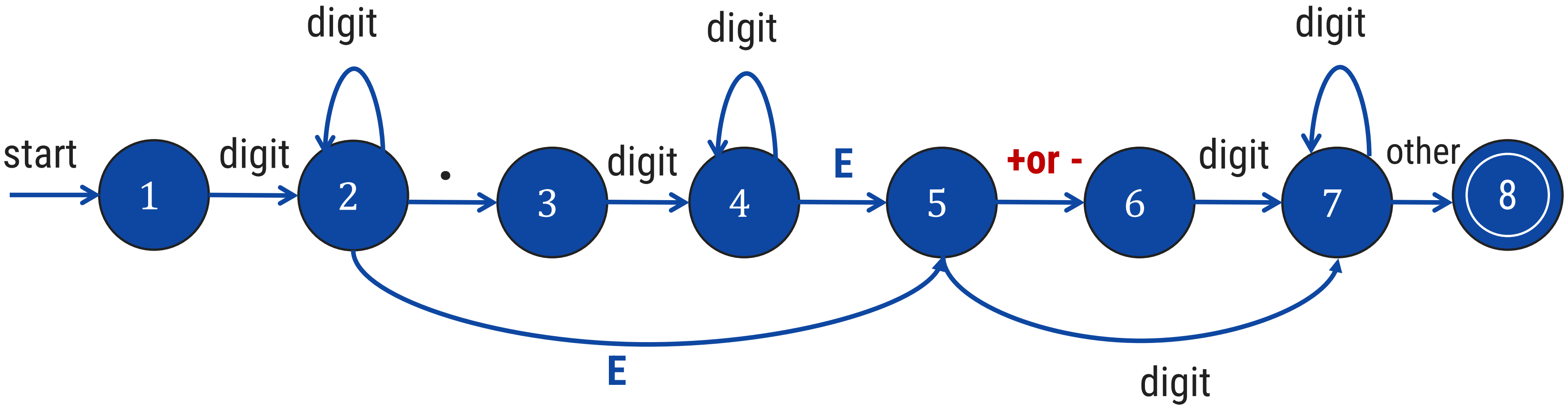

is a start state


is a final state

Transition Diagram : Relational operator



Transition diagram : Unsigned number



3

5280

39.37

1.894 **E** **-** 4

2.56 **E** **+** 7

45 **E** **+** 6

96 **E** 2

Finite Automata

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 $\mathbf{F}$

Finite Automata

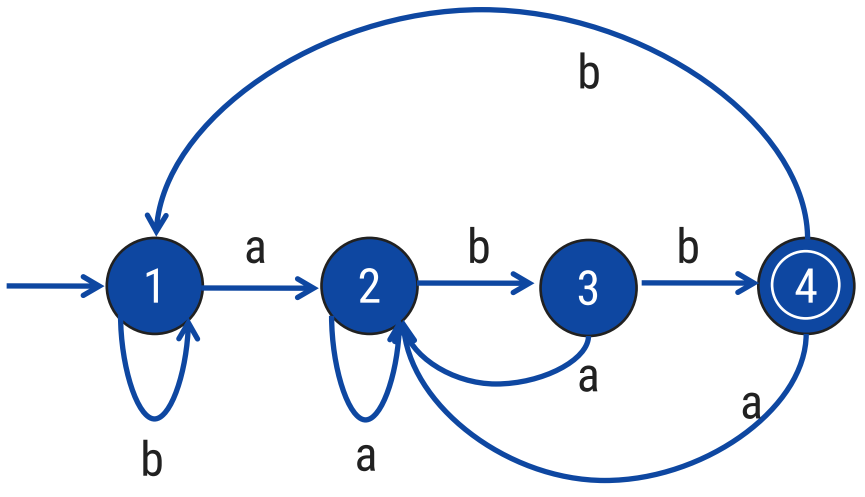
- A finite automaton, or finite state machine is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$ where
  - Q is finite set of states;
  - Σ is finite alphabet of *input symbols*;
  - $q_0 \epsilon Q$ (*initial* state);
  - $A \subseteq Q$ (the set of *accepting* states);
  - δ is a function from $Q \times \Sigma\ to\ Q$ (the *transition* function).
- For any element q of Q and any symbol $a \epsilon \Sigma$, we interpret $\delta(q, a)$ as the state to which the FA moves, if it is in state q and receives the input a.

Types of finite automata

- Types of finite automata are:

DFA

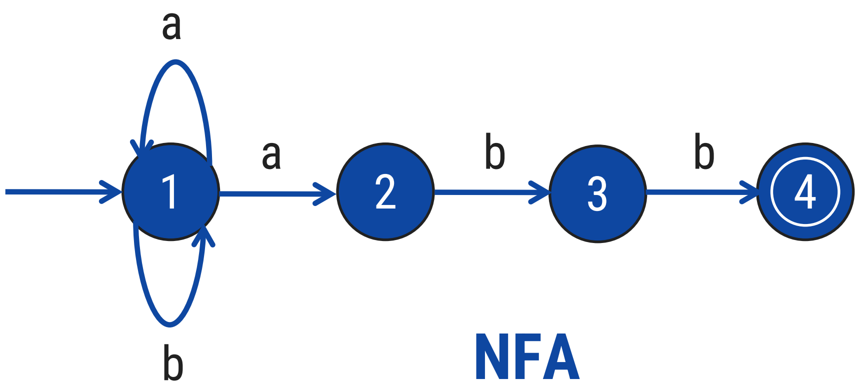
- Deterministic finite automata (DFA): have for each state exactly one edge leaving out for each symbol.



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.

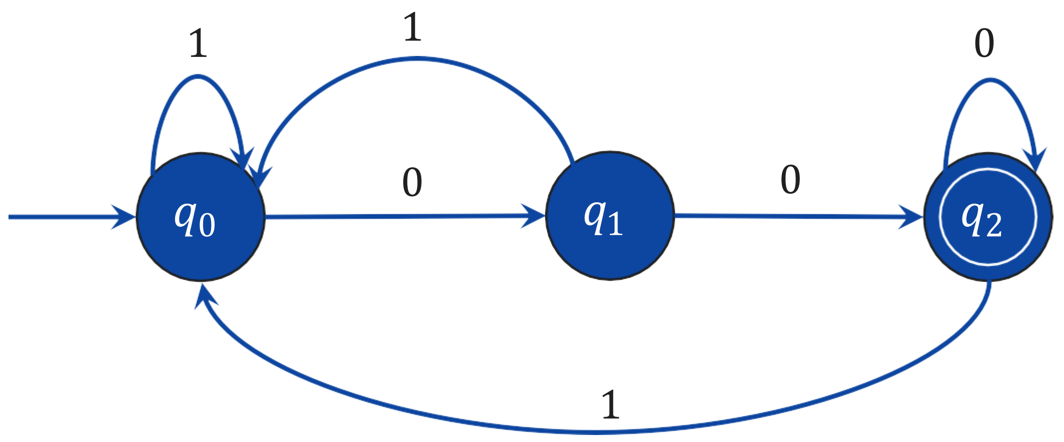


NFA

Example: Finite Automata

- $M = (Q, \Sigma, q_0, A, \delta)$
  - $Q = \{q_0, q_1, q_2\}$
  - $\Sigma = \{0,1\}$
  - $q_0 = q_0$
  - $A = \{q_2\}$
  - δ is defined as

δ	Input	
State	**0**	**1**
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0

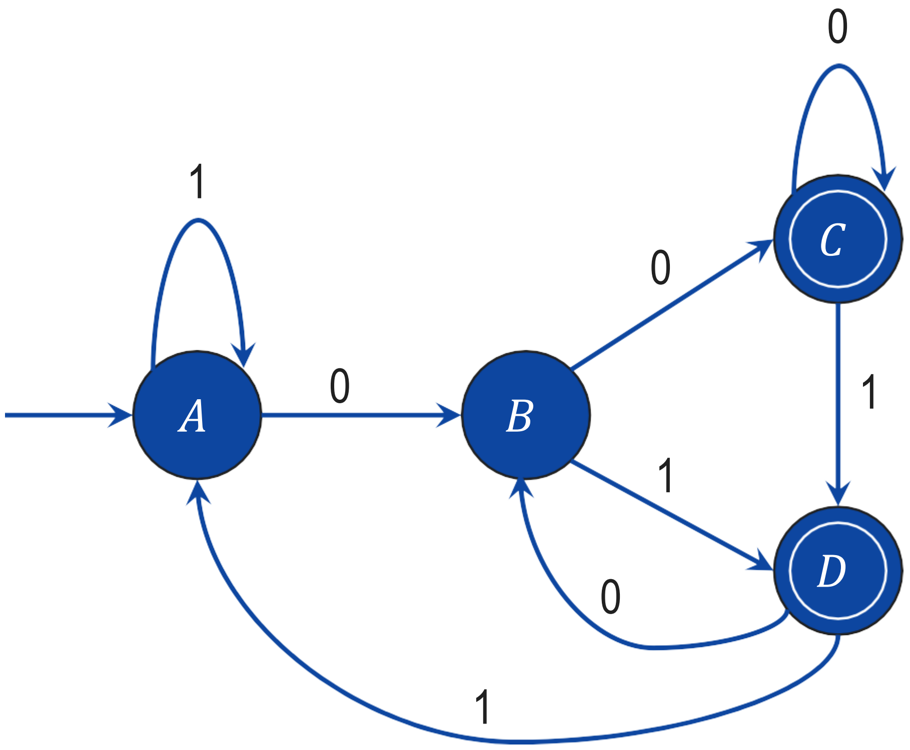


Applications of FA

- Lexical analysis phase of a compiler.
- Design of digital circuit.
- String matching.
- Communication Protocol for information exchange.

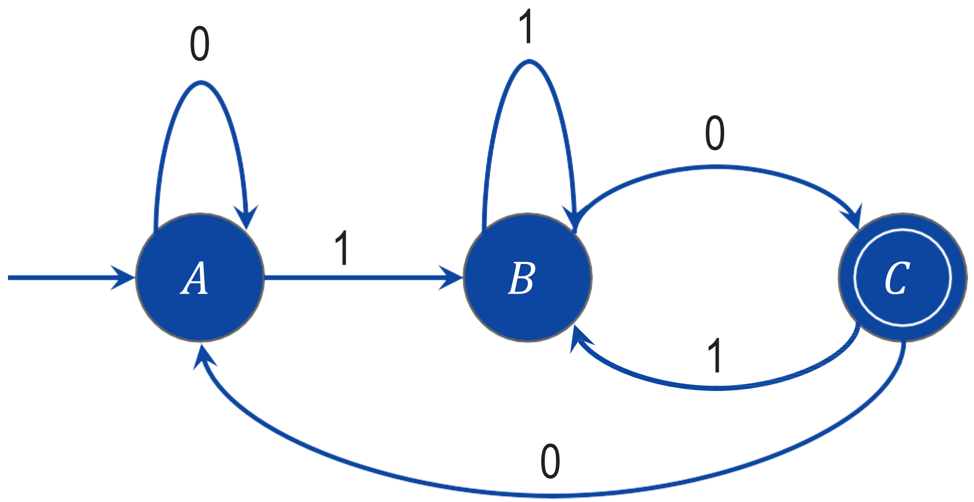
FA Examples

- The string with next to last symbol as 0.



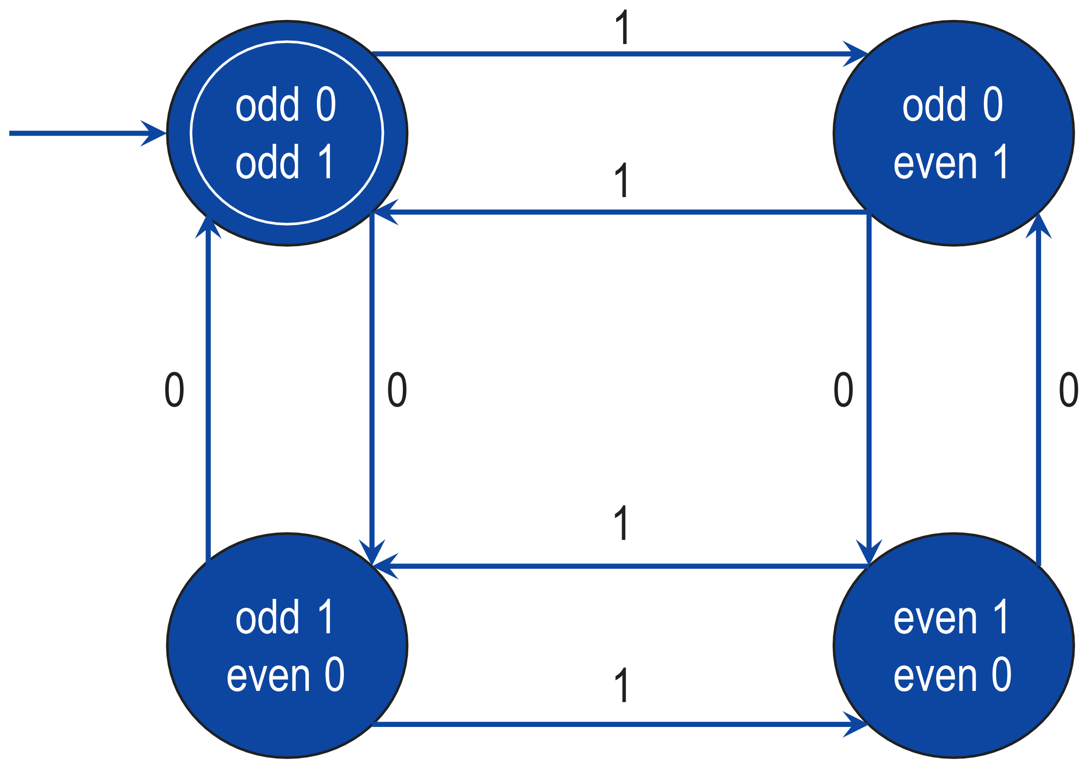
FA Examples

- The strings ending with 10.



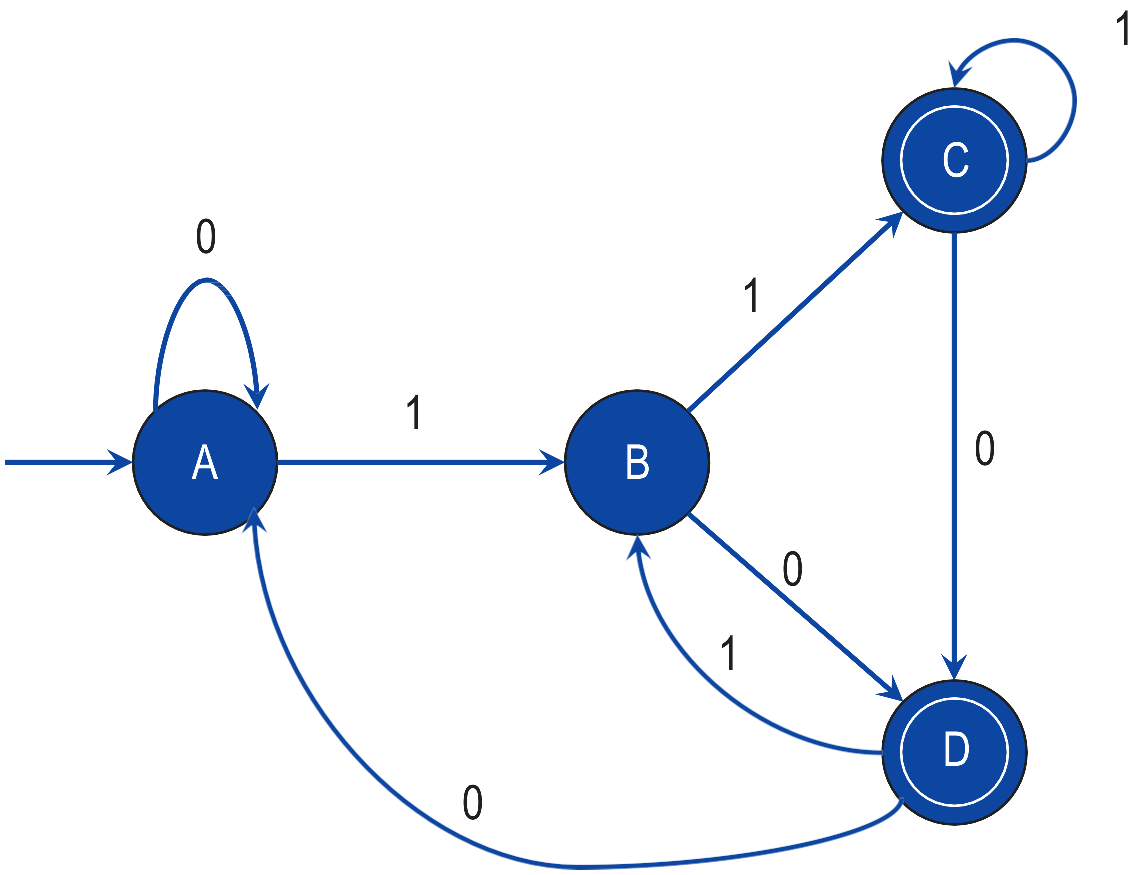
FA Examples

- The string with number of 0's and number of 1's are odd



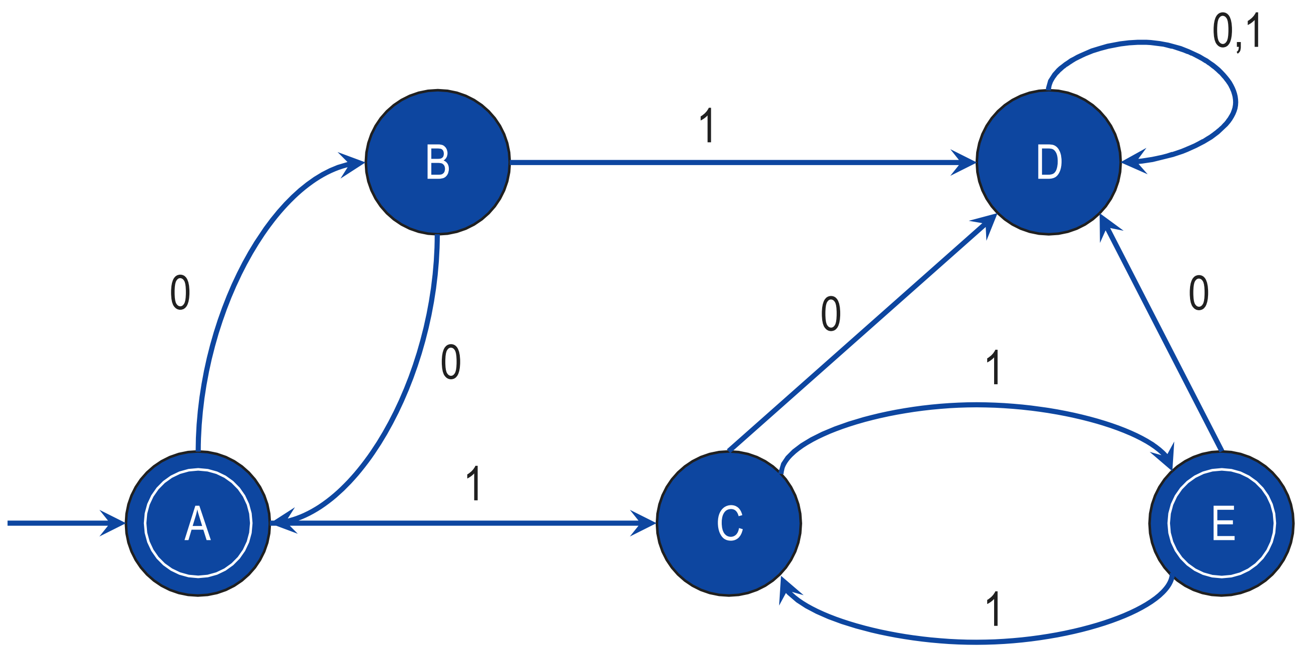
FA Examples

- The string ending in 10 or 11



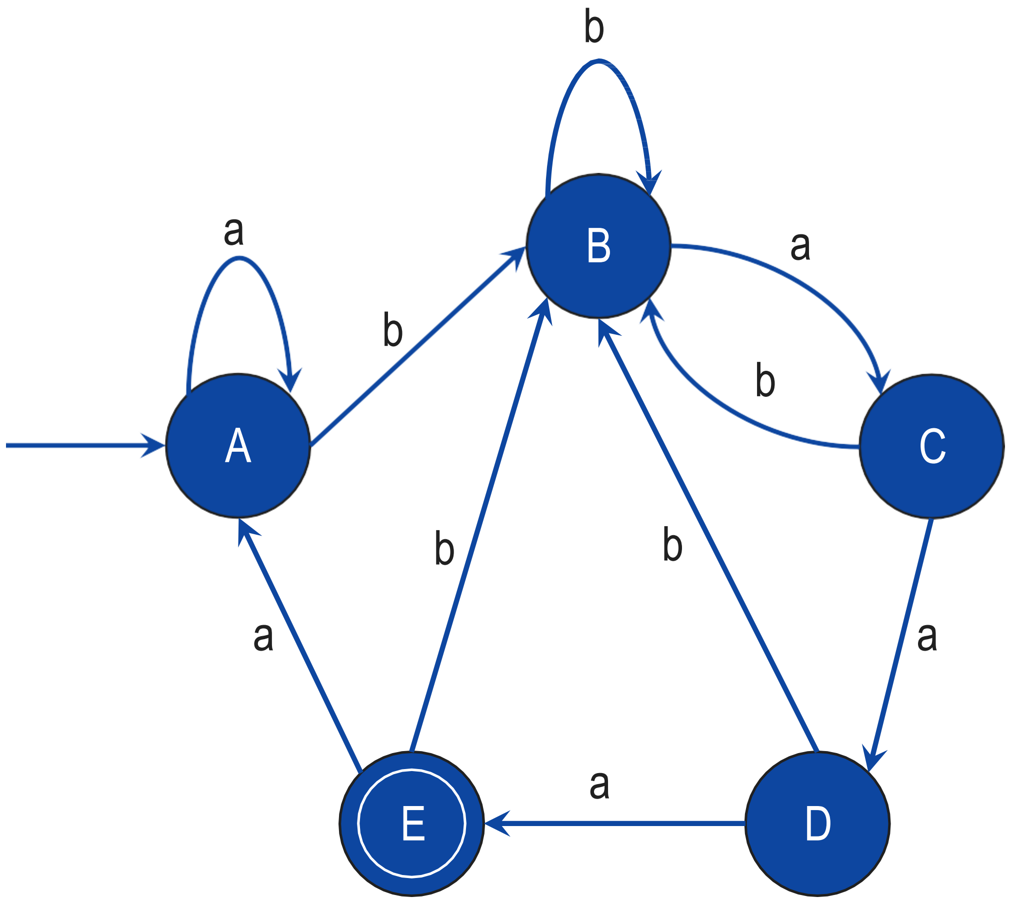
FA Examples

- The string corresponding to Regular expression {00}*{11}*



FA Examples

- (a+b)*baaa



Extended Transition Function δ^* for FA

- Let $M = (Q, \Sigma, q_0, A, \delta)$ be an Finite Automata. We define the function

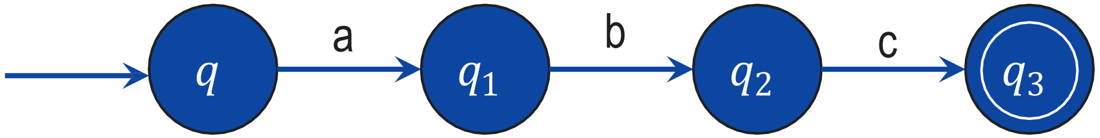
$$\delta^*: Q \times \Sigma^* \to Q$$

  as follows:
  1. For any $q \in Q, \delta^*(q, \Lambda) = q$
  2. For any $q \in Q, y \in \Sigma^*$, and $a \in \Sigma$,

$$\boldsymbol{\delta^*(q, ya) = \delta(\delta^*(q, y), a)}$$

δ^* Example

- Consider FA



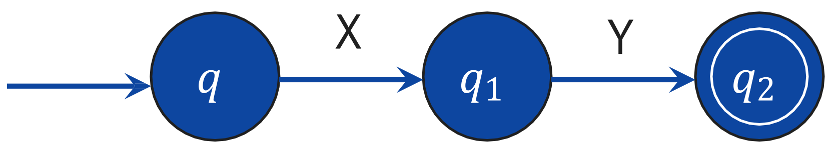
$\delta^*(q,\Lambda)$	$= q$
$\delta^*(q, ya)$	$= \delta(\delta^*(q, y), a)$

- Calculate $\delta^*(q, abc)$

$$
\begin{aligned}
\delta^*(q, abc) &= \delta(\delta^*(q, ab), c) \\
&= \delta(\delta(\delta^*(q, a), b), c) \\
&= \delta(\delta(\delta^*(q, \Lambda a), b), c) \\
&= \delta(\delta(\delta(\underline{\boldsymbol{\delta^*(q, \Lambda)}}, a), b), c) \\
&= \delta(\delta(\underline{\delta(\boldsymbol{q}, a)}, b), c) \\
&= \delta(\underline{\delta(q_1, b)}, c) \\
&= \underline{\delta(q_2, c)} \\
&= q_3
\end{aligned}
$$

Exercise

Consider following FA

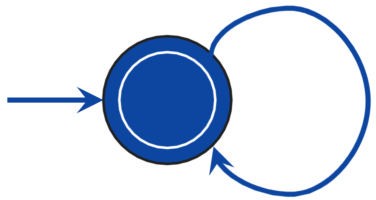


$\delta^*(q,\Lambda)$	$= q$
$\delta^*(q, ya)$	$= \delta(\delta^*(q, y), a)$

1. calculate $\delta^*(q, XY)$

Acceptance by FA

- Let $M = (Q, \Sigma, q_0, A, \delta)$ be an FA. A string $x \in \Sigma^*$ is accepted by M if $\delta^*(q_0, x) \in A$. If string is not accepted, we can say it is rejected by M. The language accepted by M, or the language recognized by M, is the set
$$L(M) = \{x \in \Sigma^* |\ x \text{ is accepted by } M\}$$
- If L is any language over Σ, L is accepted or recognized by M if and only if $L = L(M)$.



Union, Intersection & Complement of Languages

- Suppose $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ accepts languages L_1 and L_2, respectively. Let M be an FA defined by $M = (Q, \Sigma, q_0, A, \delta)$, where

$$Q = Q_1 \times Q_2$$
$$q_0 = (q_1, q_2)$$

and the transition function δ is defined by the formula

$$\delta\big((p, q), a\big) = \big(\delta_1(p, a), \delta_2(q, a)\big)$$

for any $p \in Q_1$ and $q \in Q_2$ and $a \in \Sigma$ then

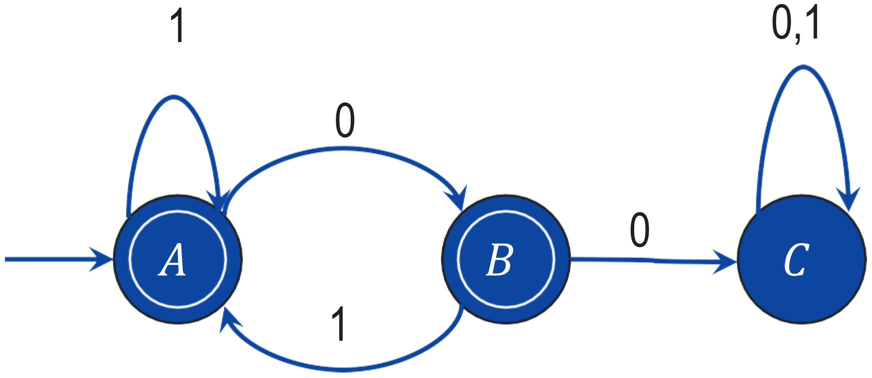
1. if $A = \{(p, q) \mid \boldsymbol{p \in A_1 \text{ or } q \in A_2}\}$, M accepts the language $\boldsymbol{L_1 \cup L_2}$;
2. if $A = \{(p, q) \mid \boldsymbol{p \in A_1 \text{ and } q \in A_2}\}$, M accepts the language $\boldsymbol{L_1 \cap L_2}$;
3. if $A = \{(p, q) \mid \boldsymbol{p \in A_1 \text{ and } q \notin A_2}\}$, M accepts the language $\boldsymbol{L_1 - L_2}$;

Example

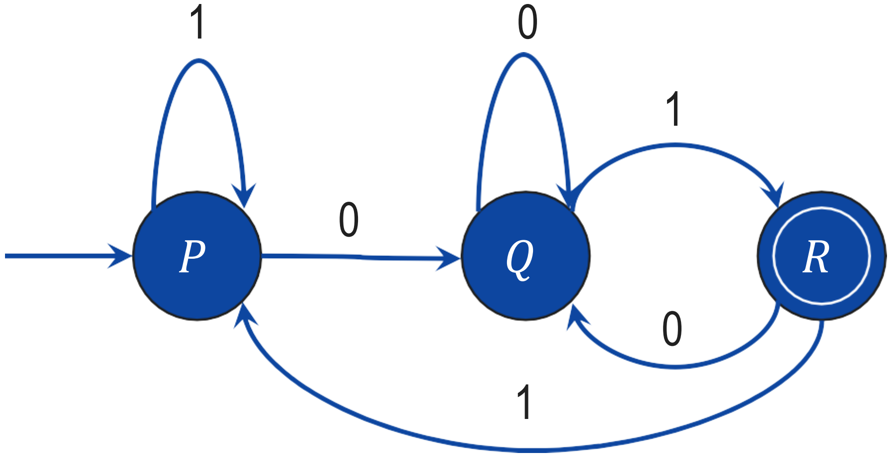
Draw Finite Automata for following languages:

1. L1={x/x 00 is not substring of x, x ∈ {0,1}*}
2. L2={x/x ends with 01, x ∈ {0,1}*}

Draw FA for $L_1 \cup L_2$, $L_1 \cap L_2$ and $L_1 - L_2$



M_1



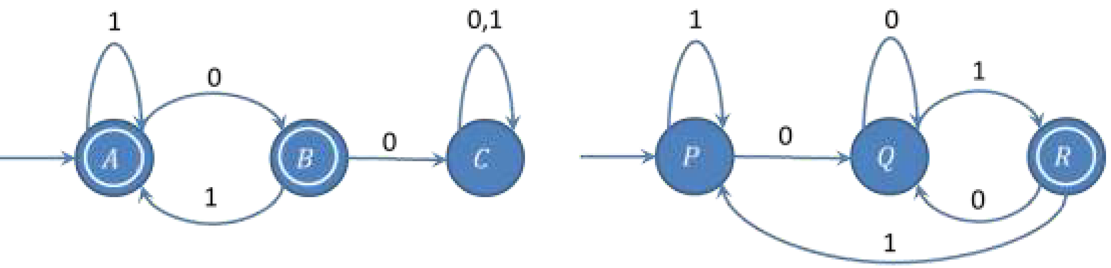
M_2

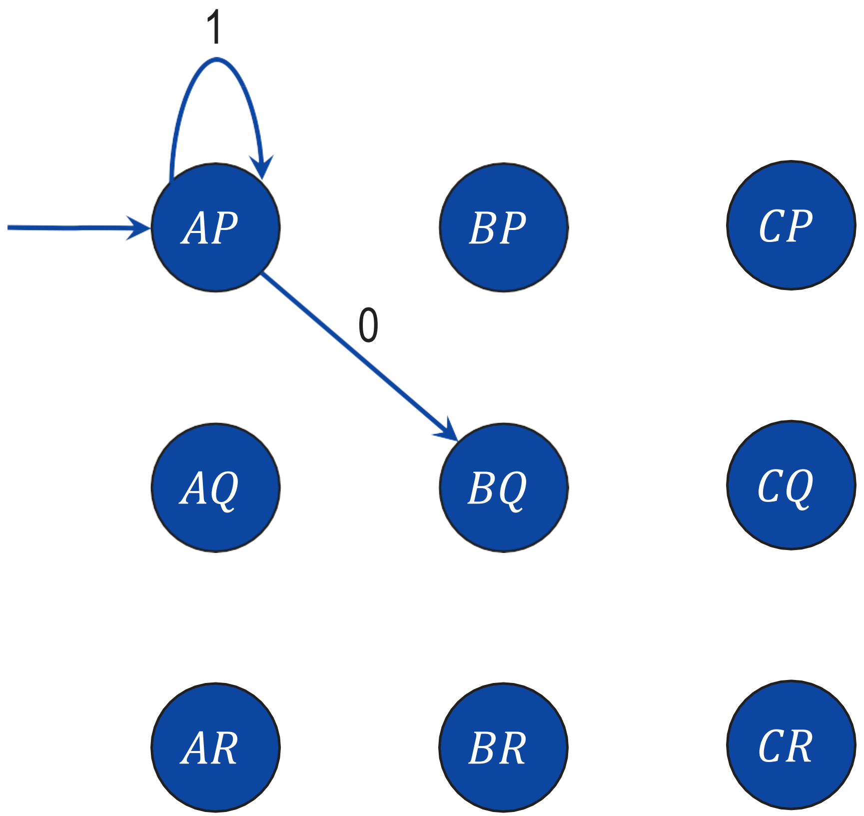
Computation for $L_1 \cup L_2$, $L_1 \cap L_2$ and $L_1 - L_2$

- Here $Q_1 = \{A, B, C\}$ and $Q_2 = \{P, Q, R\}$
- So, $Q = Q_1 \times Q_2 = \{AP,\ AQ,\ AR,\ BP,\ BQ,\ BR,\ CP,\ CQ,\ CR\}$
- $q_0 = (q_1, q_2) = (A, P)$
- Computing Transition Function δ

$$\delta\big((A,P),0\big) = (\delta(A,0), \delta(P,0)) \\ = BQ$$

$$\delta\big((A,P),1\big) = (\delta(A,1), \delta(P,1)) \\ = AP$$





BQ

Computation for $L_1 \cup L_2$, $L_1 \cap L_2$ and $L_1 - L_2$

$$\delta\big((B,Q),0\big) = (\delta(B,0),\delta(Q,0)) = CQ$$

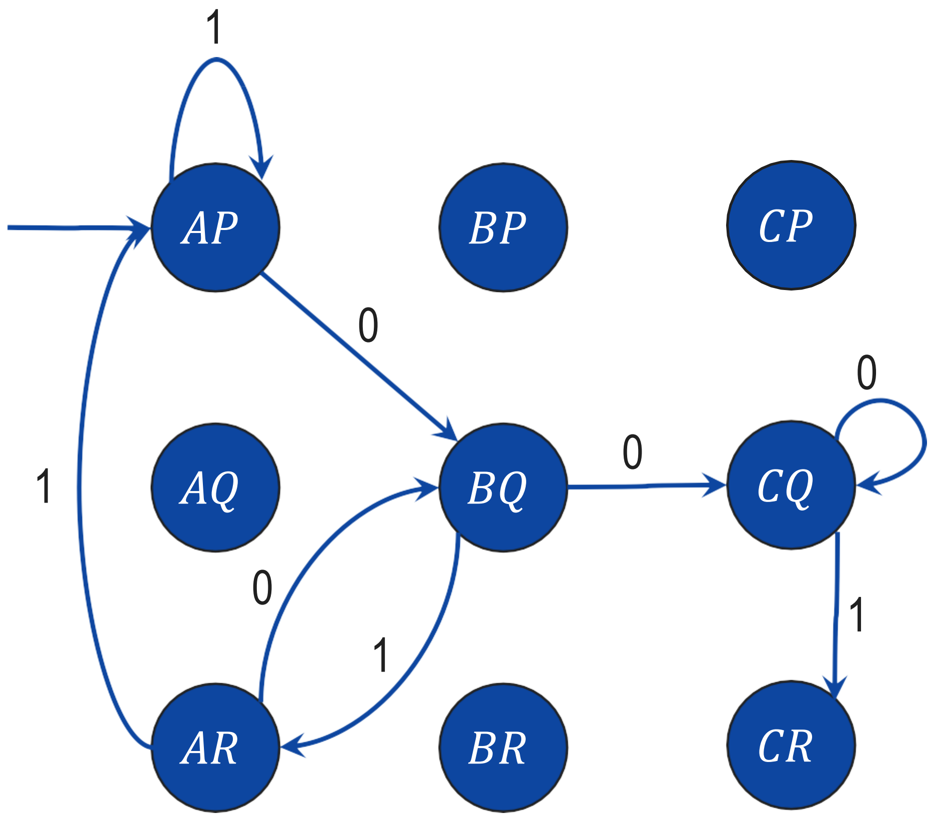
$$\delta\big((B,Q),1\big) = (\delta(B,1),\delta(Q,1)) = AR$$

$$\delta\big((C,Q),0\big) = (\delta(C,0),\delta(Q,0)) = CQ$$

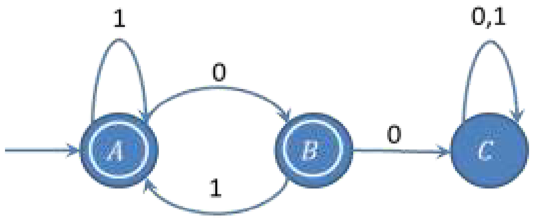
$$\delta\big((C,Q),1\big) = (\delta(C,1),\delta(Q,1)) = CR$$

$$\delta\big((A,R),0\big) = (\delta(A,0),\delta(R,0)) = BQ$$

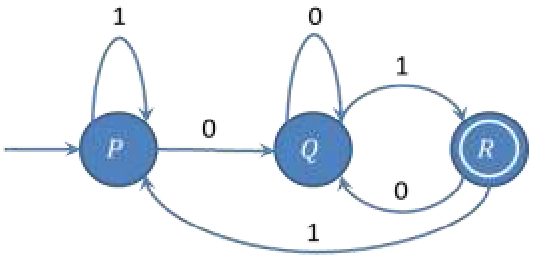
$$\delta\big((A,R),1\big) = (\delta(A,1),\delta(R,1)) = AP$$



~~BQ~~ ~~CQ~~ ~~AR~~ CR



M_1



M_2

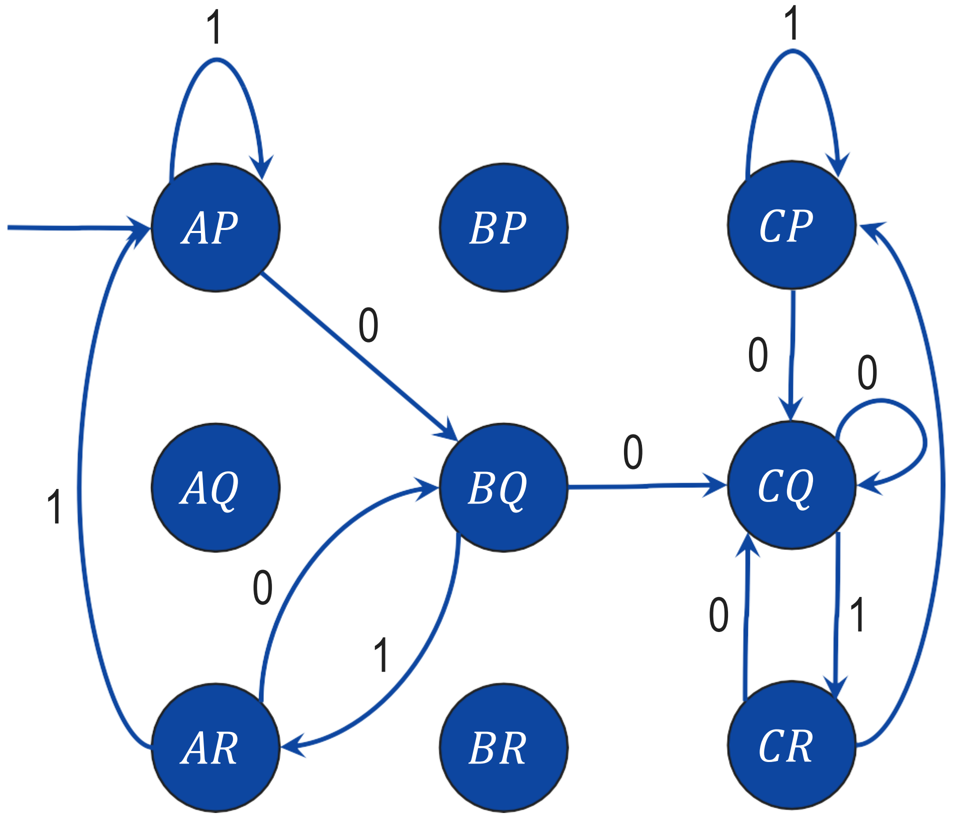
Computation for $L_1 \cup L_2$, $L_1 \cap L_2$ and $L_1 - L_2$

$$\delta\big((C,R),0\big) = (\delta(C,0),\delta(R,0))$$
$$= CQ$$

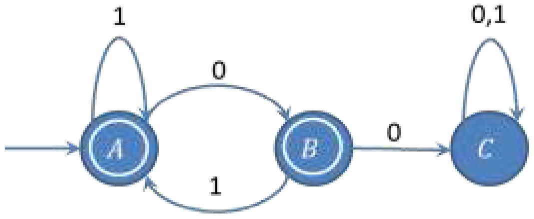
$$\delta\big((C,R),1\big) = (\delta(C,1),\delta(R,1))$$
$$= CP$$

$$\delta\big((C,P),0\big) = (\delta(C,0),\delta(P,0))$$
$$= CQ$$

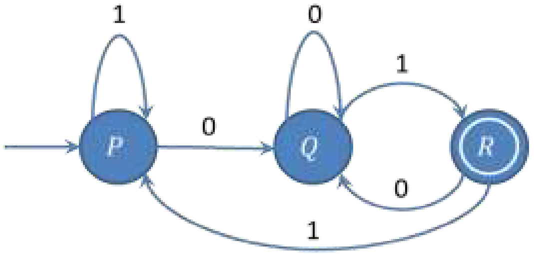
$$\delta\big((C,P),1\big) = (\delta(C,1),\delta(P,1))$$
$$= CP$$



~~BQ~~ ~~CQ~~ ~~AR~~ ~~CR~~ ~~CP~~

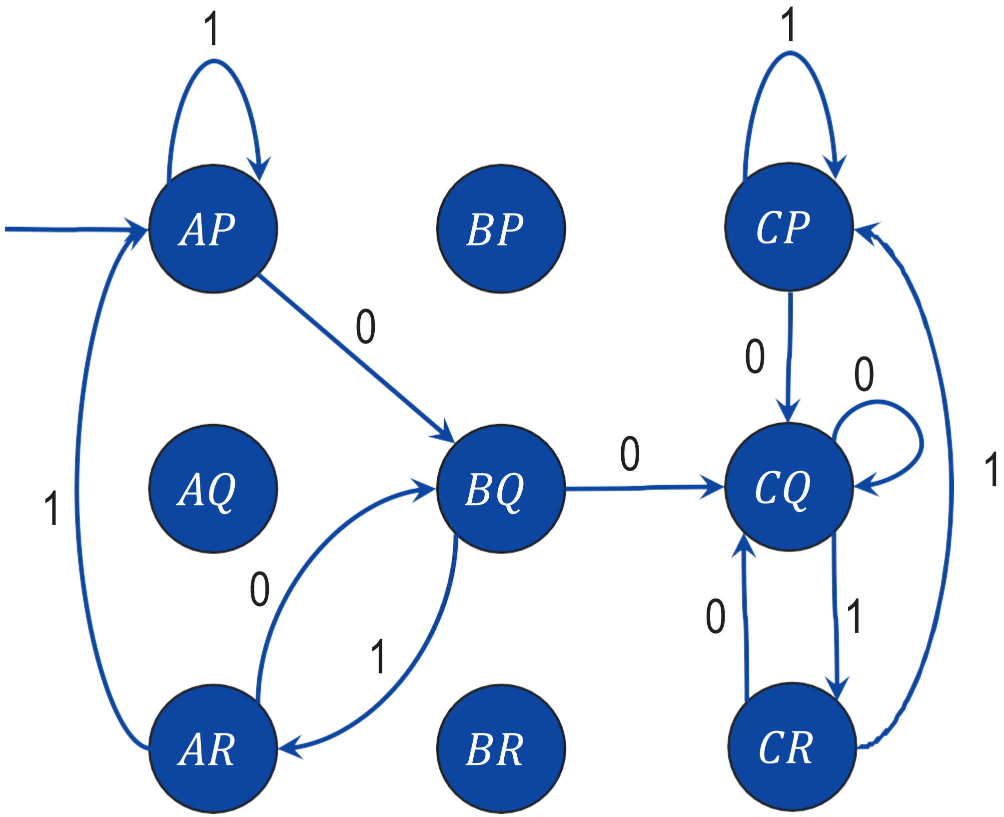


M_1



M_2

Removing Unconnected States

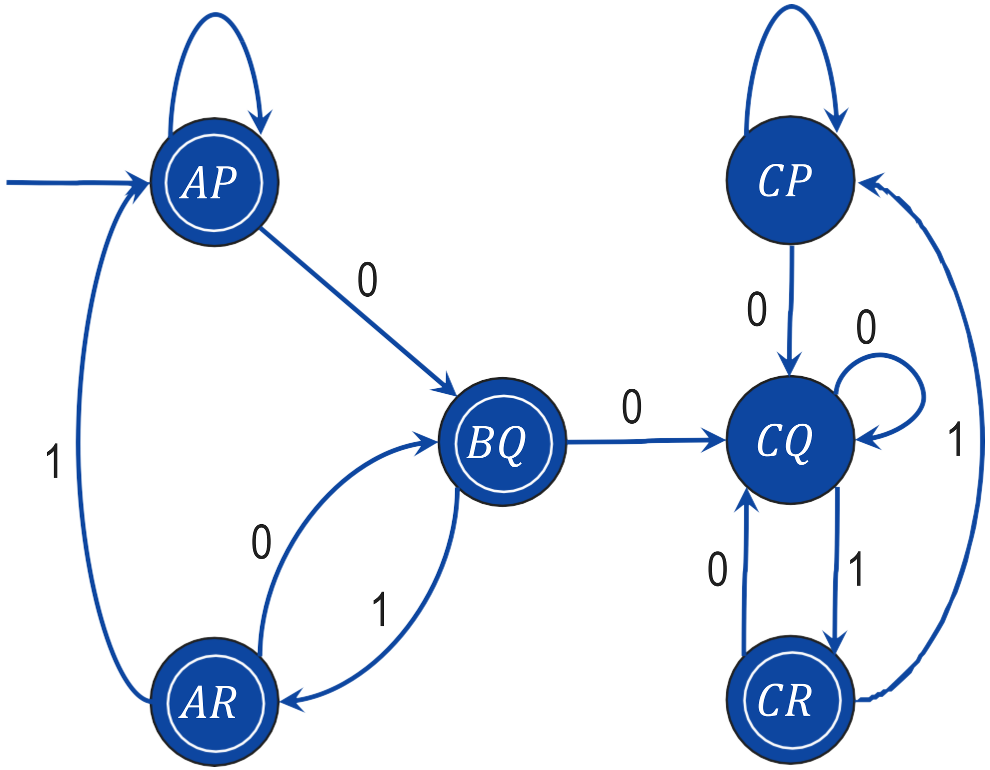


$L_1 \cup L_2$

$A_1 = \{A, B\}$

$A_2 = \{R\}$

As per theorem stated earlier, the states which consists A or B or R will be Accepting states in the resultant FA.



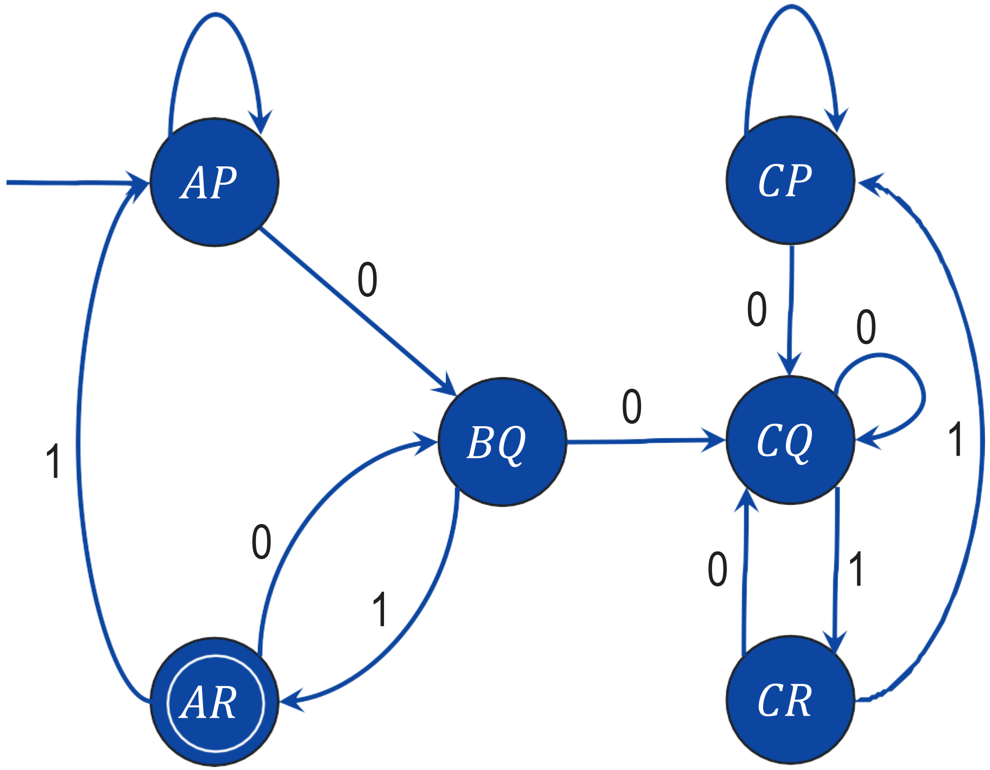
Accepting States $A = \{AP, AR, BQ, CR\}$

$L_1 \cap L_2$

$A_1 = \{A, B\}$

$A_2 = \{R\}$

Therefore, as per theorem stated earlier, the states which consists (A and R) and (B and R) will be Accepting states in the resultant FA.



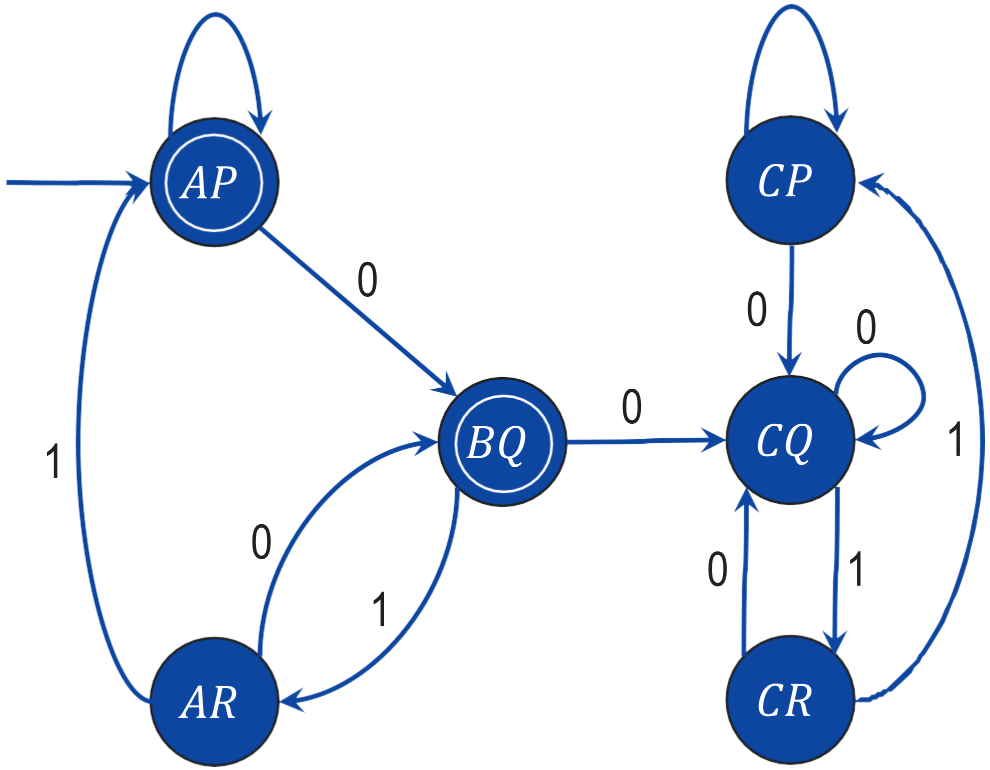
Accepting States $A = \{AR\}$

$L_1 - L_2$

$A_1 = \{A, B\}$

$A_2 = \{R\}$

Therefore, as per theorem stated earlier, the states which consists A or B but should not contain R with them will be Accepting states in the resultant FA.



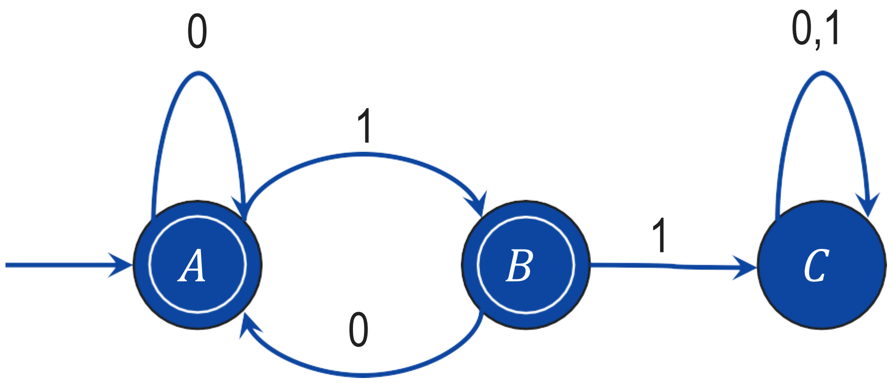
Accepting States $A = \{AP, BQ\}$

Exercise

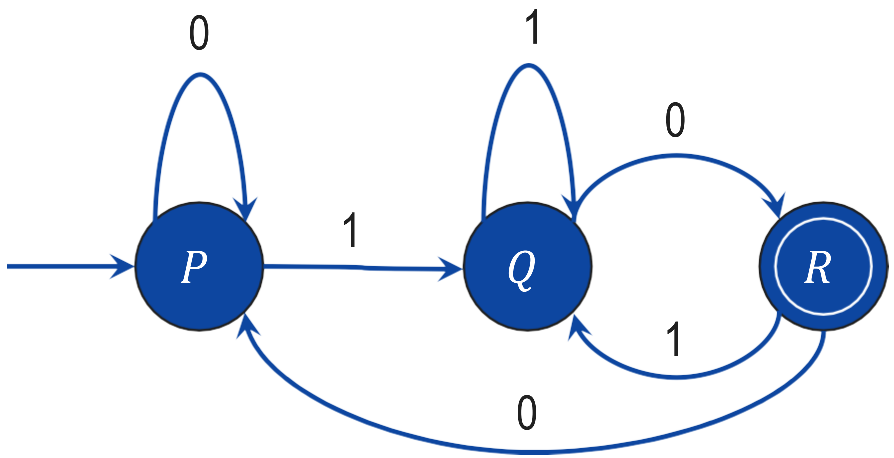
Draw Finite Automata for following languages:

1. L1={x/x 11 is not substring of x, x ∈ {0,1}*}
2. L2={x/x ends with 10, x ∈ {0,1}*}

Draw FA for $L_1 \cup L_2$, $L_1 \cap L_2$ and $L_1 - L_2$.



$\boldsymbol{M_1}$



$\boldsymbol{M_2}$

Nondeterministic Finite Automata (NFA)

- A nondeterministic finite automaton is a 5-tuple $M = (Q, \Sigma, q_0, A, \delta)$ where Q and Σ are nonempty finite sets, $q_0 \epsilon Q$, $A \subseteq Q$ and

$$\delta : Q \times \Sigma \rightarrow 2^Q$$

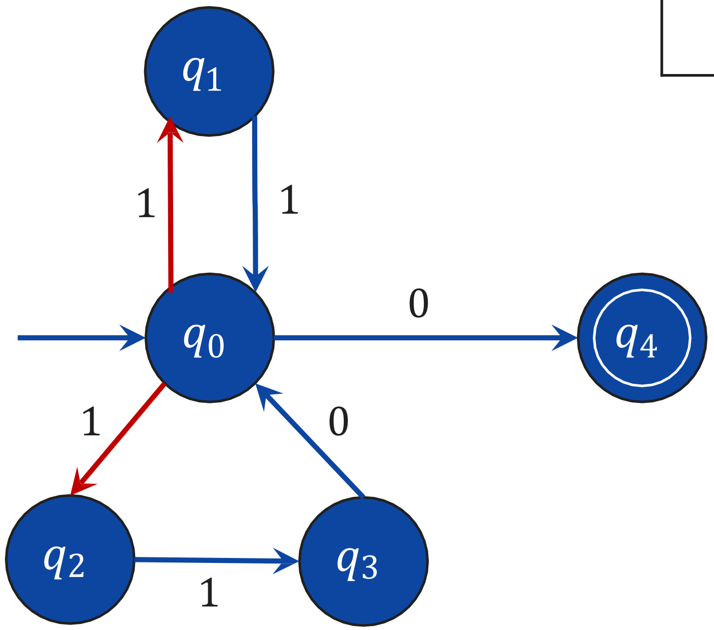
  Q is the set of states, Σ is the alphabet, q_0 is the initial state and A is the set of accepting states.

Example of NFA for $\{11,110\}^*\{0\}$

$M = (Q, \Sigma, q_0, A, \delta)$

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0,1\}$
- $q_0 = q_0$
- $A = \{q_4\}$
- δ is defined as

δ	Input	
State	0	1
q_0	$\{q_4\}$	$\{q_1, q_2\}$
q_1	$\emptyset$	$\{q_0\}$
q_2	$\emptyset$	$\{q_3\}$
q_3	$\{q_0\}$	$\emptyset$
q_4	$\emptyset$	$\emptyset$



Nonrecursive Definition of δ^* for NFA

- For an NFA $M = (Q, \Sigma, q_0, A, \delta)$, and any $p \in Q, \delta^*(p, \Lambda) = \{p\}$. For any $p \in Q$ and any $x = a_1 a_2 \dots a_n \in \Sigma^*$ $(with\, n \geq 1)$, $\delta^*(p, x)$ is the set of all states q for which there is a sequence of states $p = p_0, p_1, \dots, p_{n-1}, p_n = q$ satisfying

$$p_i \in \delta(p_{i-1}, a_i) \; for\ each\ i\ with\ 1 \leq i \leq n$$

Recursive Definition of δ^* for NFA

- Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The function $\delta^*: Q \times \Sigma^* \to 2^Q$ is defined as follows.
  1. For any $q \in Q$, $\delta^*(q, \Lambda) = \{q\}$.
  2. For any $q \in Q$, $y \in \Sigma^*$, and $a \in \Sigma$,

$$\boldsymbol{\delta^*(q, ya) = \bigcup_{r \in \delta^*(q,y)} \delta(r, a)}$$

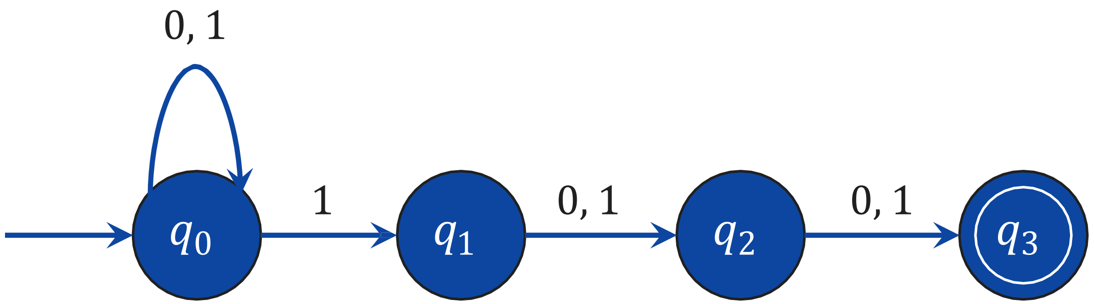
Acceptance by NFA

- Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The string $x \in \Sigma *$ is accepted by M if $\delta * (q0, x) \cap A \neq$ Ø. The language recognized, or accepted, by M is the set $L(M)$ of all string accepted by M. For any language $L \subseteq \Sigma *, L$ is recognized by $M\ if\ L = L(M)$.

Example: Recursive Definition of δ^* in NFA

- $M = (Q, \Sigma, q_0, A, \delta)$
  - $Q = \{q_0, q_1, q_2, q_3\}$
  - $\Sigma = \{0,1\}$
  - $q_0 = q_0$
  - $A = \{q_3\}$
  - δ is defined as

δ	Input	
State	**0**	**1**
q_0	$\{q_o\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\emptyset$	$\emptyset$



Example: Recursive Definition of δ^* in NFA

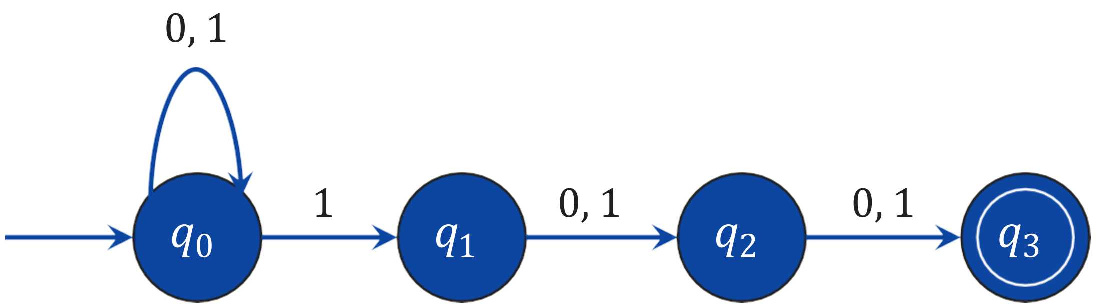
$$\delta^*(q_0, 0) = \bigcup_{r \in \boldsymbol{\delta^*(q_0, \Lambda)}} \delta(r, 0)$$

$$= \bigcup_{r \in \{\boldsymbol{q_0}\}} \delta(r, 0)$$

$$= \delta(q_0, 0)$$

$$= \{q_0\}$$

$\delta^*(q, \Lambda)$	$= q$
$\delta^*(q, ya)$	$= \bigcup_{r \in \delta^*(q, y)} \boldsymbol{\delta(r, a)}$



Example: Recursive Definition of δ^* in NFA

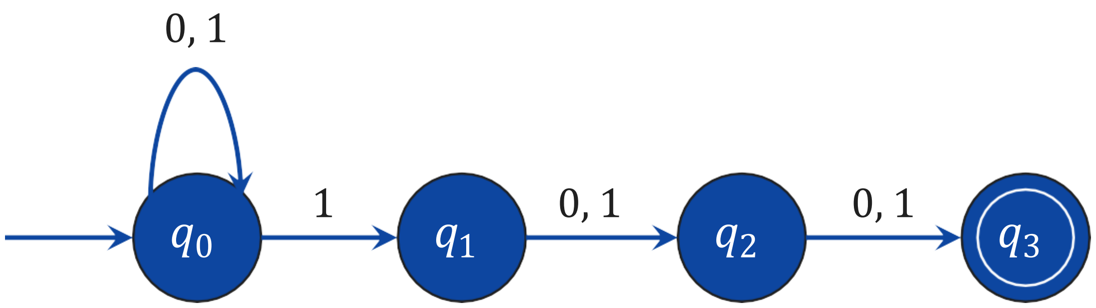
$$\delta^*(q_0, 1) = \bigcup_{r \in \boldsymbol{\delta^*(q_0, \Lambda)}} \delta(r, 1)$$

$$= \bigcup_{r \in \{\boldsymbol{q_0}\}} \delta(r, 1)$$

$$= \delta(q_0, 1)$$

$$= \{q_0, q_1\}$$

$\delta^*(q, \Lambda)$	$= q$
$\delta^*(q, ya)$	$= \bigcup_{\boldsymbol{r \in \delta^*(q, y)}} \boldsymbol{\delta(r, a)}$



Example: Recursive Definition of δ^* in NFA

$$\delta^*(q_0, 11) = \bigcup_{r \in \delta^*(q_0,1)} \delta(r, 1)$$

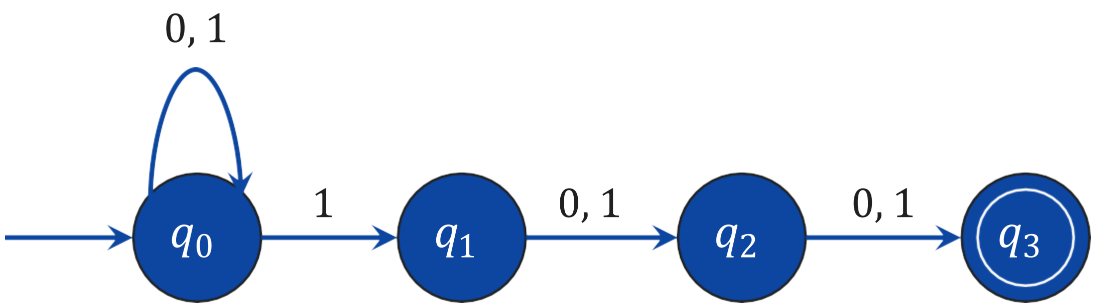
$$= \bigcup_{r \in \{q_0, q_1\}} \delta(r, 1)$$

$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$

$$= \{q_0, q_1\} \cup \{q_2\}$$

$$= \{q_0, q_1, q_2\}$$

$\delta^*(q,\Lambda)$	$= q$
$\delta^*(q, ya)$	$= \bigcup_{r \in \delta^*(q,y)} \boldsymbol{\delta(r, a)}$



Example: Recursive Definition of δ^* in NFA

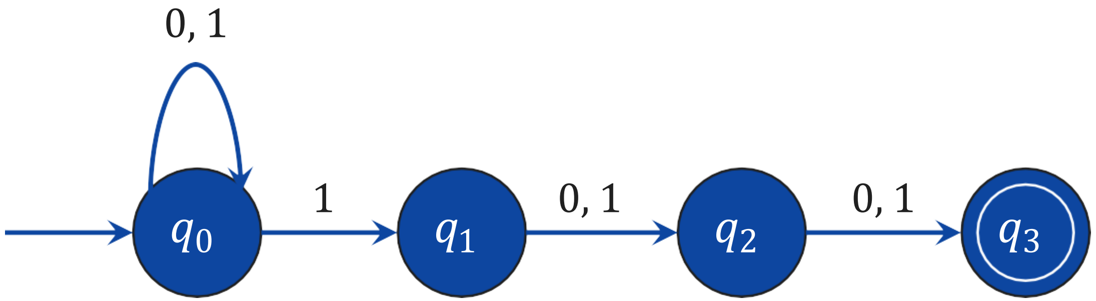
$$\delta^*(q_0, 01) = \bigcup_{r \in \boldsymbol{\delta^*(q_0, 0)}} \delta(r, 1)$$

$$= \bigcup_{r \in \{\boldsymbol{q_0}\}} \delta(r, 1)$$

$$= \delta(q_0, 1)$$

$$= \{q_0, q_1\}$$

$\delta^*(q, \Lambda)$	$= q$
$\delta^*(q, ya)$	$= \bigcup_{\boldsymbol{r \in \delta^*(q, y)}} \boldsymbol{\delta(r, a)}$



Example: Recursive Definition of δ^* in NFA

$$\delta^*(q_0, 111) = \bigcup_{r \in \delta^*(q_0, 11)} \delta(r, 1)$$

$$= \bigcup_{r \in \{q_0, q_1, q_2\}} \delta(r, 1)$$

$$= \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$$

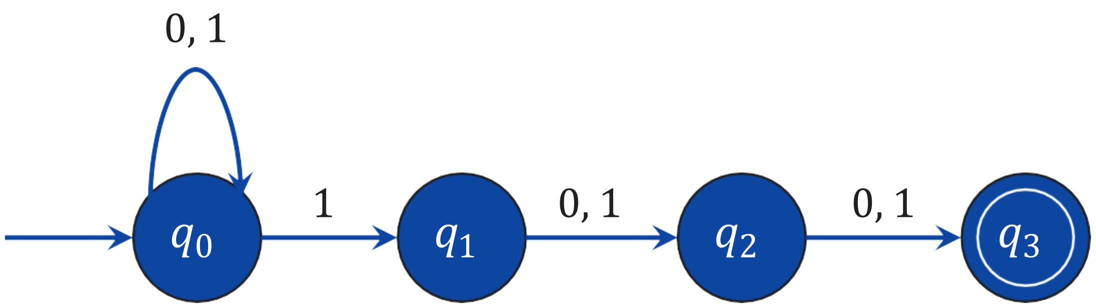
$$= \{q_0, q_1, q_2, q_3\}$$

$$\delta^*(q_0, 011) = \bigcup_{r \in \delta^*(q_0, 01)} \delta(r, 1)$$

$$= \delta(q_0, 1) \cup \delta(q_1, 1)$$

$$= \{q_0, q_1, q_2\}$$

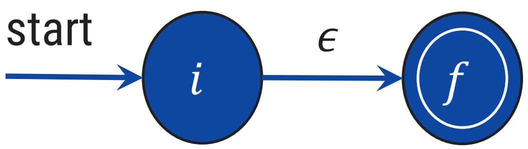
$\delta^*(q, \Lambda)$	$= q$
$\delta^*(q, ya)$	$= \bigcup_{r \in \delta^*(q, y)} \delta(r, a)$



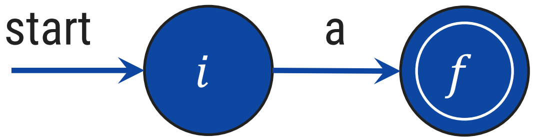
Regular expression to NFA using Thompson's rule

Regular expression to NFA using Thompson's rule

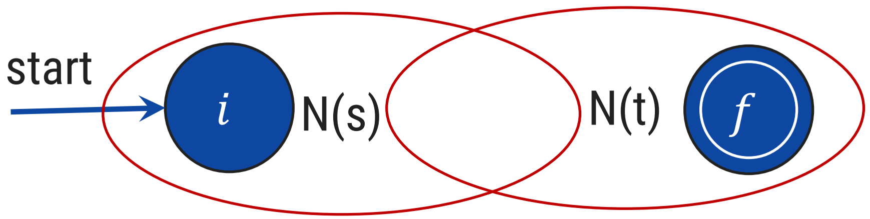
1. For ∈ , construct the NFA



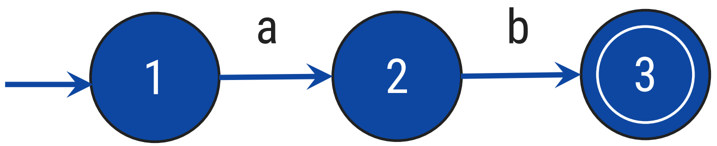
2. For a in Σ, construct the NFA



3. For regular expression st

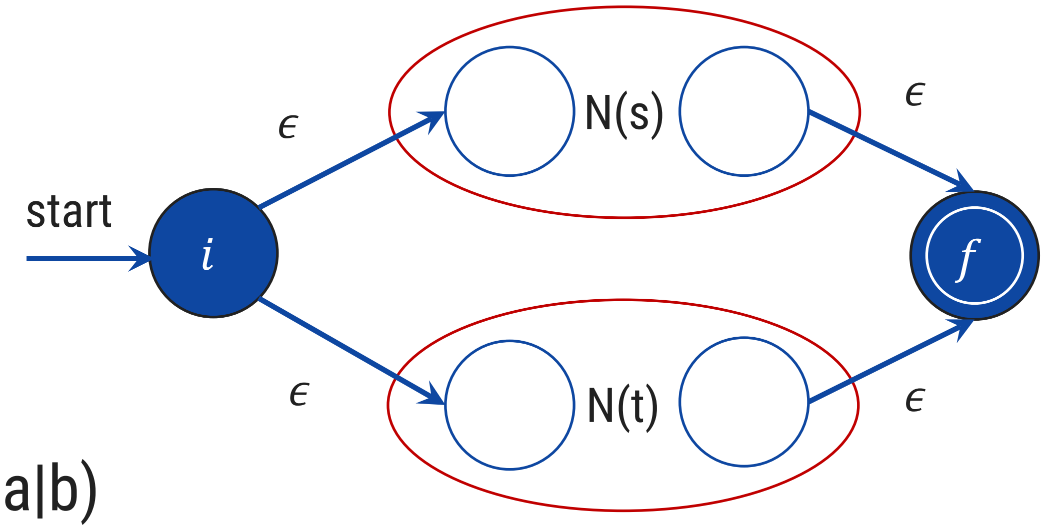


Ex: ab

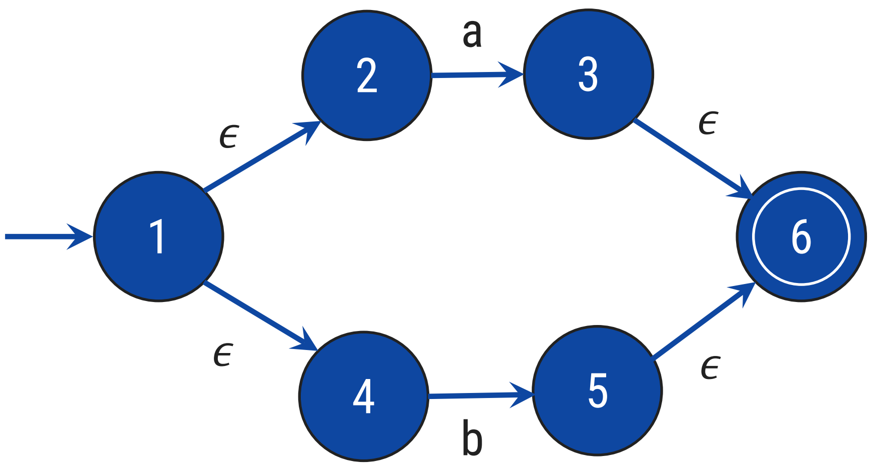


Regular expression to NFA using Thompson's rule

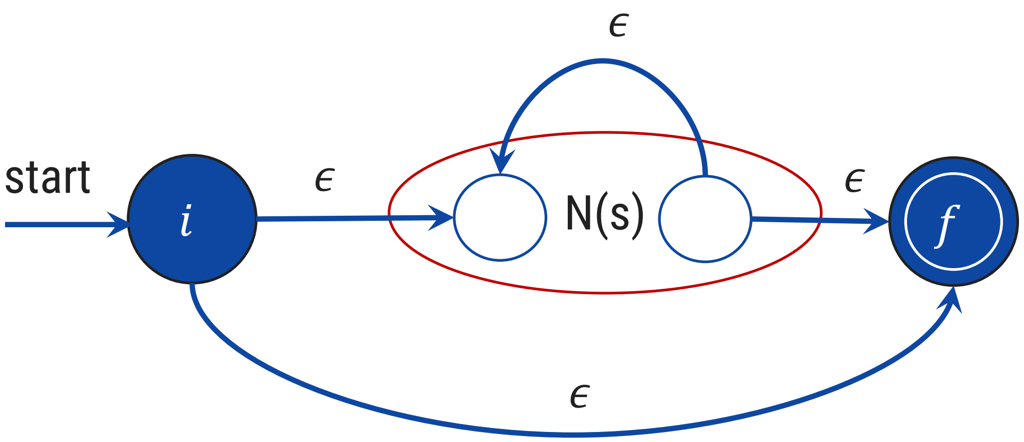
4. For regular expression $s|t$



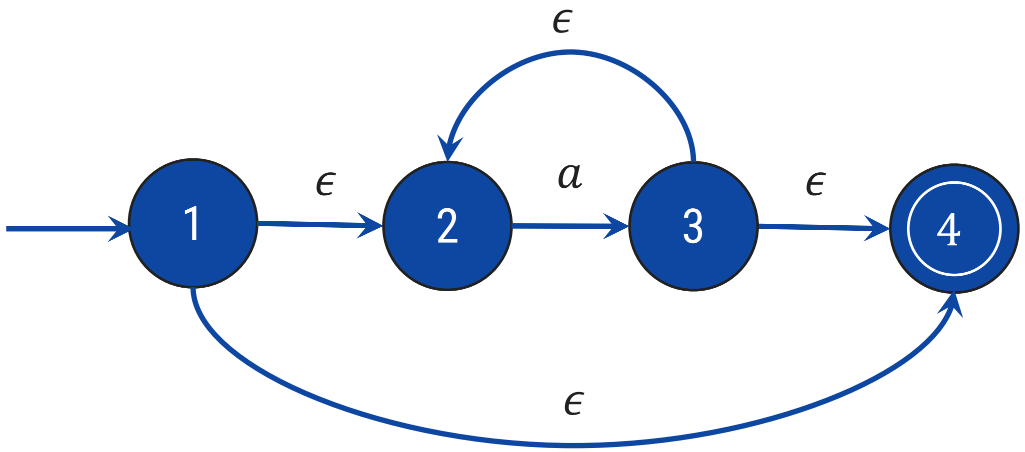
Ex: (a|b)



5. For regular expression $s*$

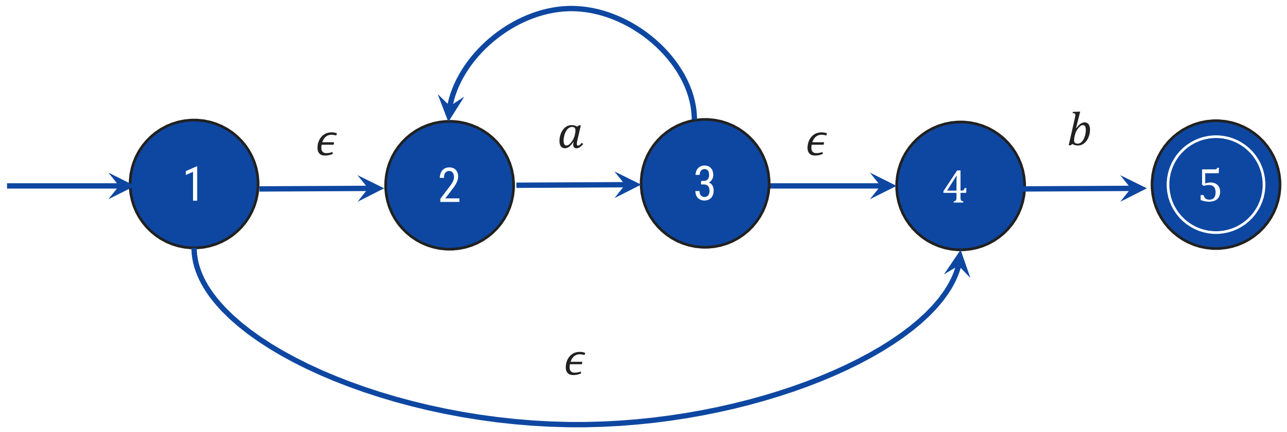


Ex: a*

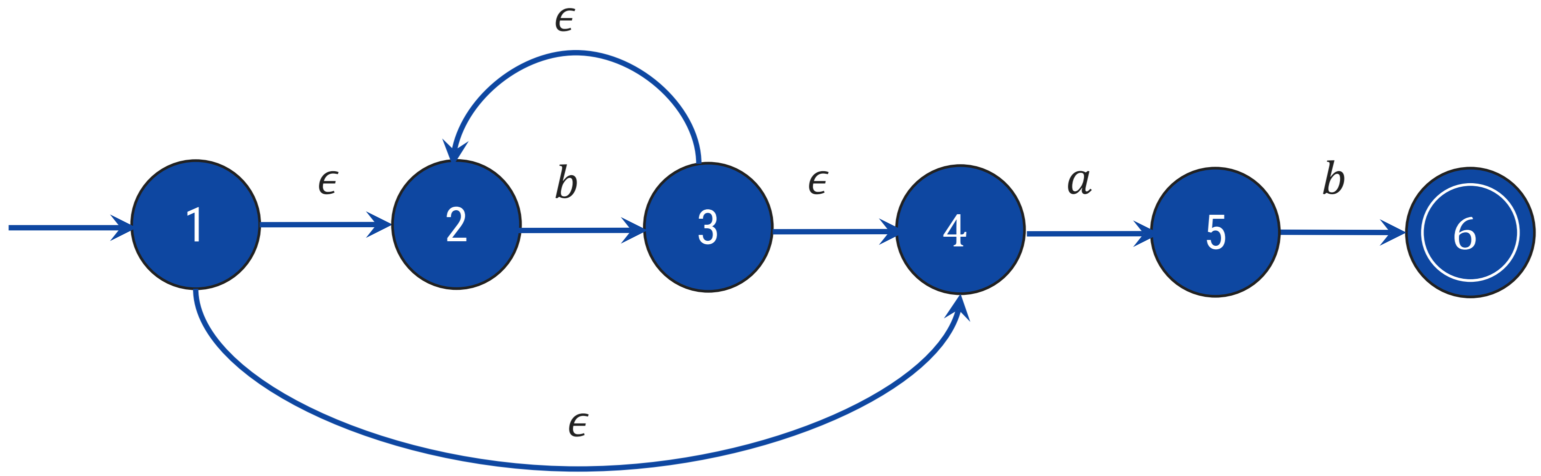


Regular expression to NFA using Thompson's rule

- a*b



- b*ab



Exercise

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)*

Conversion from NFA to DFA using subset construction method

Subset construction algorithm

Input: An NFA N.

Output: A DFA D accepting the same language.

Method: Algorithm construct a transition table $Dtran$ for D. We use the following operation:

OPERATION	DESCRIPTION
$\in -closure(s)$	Set of NFA states reachable from NFA state s on $\in-$ transition alone.
$\in -closure(T)$	Set of NFA states reachable from some NFA state s in T on $\in-$ transition alone.
$Move\ (T, a)$	Set of NFA states to which there is a transition on input symbol a from some NFA state s in T.

Subset construction algorithm

initially $\in -closure(s_0)$ be the only state in $Dstates$ and it is unmarked;

while there is unmarked states T in $Dstates$ ***do begin***

mark T;

for each input symbol a ***do begin***

$U = \epsilon - closure\big(move(T, a)\big)$;

if U is not in $Dstates$ ***then***

add U as unmarked state to $Dstates$;

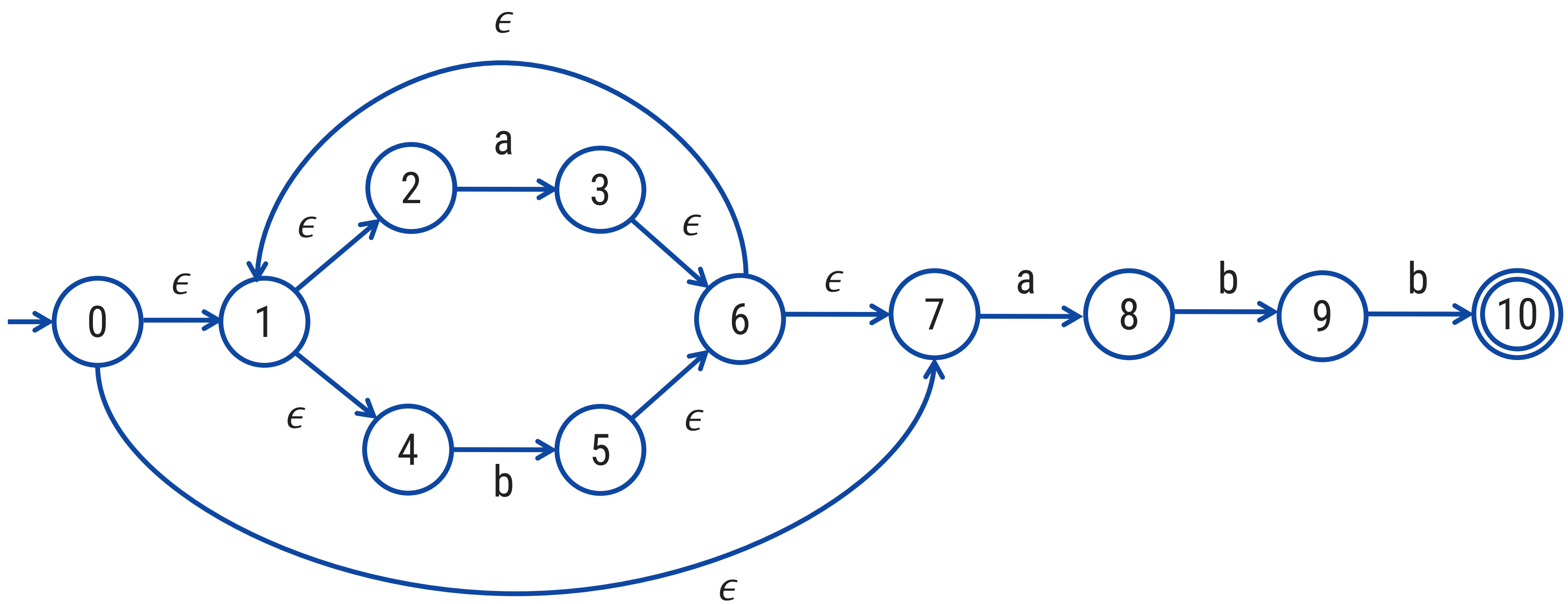
$Dtran[T, a] = U$

end

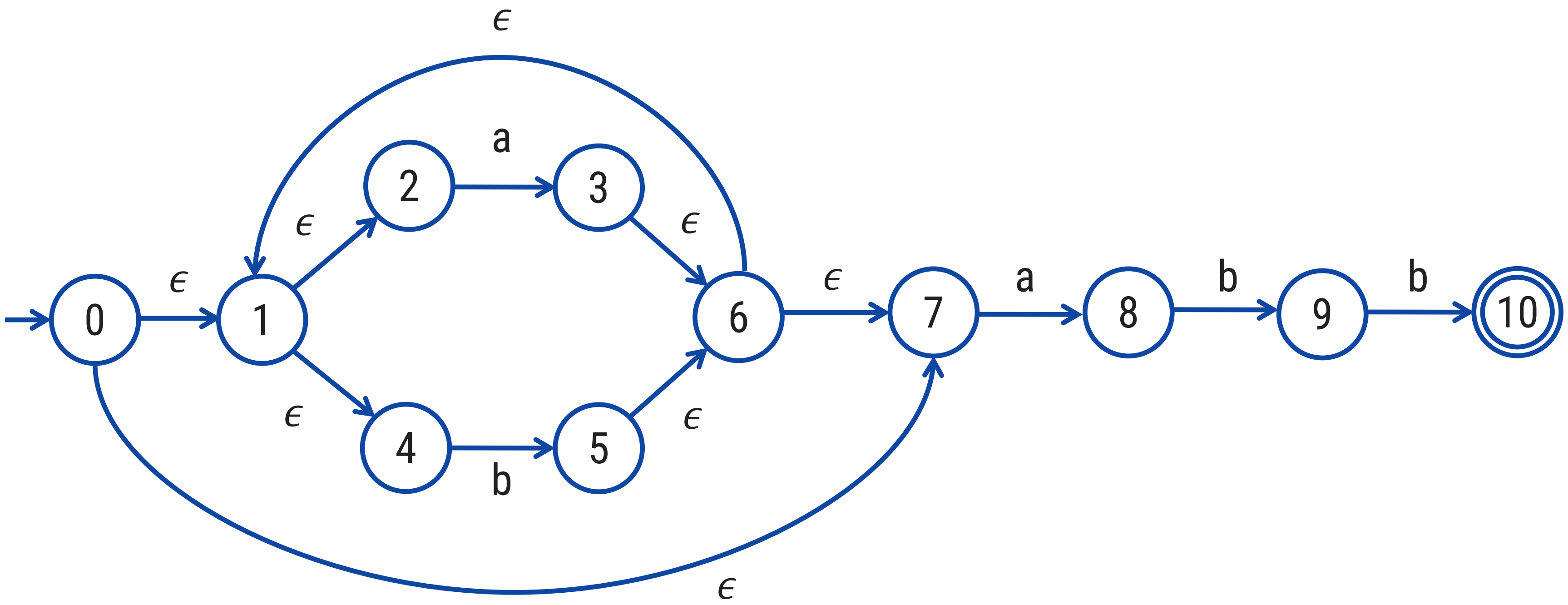
end

Conversion from NFA to DFA

$(a|b)^{*}abb$



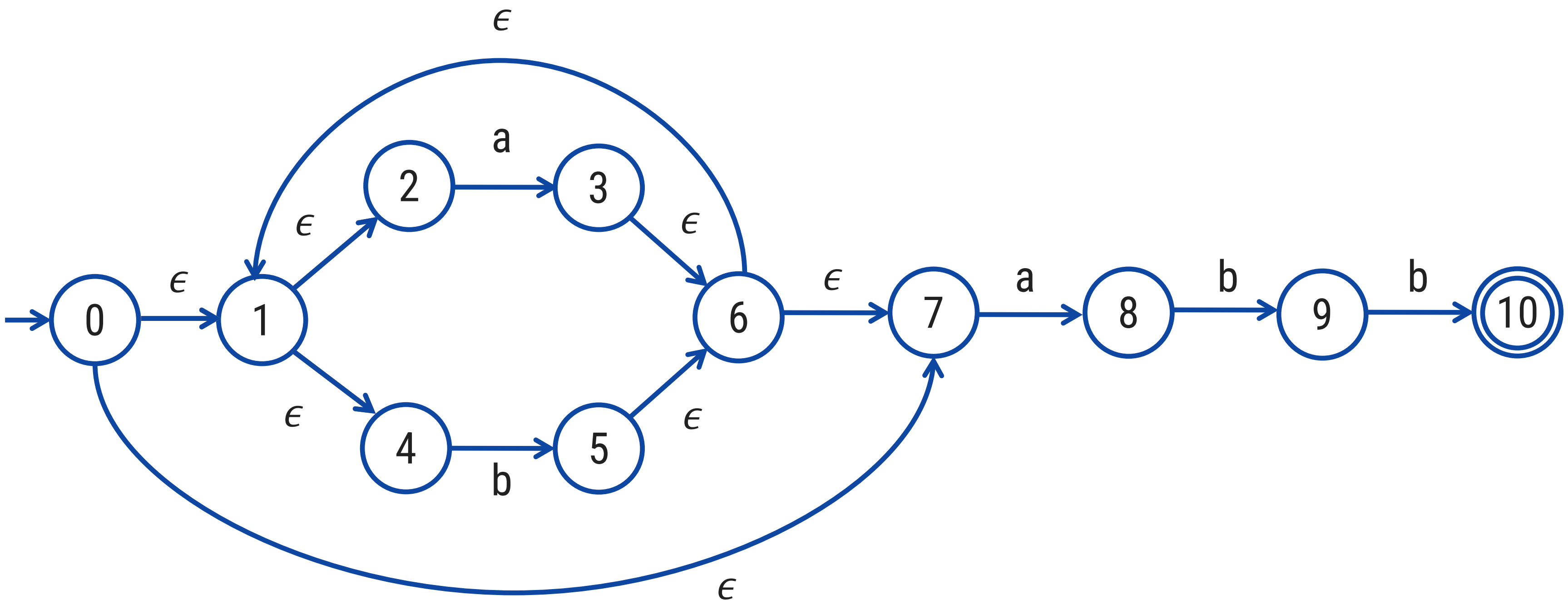
Conversion from NFA to DFA



ϵ- Closure(0)=

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

Conversion from NFA to DFA



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

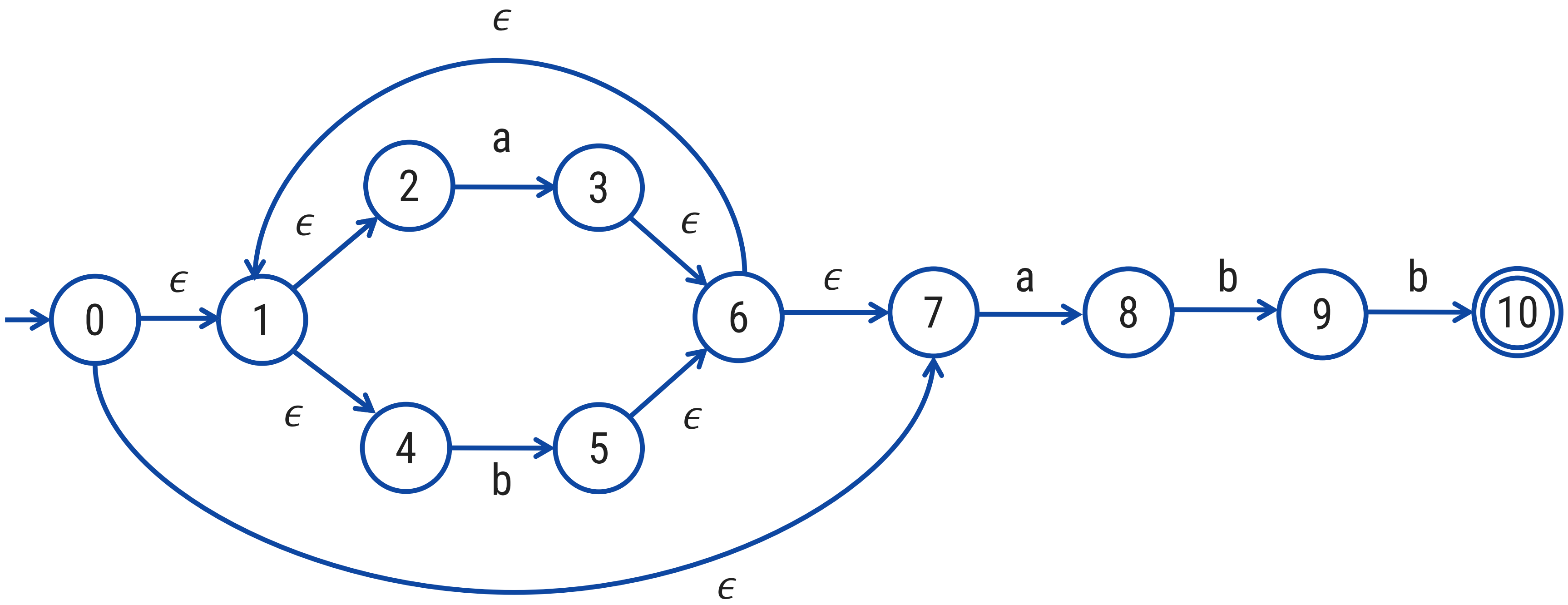
Move(A,a) = {3,8}

ϵ- Closure(Move(A,a))

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

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

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}		

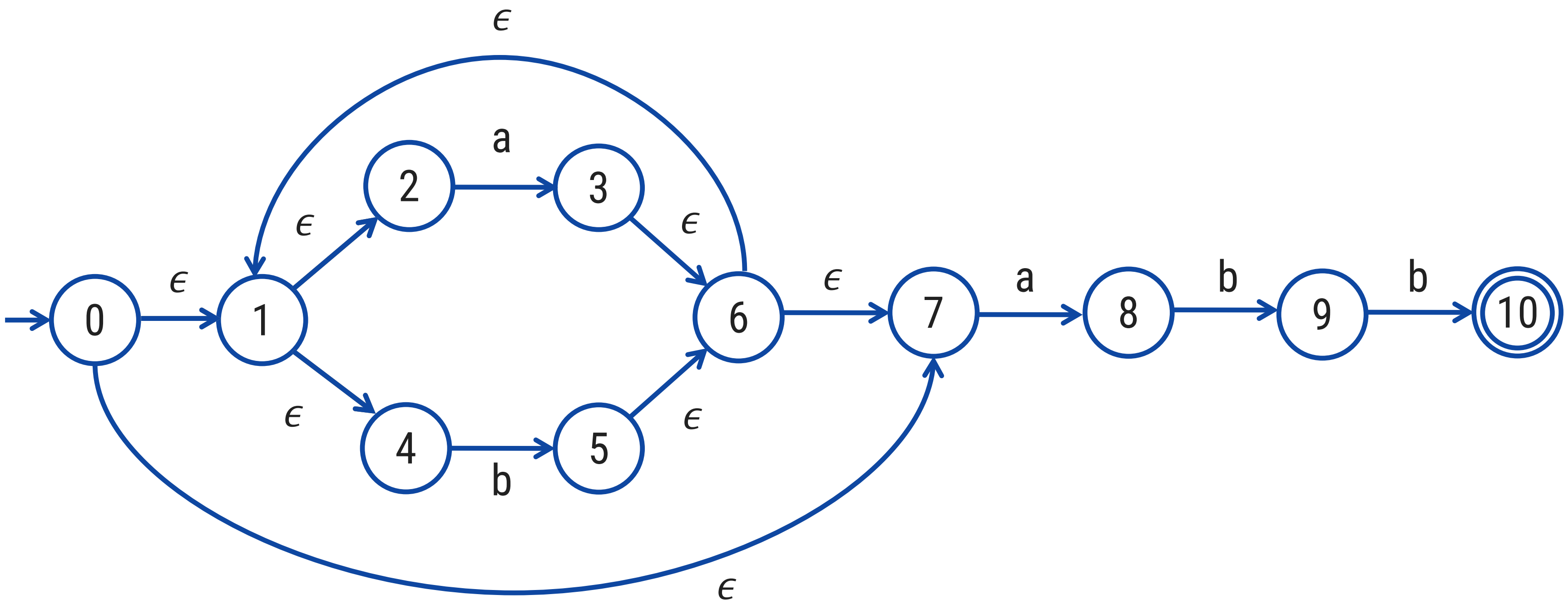
A= {0, 1, 2, 4, 7}

Move(A,b) =

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}		

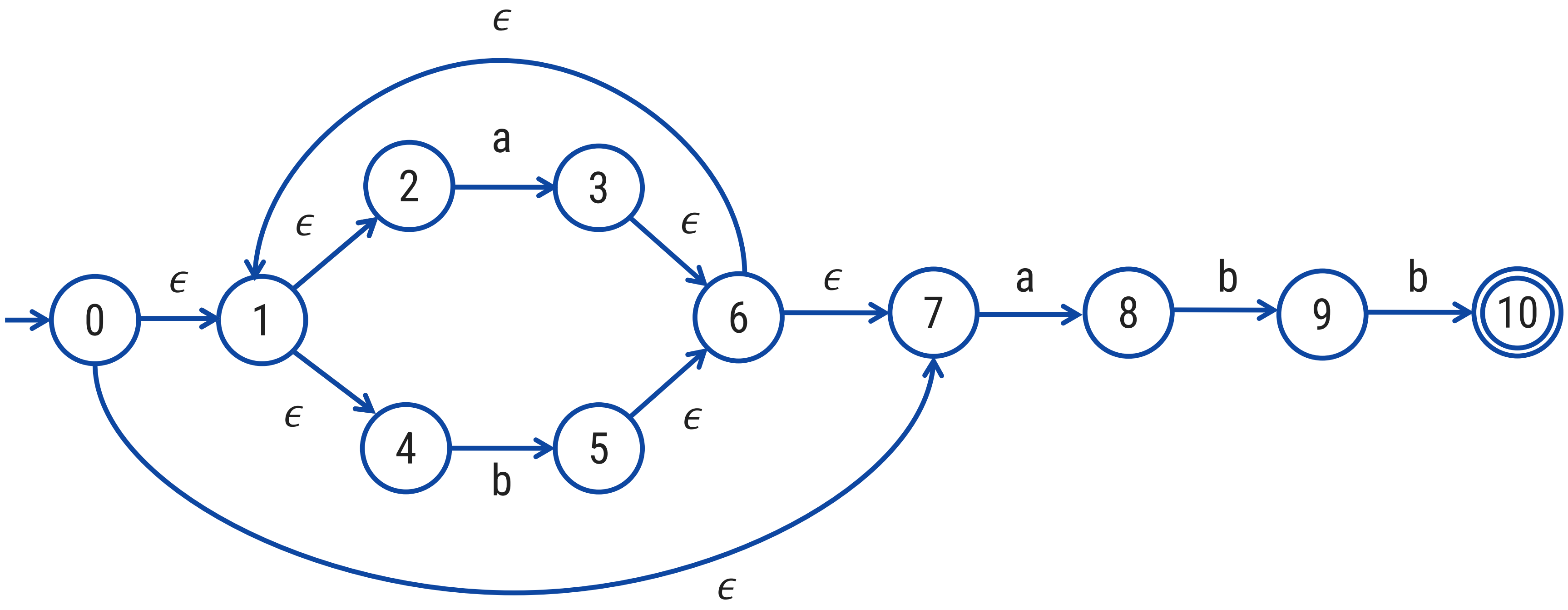
B = {1, 2, 3, 4, 6, 7, 8}

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

ϵ- Closure(Move(B,a))

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

Conversion from NFA to DFA



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}		

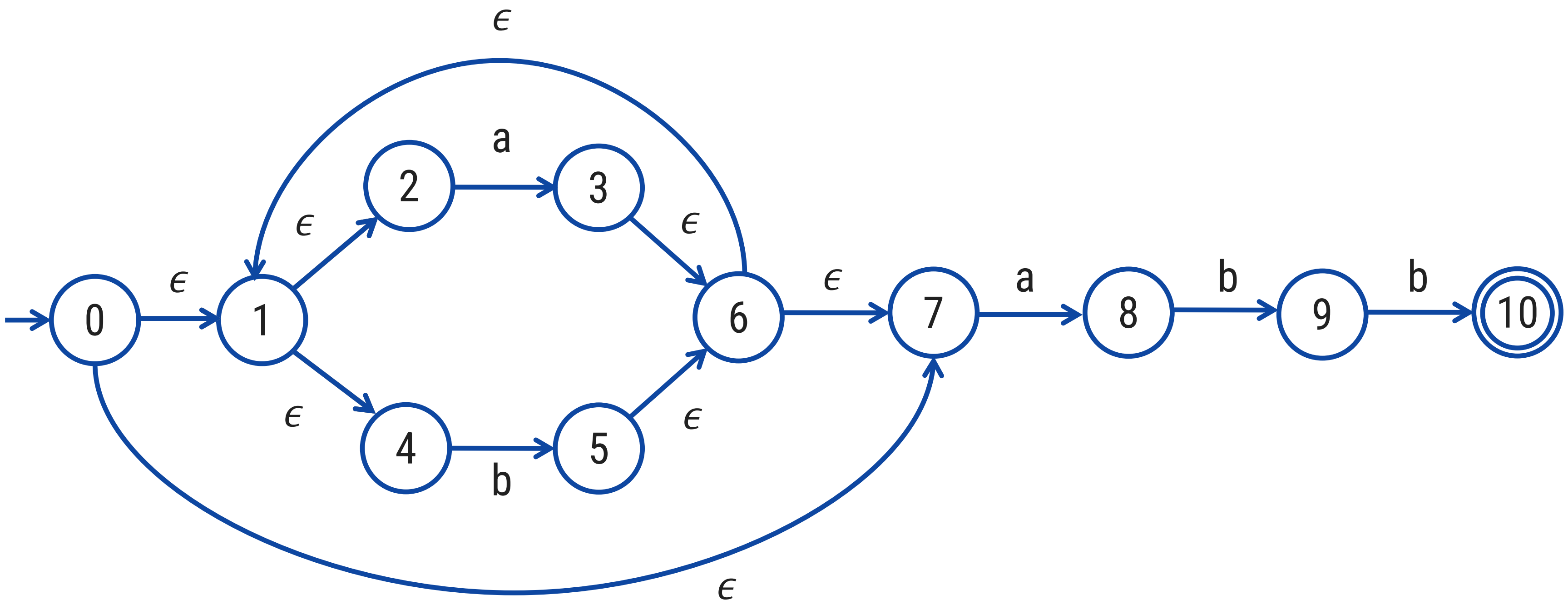
B= {1, 2, 3, 4, 6, 7, 8}

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

ϵ- Closure(Move(B,b))

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

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}		

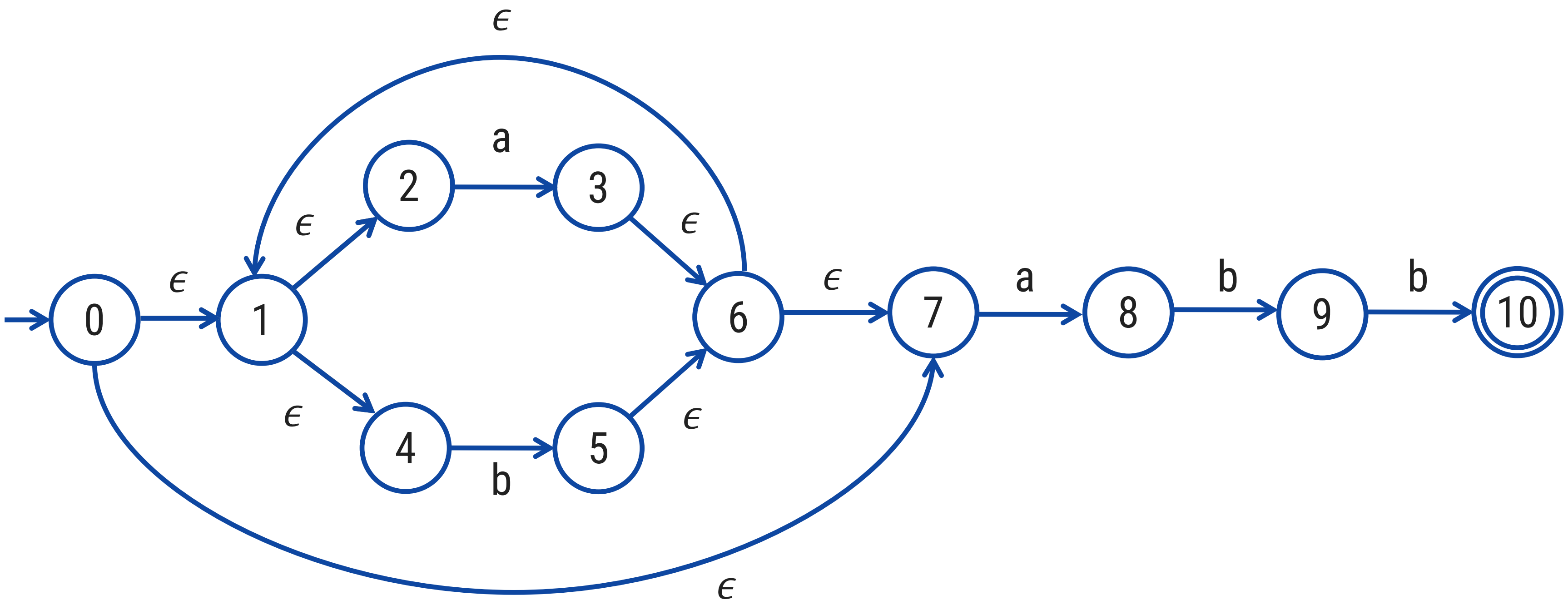
C= {1, 2, 4, 5, 6 ,7}

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}		

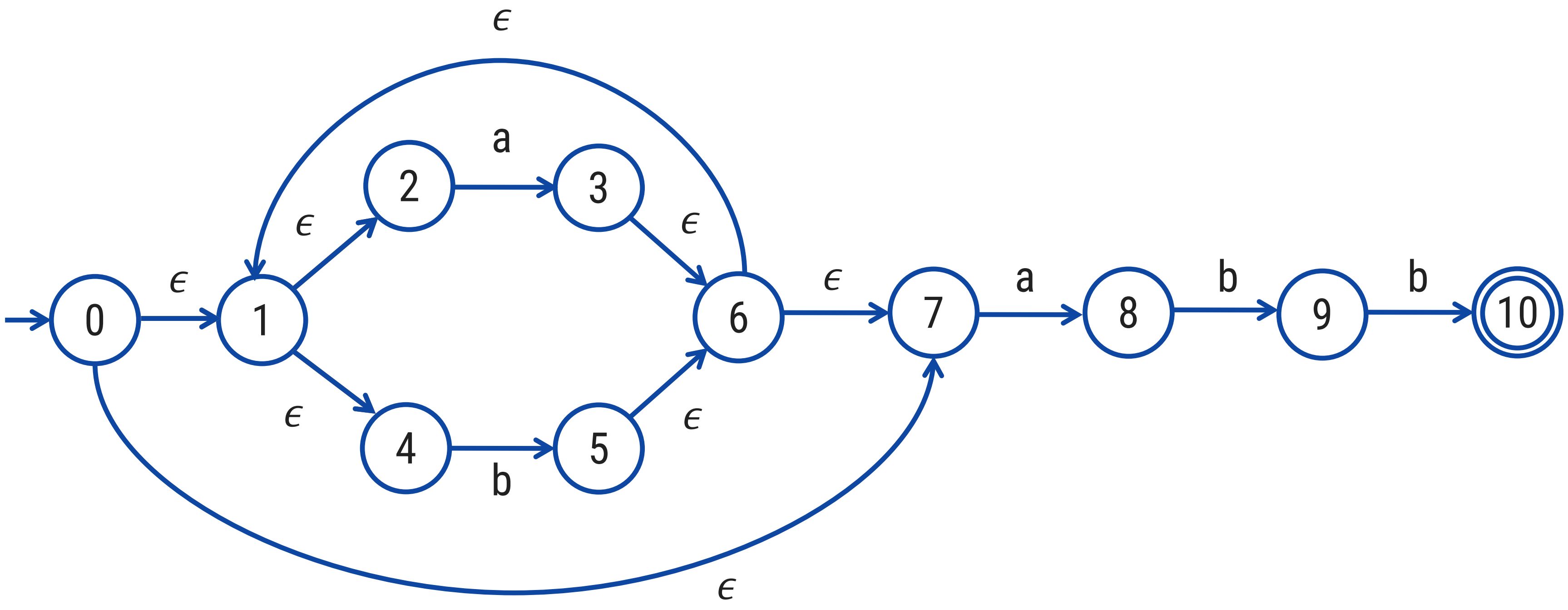
C= {1, 2, 4, 5, 6, 7}

Move(C,b) =

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}		

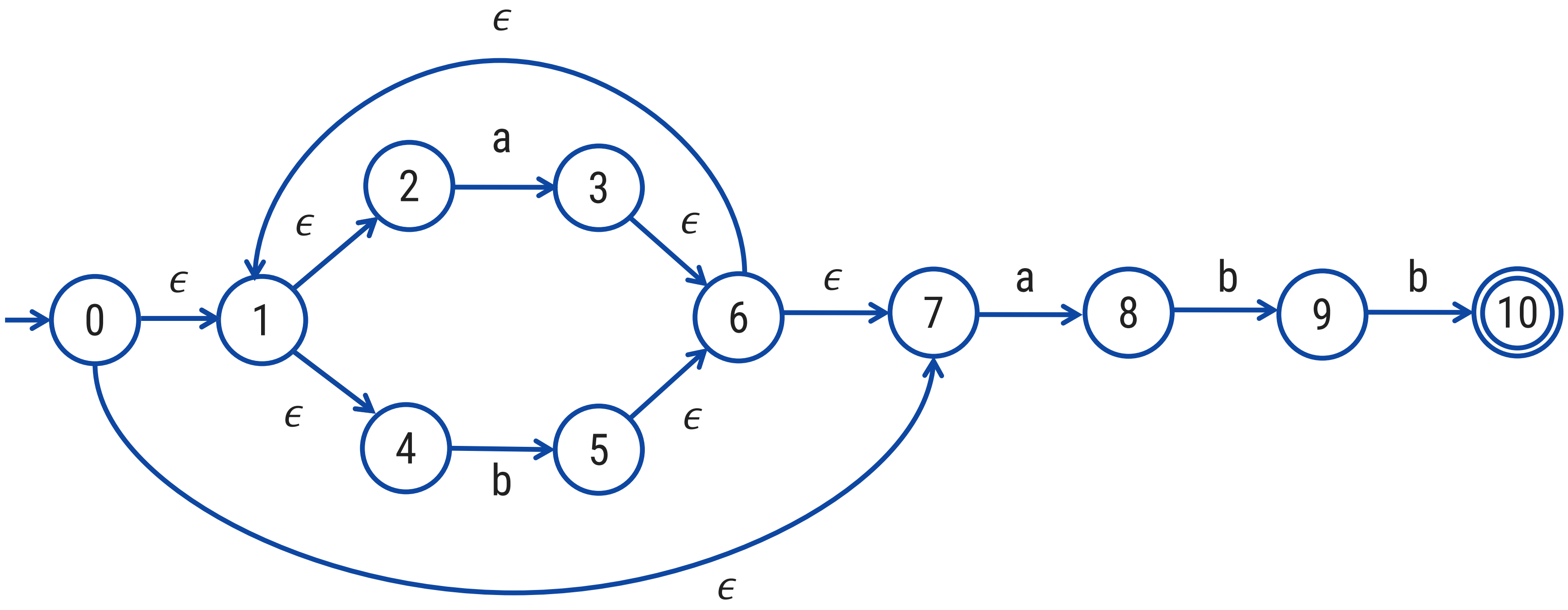
D= {1, 2, 4, 5, 6, 7, 9}

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}		

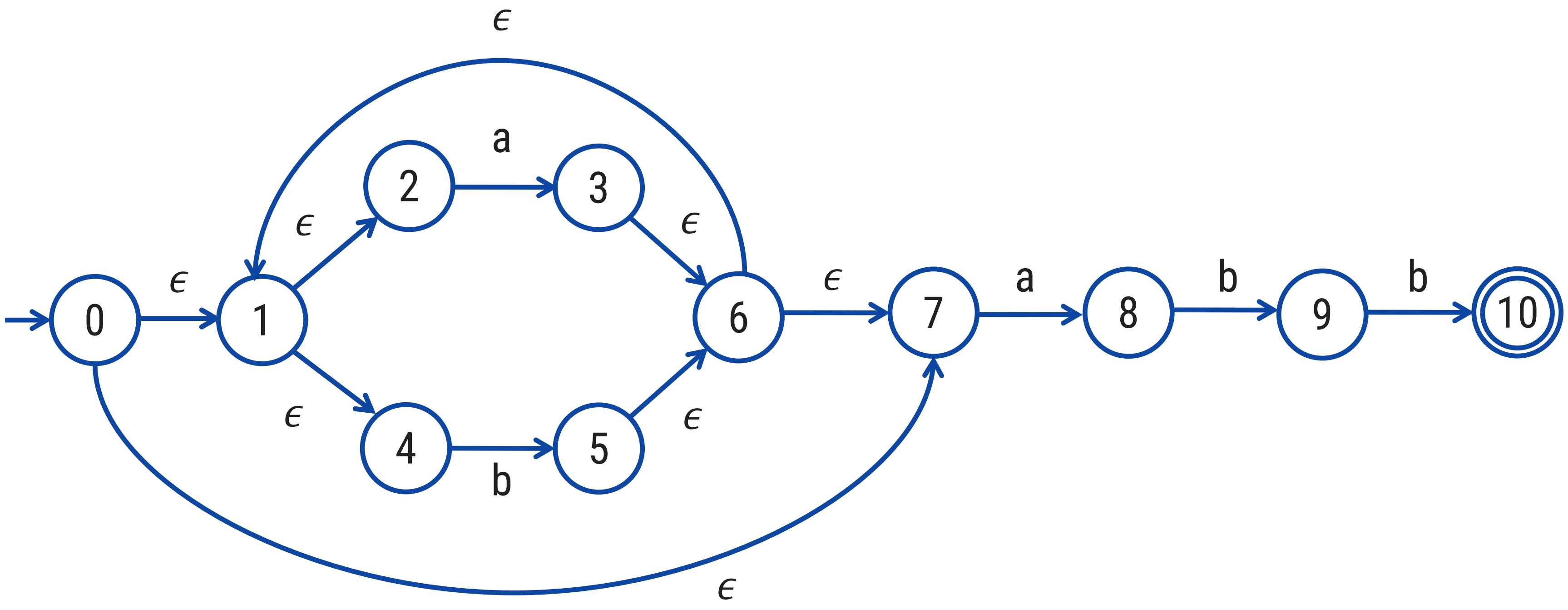
D= {1, 2, 4, 5, 6, 7, 9}

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

ϵ- Closure(Move(D,b))

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

Conversion from NFA to DFA



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}		

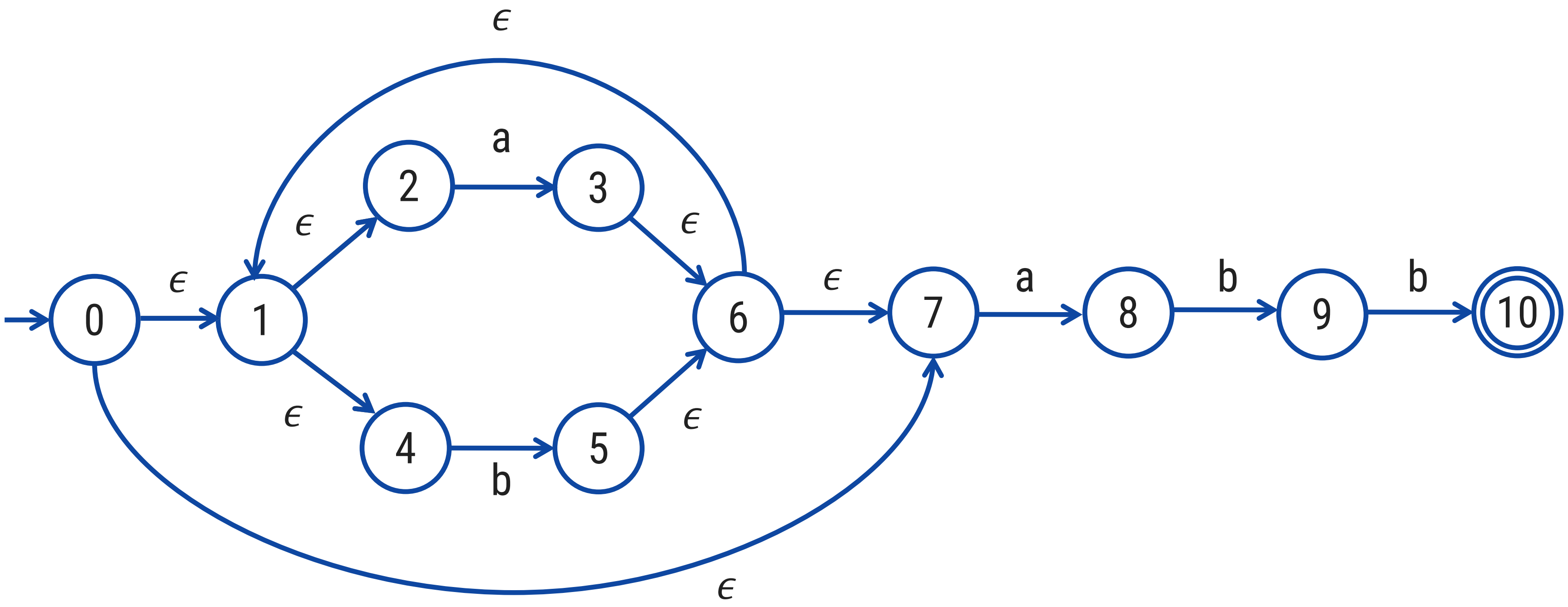
E= {1, 2, 4, 5, 6, 7, 10}

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

ϵ- Closure(Move(E,a))

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

Conversion from NFA to DFA



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

Move(E,b)=

ϵ- Closure(Move(E,b))=

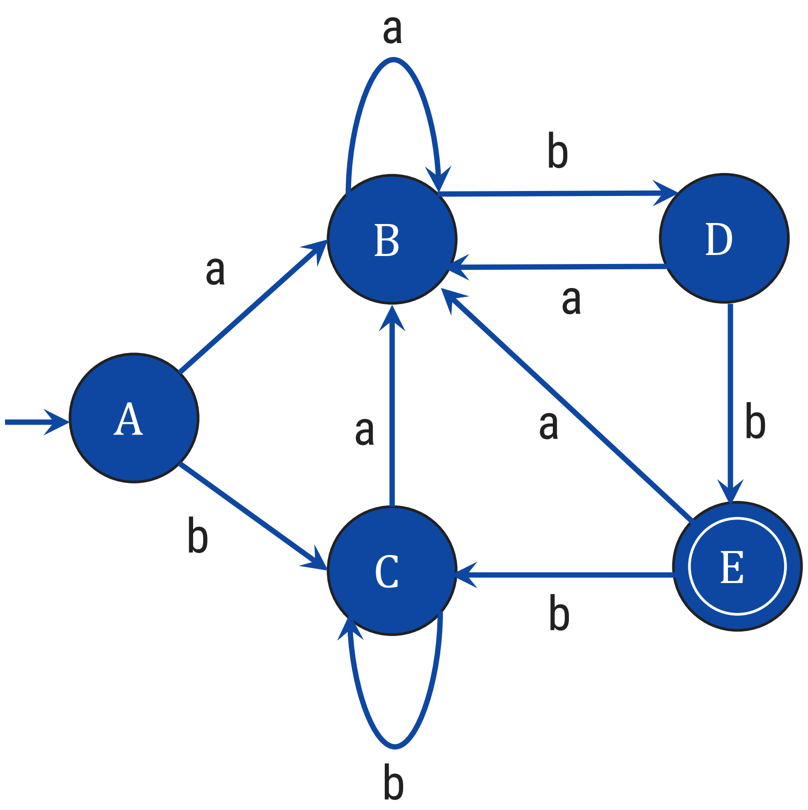
= {1,2,4,5,6,7} **---- C**

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**

Exercise

- Convert following regular expression to DFA using subset construction method:
  1. (a+b)*a(a+b)
  2. (a+b)*ab*a

DFA optimization

DFA optimization

1. Construct an initial partition Π of the set of states with two groups: the accepting states F and the non-accepting states $S - F$.
2. Apply the repartition procedure to Π to construct a new partition Πnew.
3. If $\Pi new = \Pi$, let $\Pi final = \Pi$ and continue with step (4). Otherwise, repeat step (2) with $\Pi = \Pi new$.

for each group G of Π **do begin**

partition G into subgroups such that two states s and t of G are in the same subgroup if and only if for all input symbols a, states s and t have transitions on a to states in the same group of Π.

replace G in Πnew by the set of all subgroups formed.

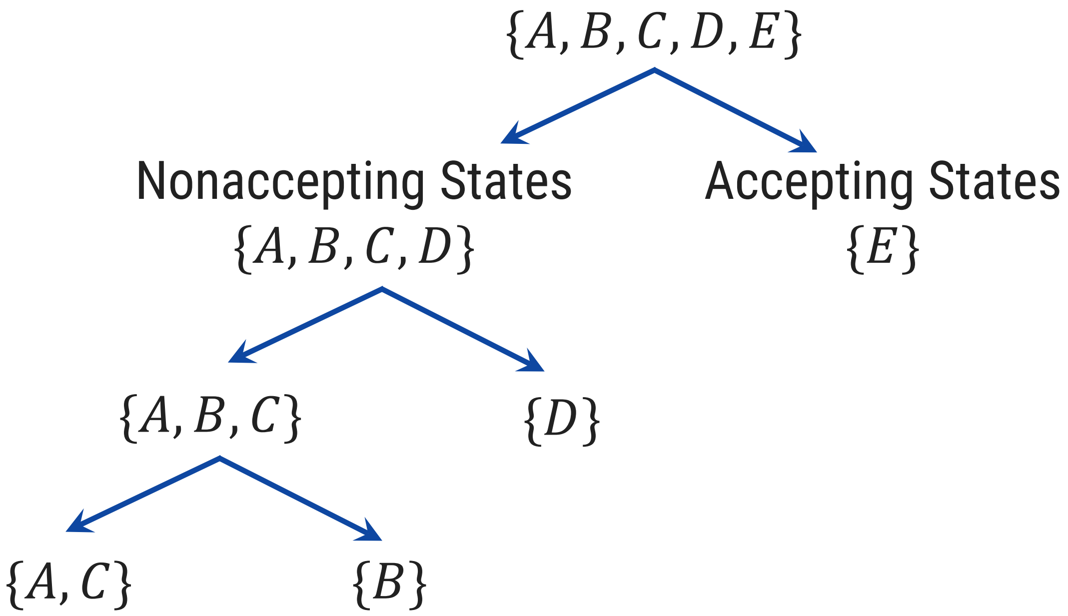
end

DFA optimization

4. Choose one state in each group of the partition $\Pi final$ as the representative for that group. The representatives will be the states of M'. Let s be a representative state, and suppose on input a there is a transition of M from s to t. Let r be the representative of t's group. Then M' has a transition from s to r on a. Let the start state of M' be the representative of the group containing start state s_0 of M, and let the accepting states of M' be the representatives that are in F.

5. If M' has a dead state d, then remove d from M'. Also remove any state not reachable from the start state.

DFA optimization



- Now no more splitting is possible.
- If we chose A as the representative for group (AC), then we obtain reduced transition table

States	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

States	a	b
A	B	A
B	B	D
D	B	E
E	B	A

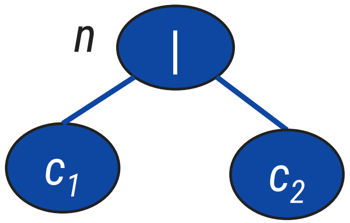
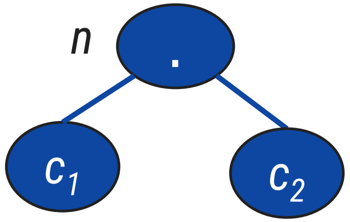
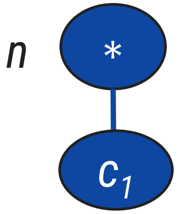
Optimized Transition Table

Conversion from regular expression to DFA

Rules to compute nullable, firstpos, lastpos

- nullable(n)
  - ↪ The subtree at node n generates languages including the empty string.
- firstpos(n)
  - ↪ The 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.

Rules to compute nullable, firstpos, lastpos

Node n	nullable(n)	firstpos(n)	lastpos(n)
A leaf labeled by ε	**true**	$\emptyset$	$\emptyset$
A leaf with position **i**	**false**	$\{i\}$	$\{i\}$
	nullable(c_1) **or** nullable(c_2)	firstpos(c_1) $\cup$ firstpos(c_2)	lastpos(c_1) $\cup$ lastpos(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)
	**true**	firstpos(c_1)	lastpos(c_1)

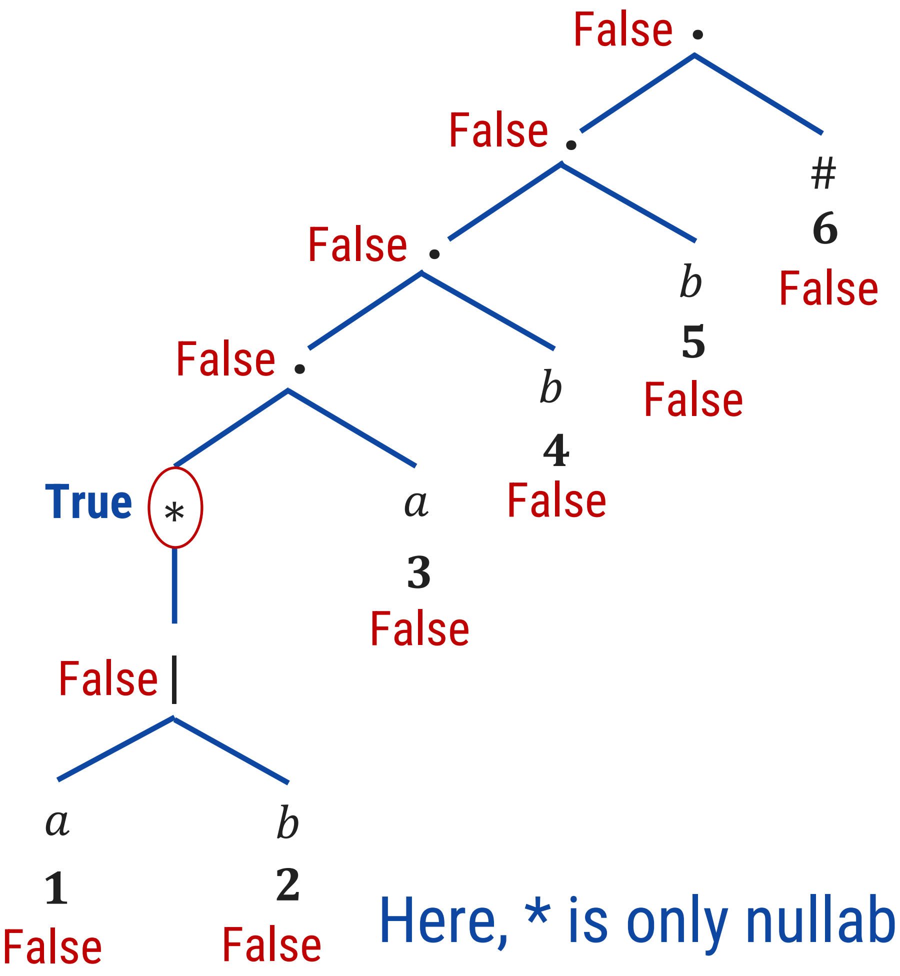
Rules to compute followpos

1. If n is **concatenation** node with left child c1 and right child c2 and ***i*** is a position in lastpos(c1), then all position in firstpos(c2) are in followpos(i)

2. If n is * node and ***i*** is position in lastpos(n), then all position in firstpos(n) are in followpos(i)

Conversion from regular expression to DFA

(a|b) * abb #



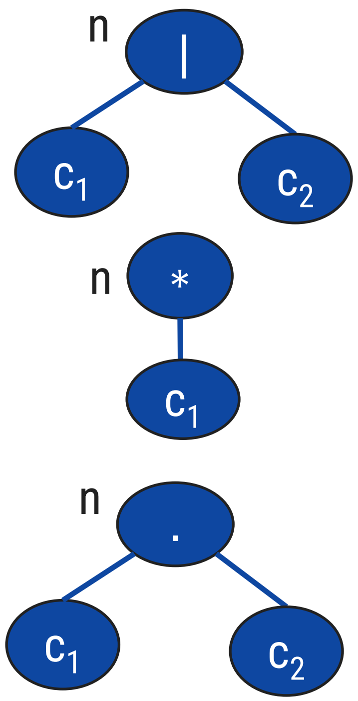
Here, * is only nullable node

Step 1: Construct Syntax Tree

Step 2: Nullable node

A leaf labeled by ε = **True**

A leaf with position i = **false**



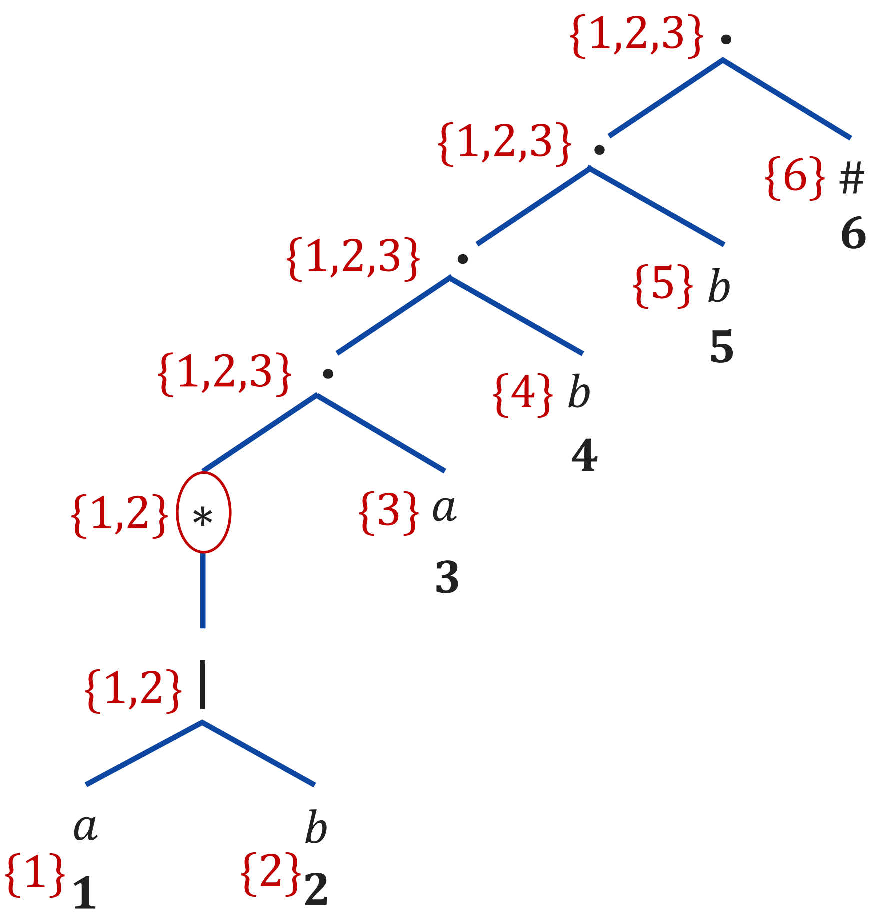
nullable(c_1)
or
nullable(c_2)

true

nullable(c_1)
and
nullable(c_2)

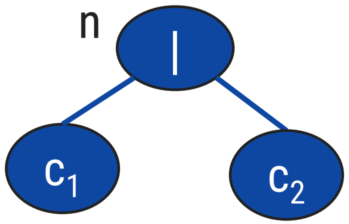
Conversion from regular expression to DFA

Step 3: Calculate firstpos

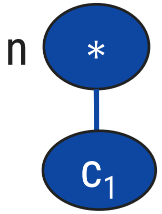


Firstpos —

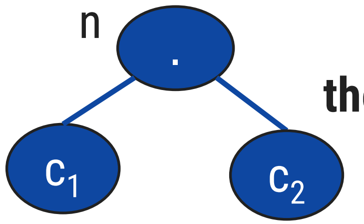
A leaf with position $i = \{i\}$



$firstpos(c_1) \cup firstpos(c_2)$



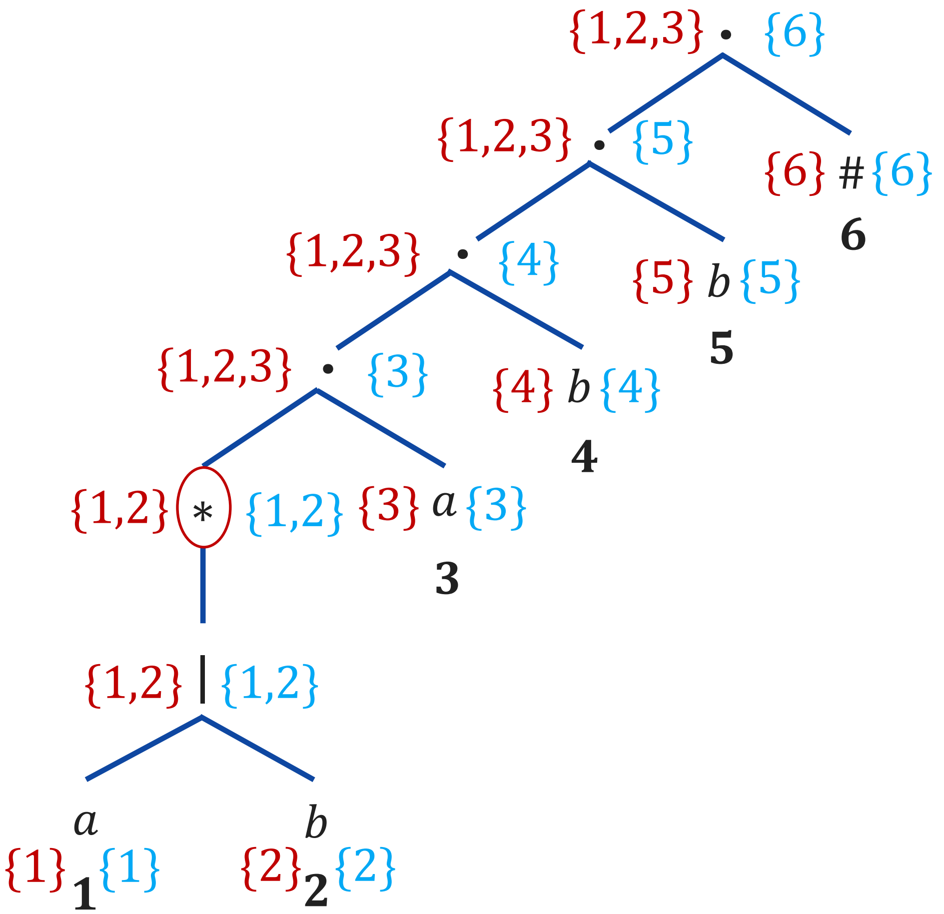
$firstpos(c_1)$



if $(nullable(c_1))$
then $firstpos(c_1) \cup firstpos(c_2)$
else $firstpos(c_1)$

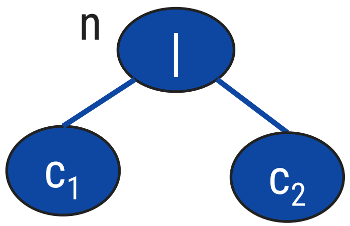
Conversion from regular expression to DFA

Step 3: Calculate lastpos

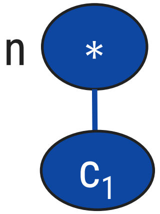


Lastpos —

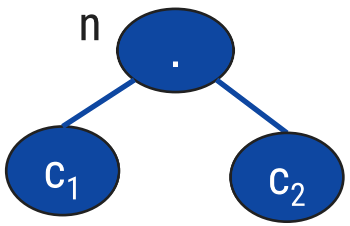
A leaf with position $\boldsymbol{i}$ = $\{\boldsymbol{i}\}$



$lastpos(c_1) \cup lastpos(c_2)$



$lastpos(c_1)$

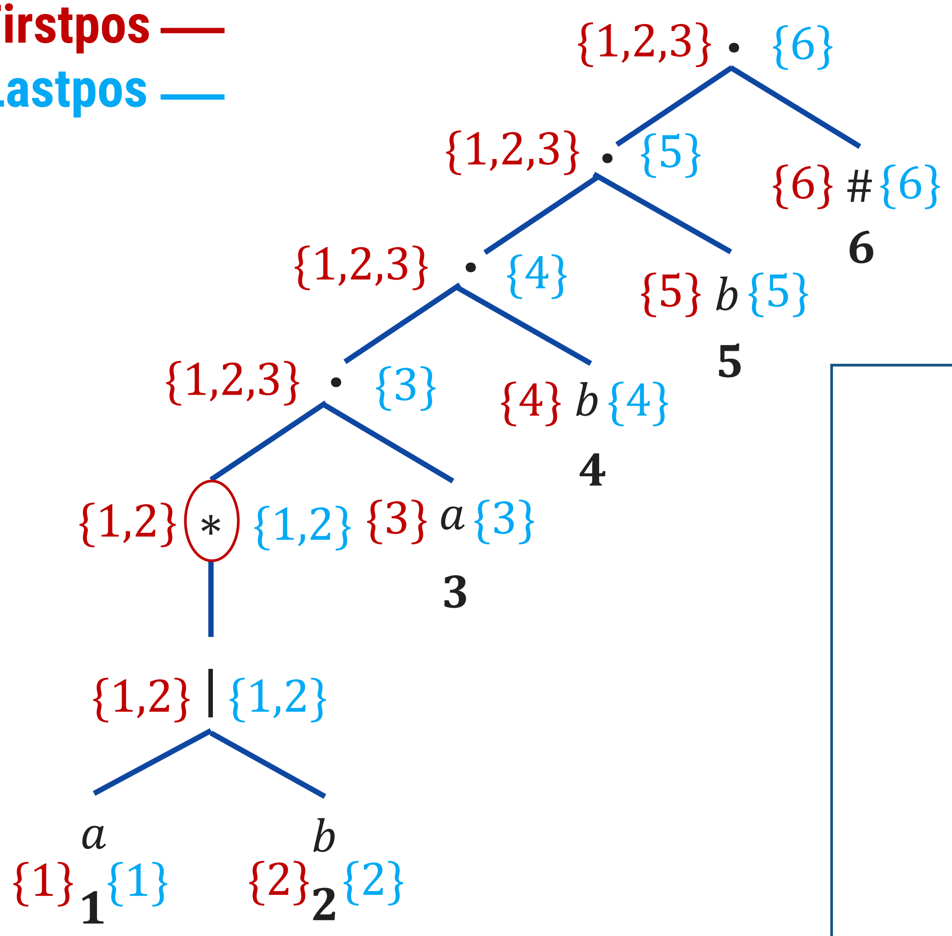


if $(nullable(c_2))$ **then** $lastpos(c_1) \cup lastpos(c_2)$ **else** $lastpos(c_2)$

Conversion from regular expression to DFA

Step 4: Calculate followpos

Firstpos ——
Lastpos ——



Position	followpos
5	
4	
3	
2	1,2,
1	1,2,

Rule:
If n is * node and ***i*** is position in lastpos(n), then all position in firstpos(n) are in followpos(i)

{1,2} * {1,2}

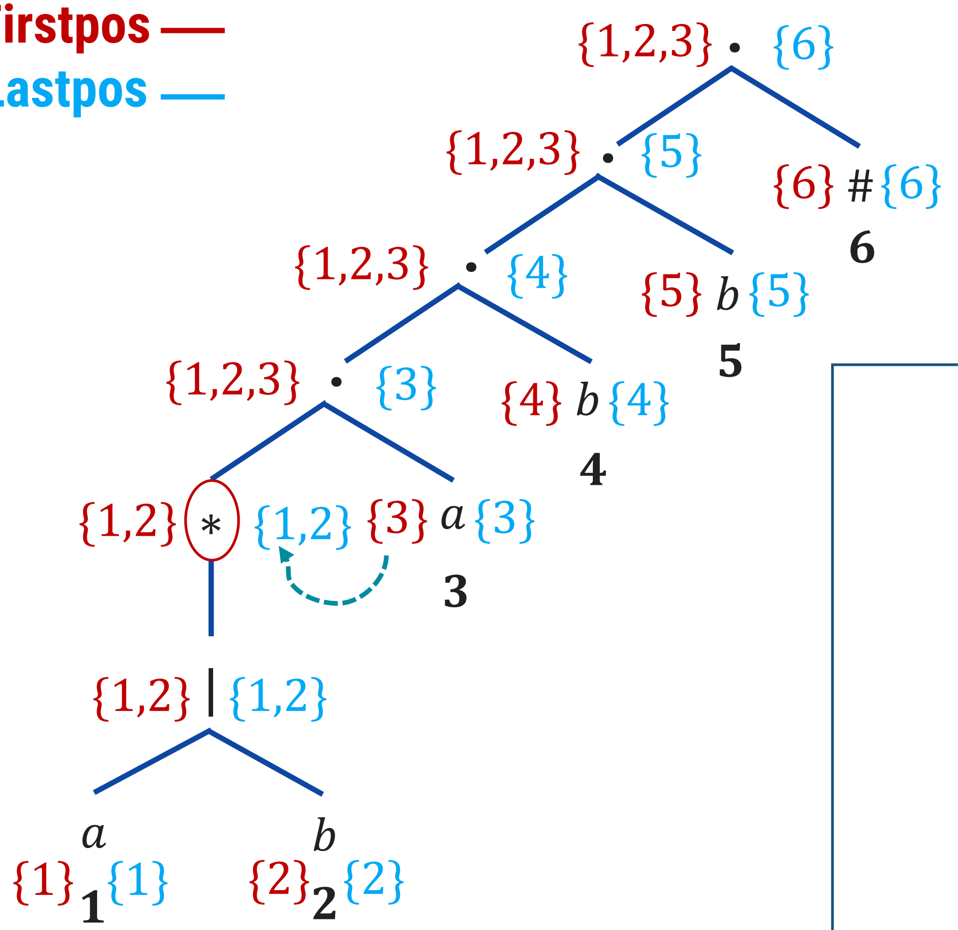
n

$$i = lastpos(n) = \{1,2\}$$
$$firstpos(n) = \{1,2\}$$
$$followpos(1) = \{1,2\}$$
$$followpos(2) = \{1,2\}$$

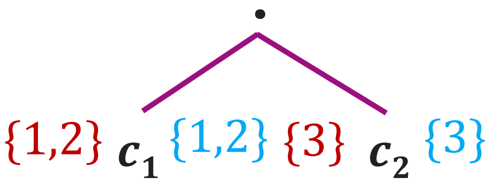
Conversion from regular expression to DFA

Step 4: Calculate followpos

Firstpos —
Lastpos —



Position	followpos
5	
4	
3	
2	1,2,3
1	1,2,3

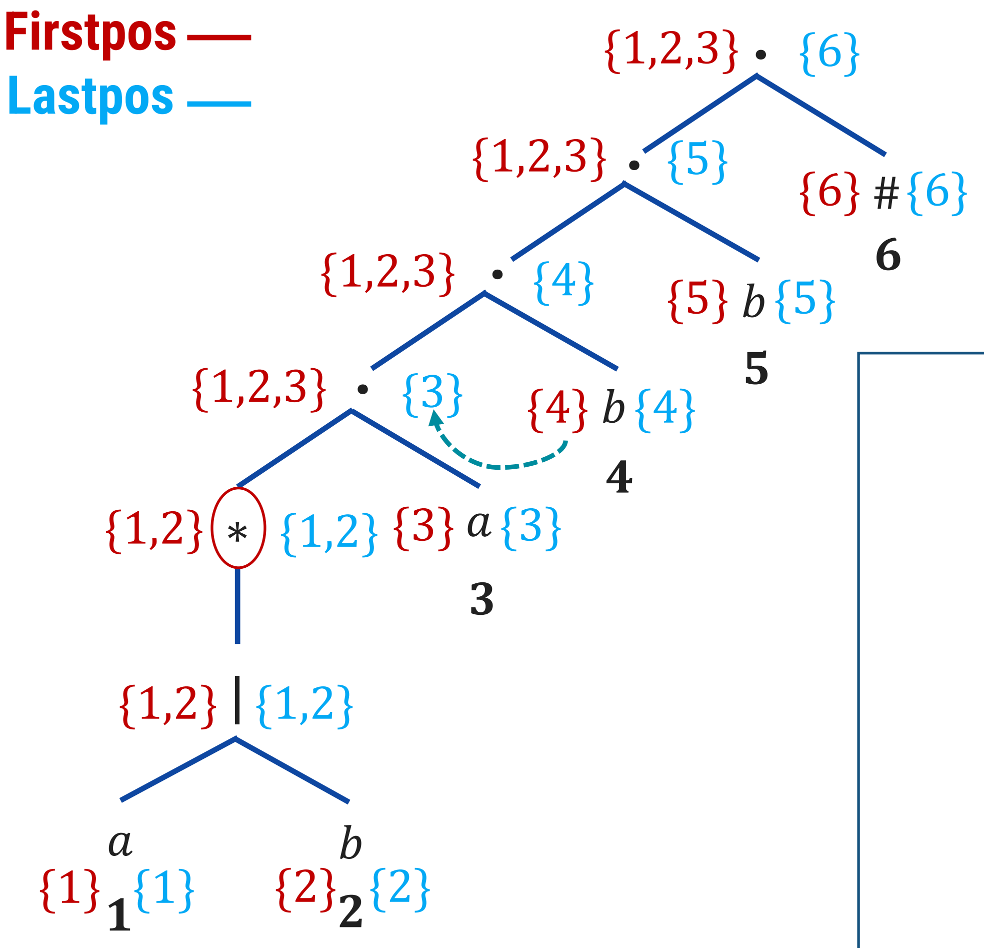


$$i = lastpos(c_1) = \{1,2\}$$
$$firstpos(c_2) = \{3\}$$
$$followpos(1) = \{3\}$$
$$followpos(2) = \{3\}$$

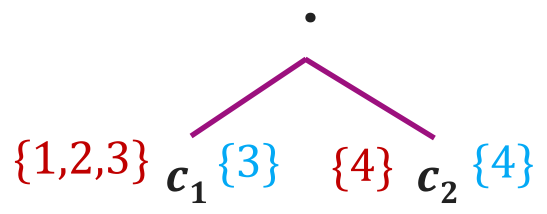
Rule:
If n is **concatenation** node with left child c1 and right child c2 and ***i*** is a position in lastpos(c1), then all position in firstpos(c2) are in followpos(i)

Conversion from regular expression to DFA

Step 4: Calculate followpos



Position	followpos
5	
4	
3	4
2	1,2,3
1	1,2,3

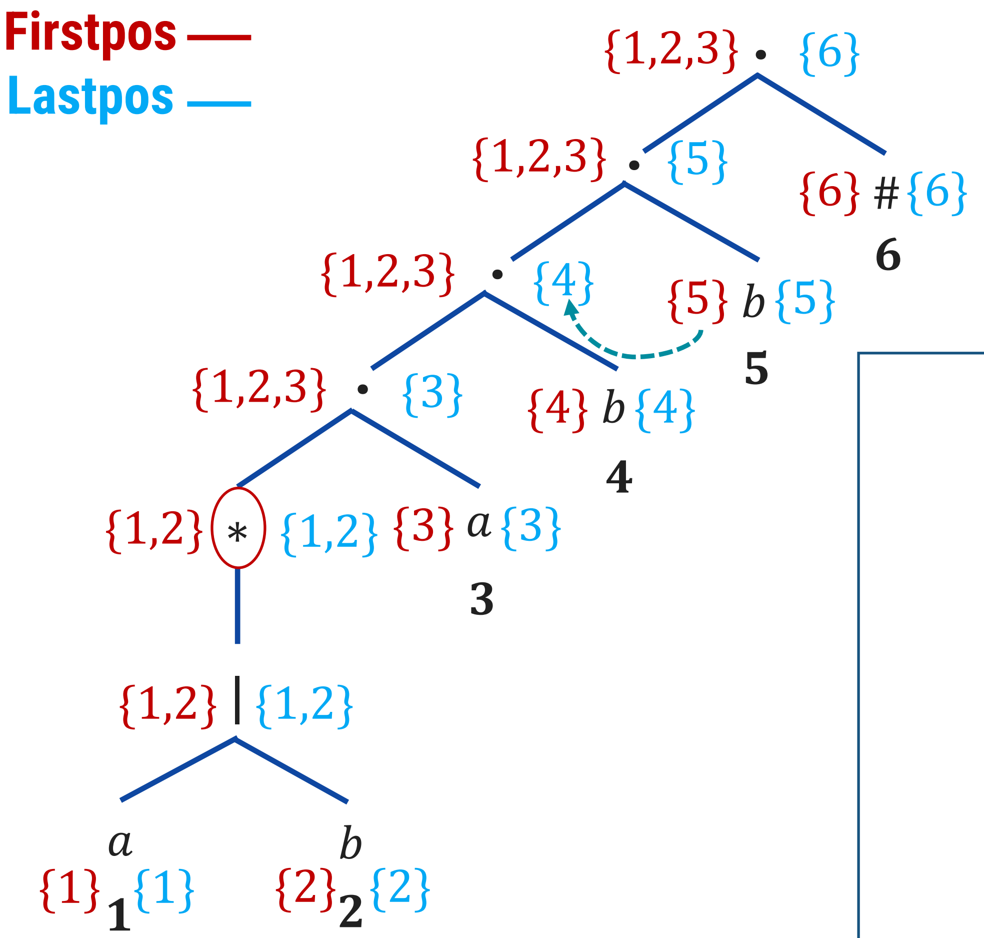


$$i = lastpos(c_1) = \{3\}$$
$$firstpos(c_2) = \{4\}$$
$$followpos(3) = \{4\}$$

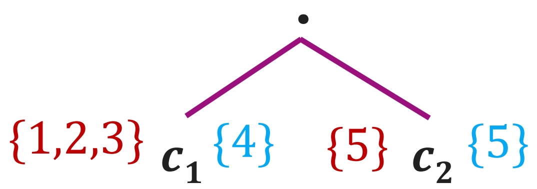
Rule:
If n is **concatenation** node with left child c1 and right child c2 and ***i*** is a position in lastpos(c1), then all position in firstpos(c2) are in followpos(i)

Conversion from regular expression to DFA

Step 4: Calculate followpos



Position	followpos
5	
4	5
3	4
2	1,2,3
1	1,2,3



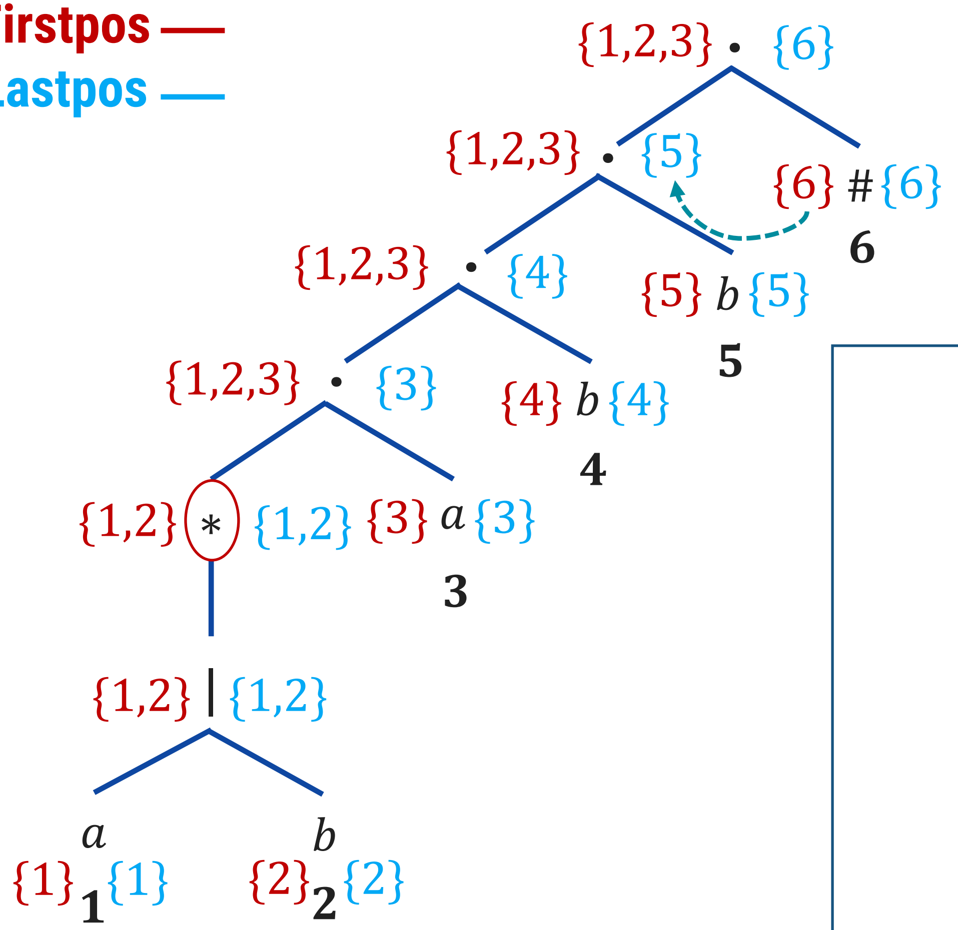
$$i = lastpos(c_1) = \{4\}$$
$$firstpos(c_2) = \{5\}$$
$$followpos(4) = \{5\}$$

Rule:
If n is **concatenation** node with left child c1 and right child c2 and ***i*** is a position in lastpos(c1), then all position in firstpos(c2) are in followpos(i)

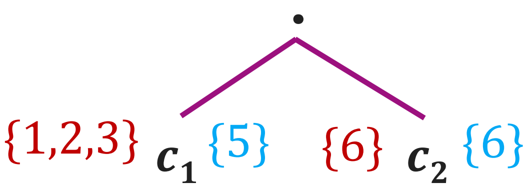
Conversion from regular expression to DFA

Step 4: Calculate followpos

Firstpos —
Lastpos —



Position	followpos
5	6
4	5
3	4
2	1,2,3
1	1,2,3



$$i = lastpos(c_1) = \{5\}$$
$$firstpos(c_2) = \{6\}$$
$$followpos(5) = \{6\}$$

Rule:
If n is **concatenation** node with left child c1 and right child c2 and ***i*** is a position in lastpos(c1), then all position in firstpos(c2) are in followpos(i)

Conversion from regular expression to DFA

Initial state = $firstpos$ of root = {1,2,3} ----- A

<u>**State A**</u>

δ((1,2,3),a) = followpos(1) U followpos(3)

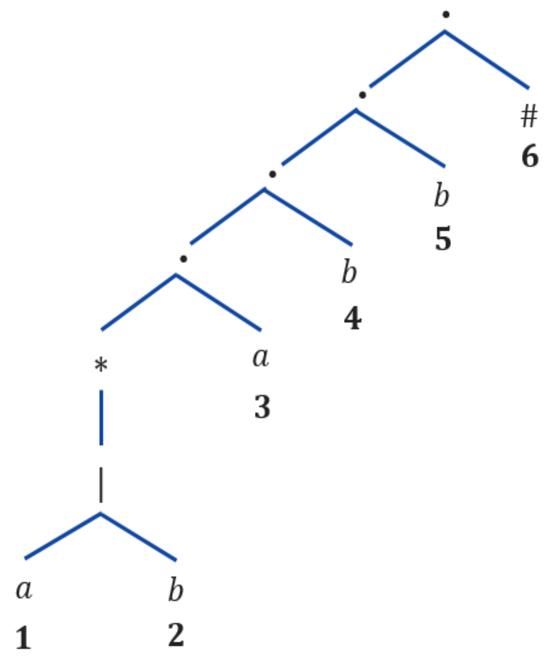
=(1,2,3) U (4) = {1,2,3,4} ----- B

δ((1,2,3),b) = followpos(2)

=(1,2,3) ----- A

Position	followpos
5	6
4	5
3	4
2	1,2,3
1	1,2,3

States	a	b
A={1,2,3}		
B={1,2,3,4}		



Conversion from regular expression to DFA

State B

δ((1,2,3,4),a) = followpos(1) U followpos(3)

=(1,2,3) U (4) = {1,2,3,4} ----- B

δ((1,2,3,4),b) = followpos(2) U followpos(4)

=(1,2,3) U (5) = {1,2,3,5} ----- C

State C

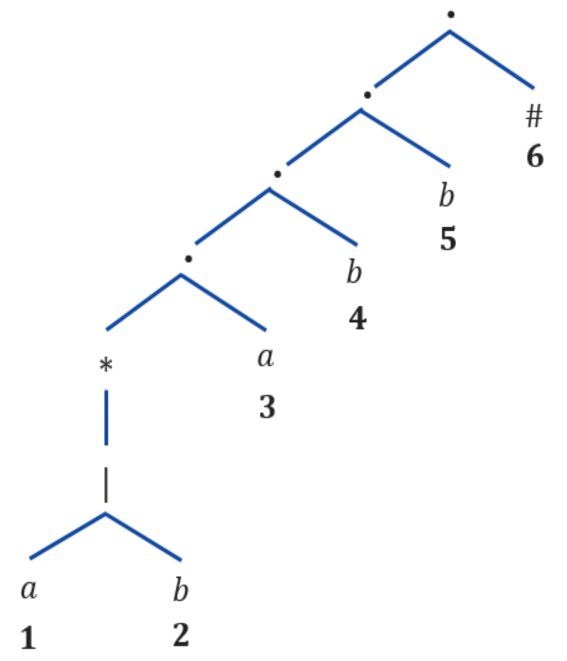
δ((1,2,3,5),a) = followpos(1) U followpos(3)

=(1,2,3) U (4) = {1,2,3,4} ----- B

δ((1,2,3,5),b) = followpos(2) U followpos(5)

=(1,2,3) U (6) = {1,2,3,6} ----- D

Position	followpos
5	6
4	5
3	4
2	1,2,3
1	1,2,3



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

Conversion from regular expression to DFA

State D

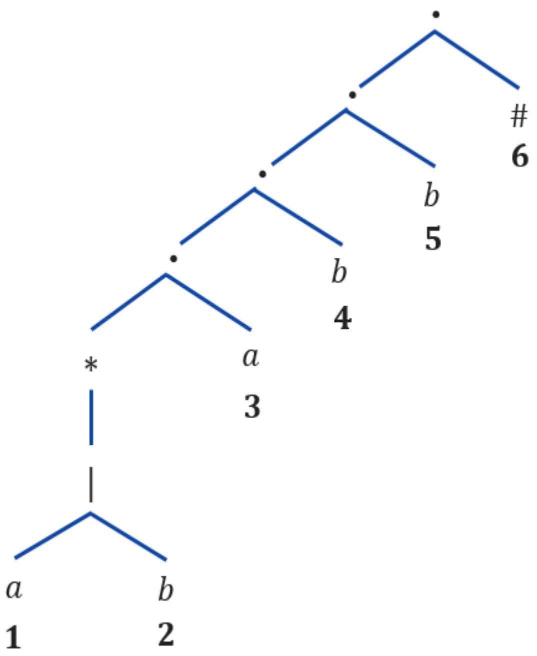
δ((1,2,3,6),a) = followpos(1) U followpos(3)

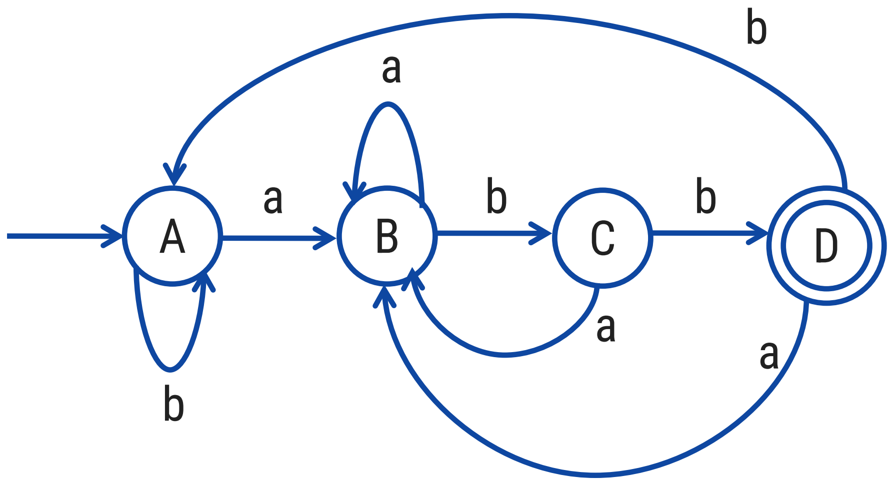
=(1,2,3) U (4) = {1,2,3,4} ----- B

δ((1,2,3,6),b) = followpos(2)

=(1,2,3) ----- A

Position	followpos
5	6
4	5
3	4
2	1,2,3
1	1,2,3





DFA

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

An Elementary Scanner Design & It’s Implementation

An Elementary Scanner Design & It’s Implementation

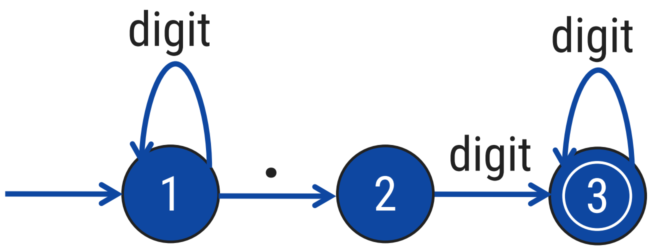
Tasks of Scanner

1. The main purpose of the scanner is to return the next input token to the parser.
2. The scanner must identify the complete token and sometimes differentiate between keywords and identifiers.
3. The scanner may perform symbol-table maintenance, inserting identifiers, literals, and constants into the tables.
4. The scanner also eliminate the white spaces.

Regular Expression: Tokens can be Specified using regular expression.

Example: id→ letter(letter | digit)*

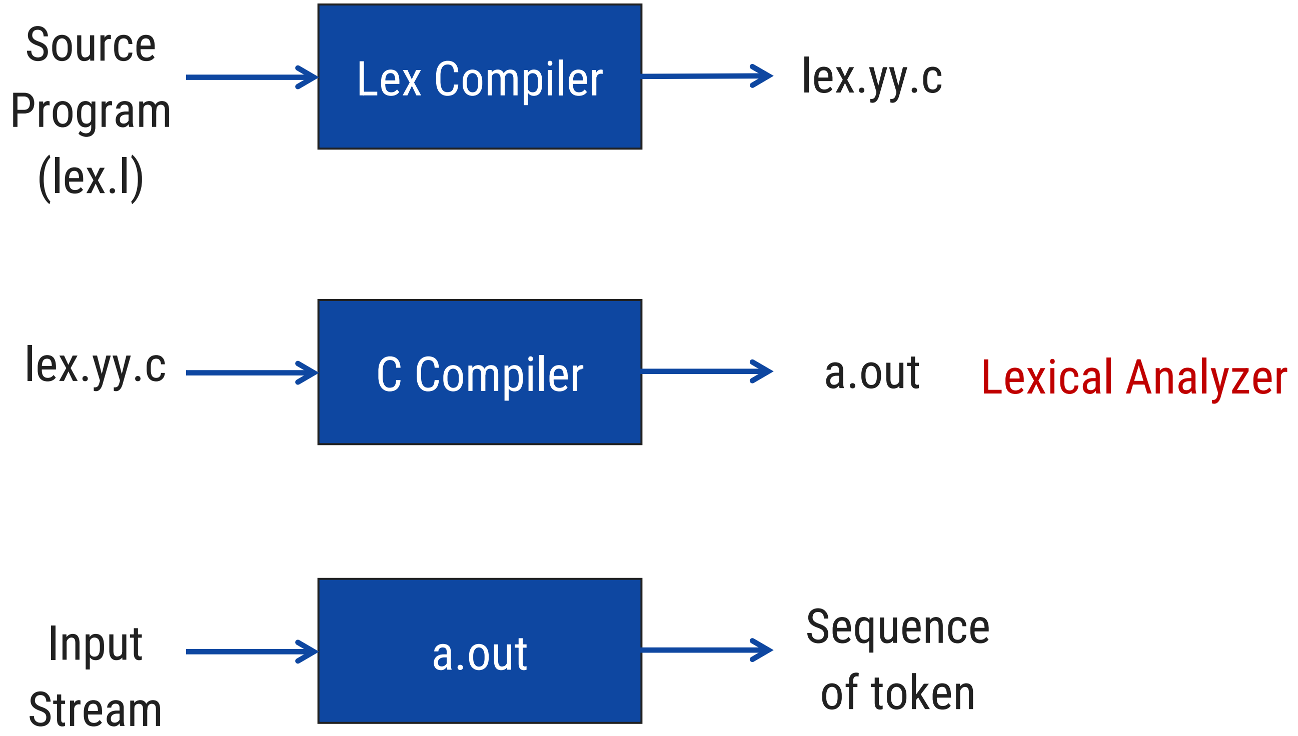
Transition Diagram: Finite-state diagrams or transition diagrams are often used to recognize a token



Implementation of Lexical Analyzer (Lex)

- Lex is tool or a language which is useful for generating a lexical Analyzer and it specifies the regular expression
- Regular expression is used to represent the patterns for a token.

Creating Lexical Analyzer with LEX



Structure of Lex Program

- Any lex program contains mainly three sections
    1. Declaration
    2. Translation rules
    3. Auxiliary Procedures

Structure of Program

Declaration → It is used to declare variables, constant & regular definition

%%

Translation rule →

Syntax: Pattern {Action}

Example:

```
%%
        pattern1   {Action1}
        pattern2   {Action2}
        pattern3   {Action3}
%%
```

%%

Auxiliary Procedures → All the function needed are specified over here.

Example: Lex Program

- Program: Write Lex program to recognize identifier, keywords, relational operator and numbers

```
/* Declaration */
%{
        /* Lex program for recognizing tokens */
%}
Letter      [a-z A-z]
Digit       [0-9]
Id          {Letter}({Letter}|{Digit})*
Numbers     {Digit}+ (.{Digit}+)? (E[+ -]? Digit+)?

/* Auxiliary Procedures */
install_id()
{
        /* procedure to lexeme into the symbol table and return a pointer */
}
```

```
/* Translation rule */
%%
{Id}        {printf("%s is an identifier",yytext);}
If          {printf("%s is a keyword",yytext);}
else        {printf("%s is a keyword",yytext);}
"<"         {printf("%s is a less then operator",yytext);}
">="        {printf("%s is a greater then equal to operator",yytext);}
{Numbers} {printf("%s is a number",yytext);}
%%
```

Input string: If year < 2021

References

Books:

1. **Compilers Principles, Techniques and Tools, PEARSON Education (Second Edition)**

   Authors: Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

2. **Compiler Design, PEARSON (for Gujarat Technological University)**

   Authors: Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman