Institute of Applied Computer Science
Lodz University of Technology

Mathematical Linguistics

Theory of automata

Alphabet & language

Alphabet Σ is a finite set of symbols.

String (a word over alphabet) is a finite set of symbols from Σ combined together.

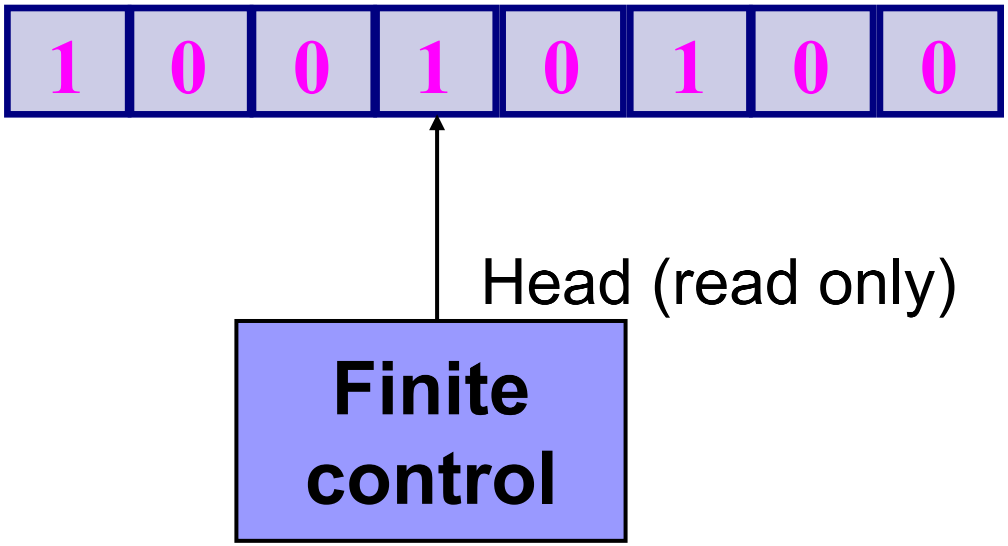
A **formal language** is a subset of finite strings of elements of the finite set which is called the alphabet.

Examples of languages

1) Set of palindromes over the alphabet lowercase Latin, $\Sigma = \{a, b, c, \ldots , z\}$
2) A set of binary numbers without insignificant zeros, $\Sigma = \{0,1\}$
3) A set of binary numbers divisible by 3, $\Sigma = \{0,1\}$
4) The set of prime decimal numbers, $\Sigma = \{0,1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Finite-state machine (FSM)

FSM - an abstract machine with a finite number of states, which reads symbols written on the tape, and changes its state according to the defined transition function



Deterministic Finite Automaton (DFA)

Definition of DFA:

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

where:

Q – finite set of states,

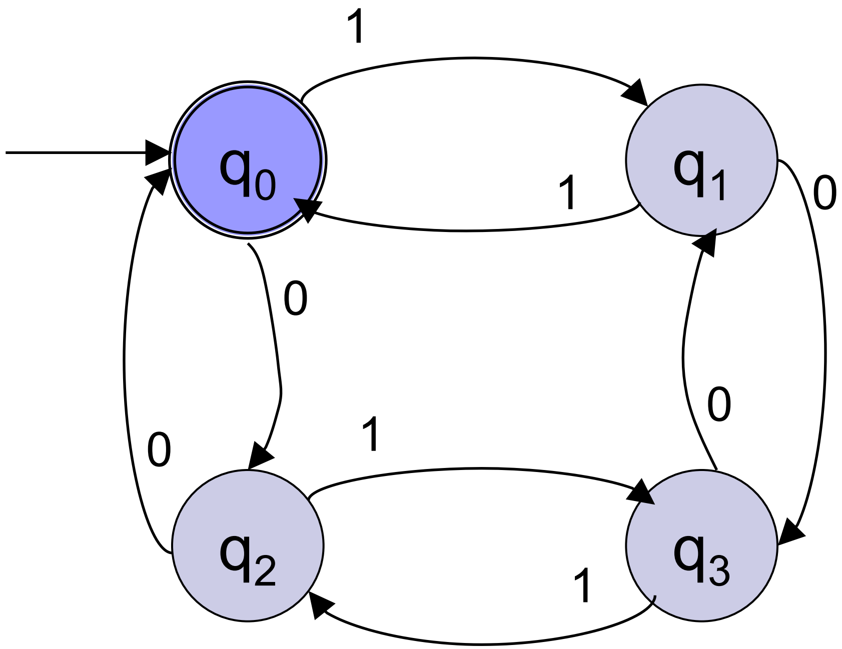
Σ – finite input alphabet,

δ – transition function mapping from Q x Σ to Q,

q_0 – initial state, $q_0 \in Q$

F – set of final states.

Deterministic Finite Automaton



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

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$

$F = q_0$

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Transition diagram and table with transition function δ(q,a)

Strings accepted by this automaton: 11, 00, 1010, 0101, 110101, 010001, … etc. These are the words belonging to L(M).

Deterministic Finite Automaton

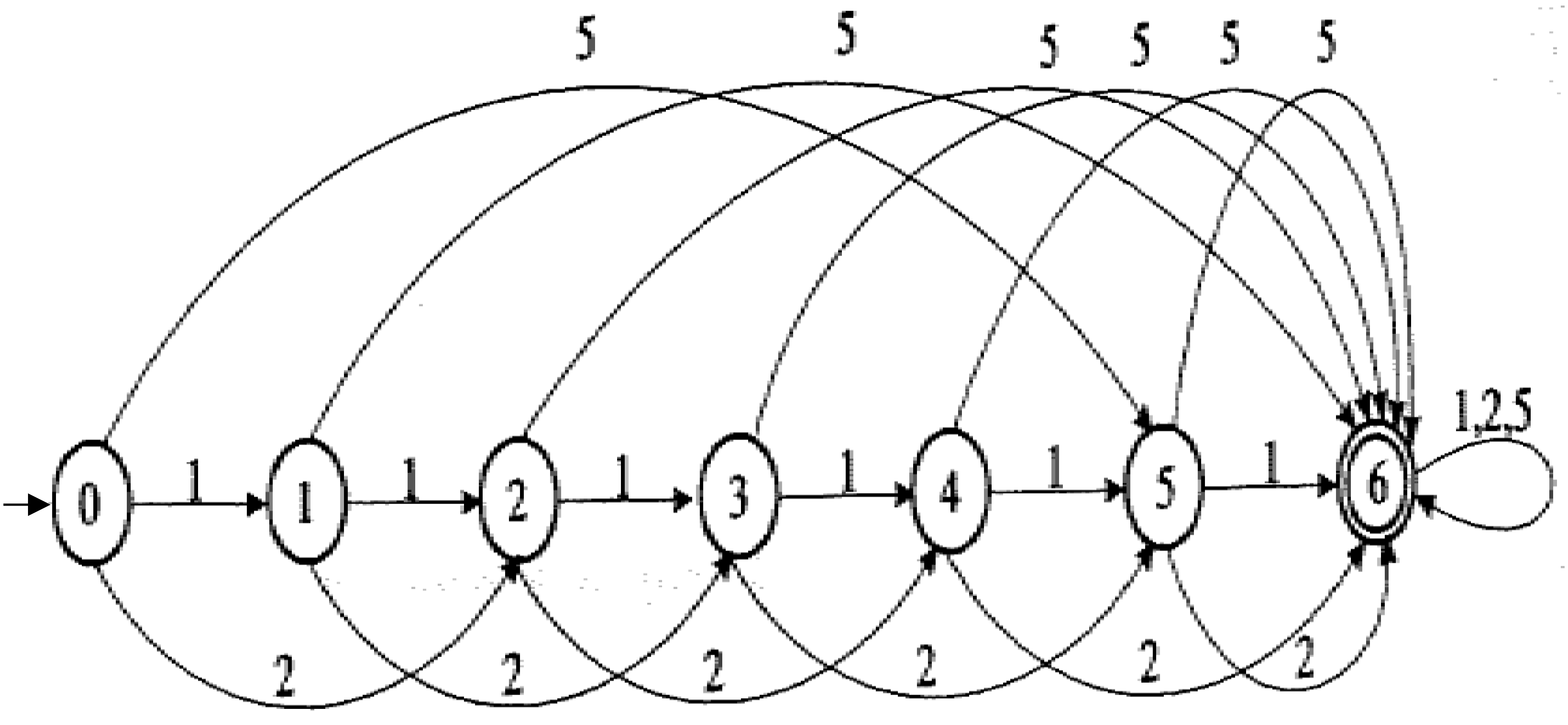
The definition of the language accepted by the DAS:

The language accepted by the DAS is a set of words over the alphabet Σ, for which the machine ends calculations in the accepting state.

Example 1

Design a finite state machine - vending machine, which gives the product, when the sum of the thrown money is equal to 6 PLN or more. The machine does not give change and accepts coins:
1 PLN, 2 PLN i 5 PLN.

Example 1



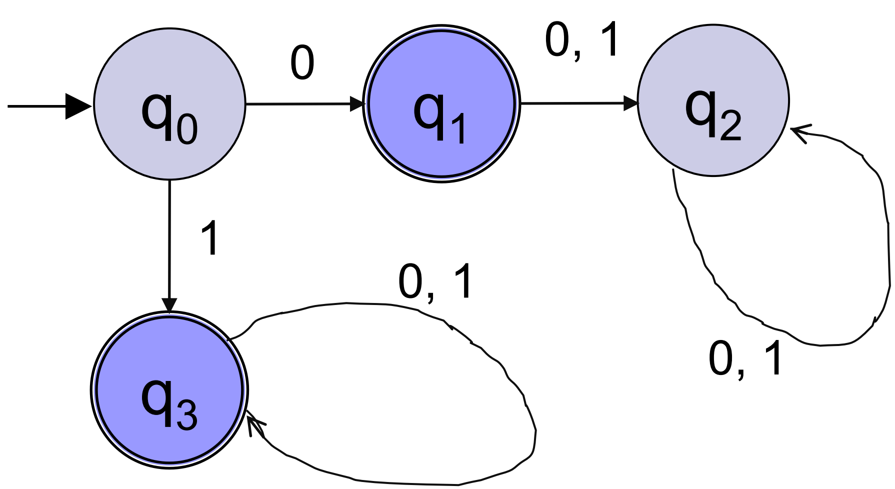
Example 2

Design a deterministic finite automaton accepting binary numbers without insignificant zeros.

Example 2

Words belonging to L(M):
0, 1, 101, 11010, …

$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$
$F = \{q_1, q_3\}$



δ	0	1
q_0	q_1	q_3
q_1	q_2	q_2
q_2	q_2	q_2
q_3	q_3	q_3

The given transition function describes the machine:

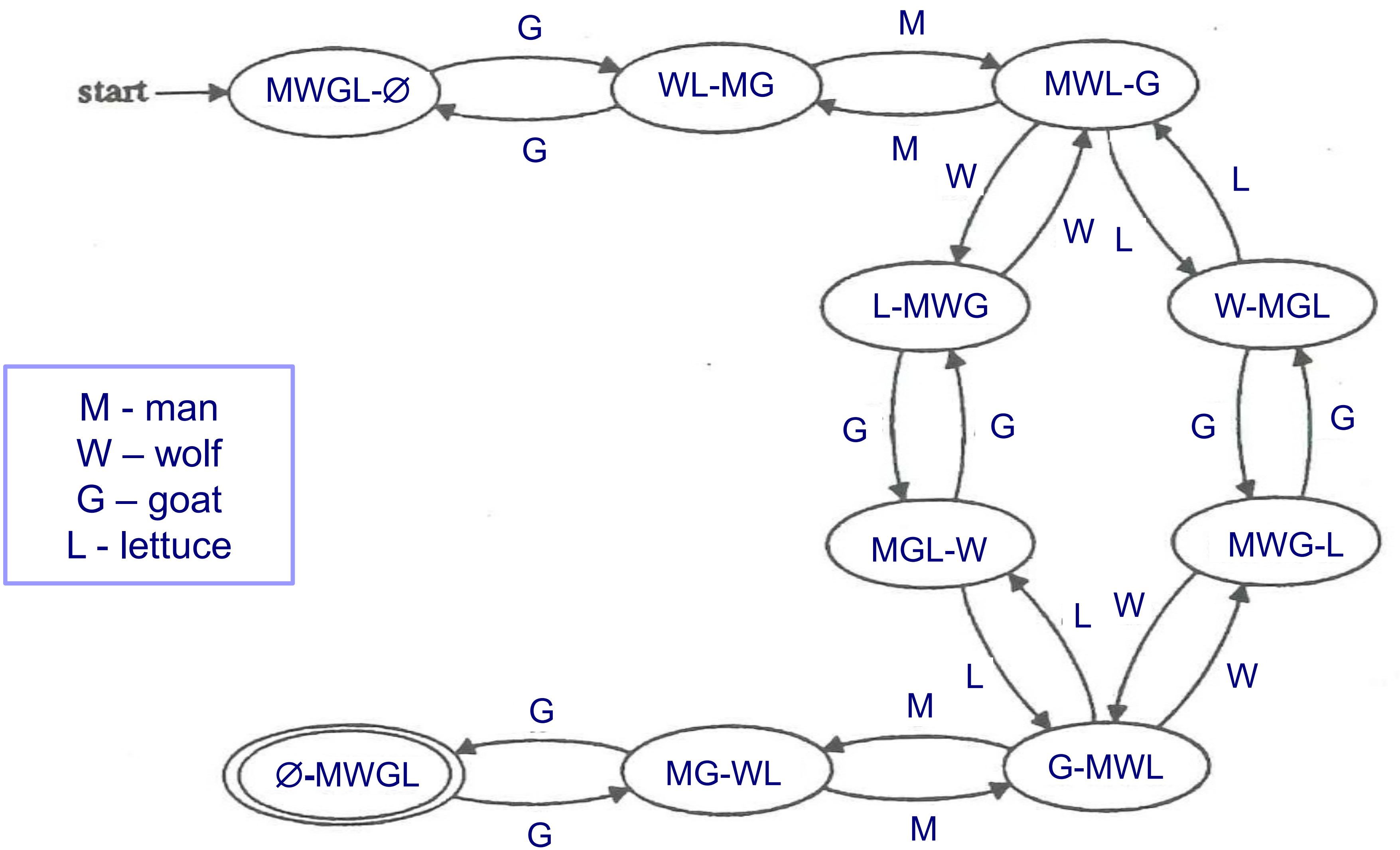
$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_1, q_3\})$$

Example

Design an algorithm solving the following problem:

1. A man has a wolf, a goat and a lettuce.
2. He wants to transport everything to the other side of a river.
3. He has a small boat and can take with him only two things at once.
4. He can not leave the sheep with the wolf, because the wolf will eat the sheep.
5. He can not leave the sheep with the lettuce, because the sheep will eat the lettuce.

Algorithm implementation



Non-deterministic Finite Automaton (NFA)

Definition of NFA:

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

where:

Q – finite set of states,

Σ – finite input alphabet,

δ – transition function mapping: $Q \times \Sigma \rightarrow 2^Q$,

q_0 – initial state, $q_0 \in Q$

F – set of final states.

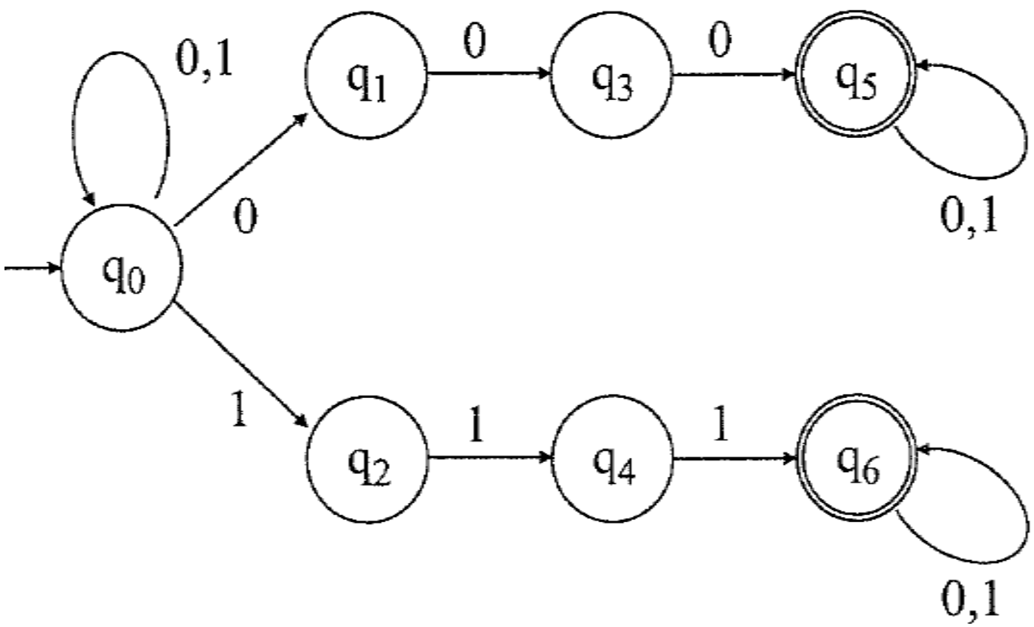
Example 3

Design a non-deterministic finite automaton that accepts the language consisting of words containing a string of three binary zeros or three ones.

Example 3

Words belonging to L(M):
000, 111, 10111, 1100010, …

$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$,
$\Sigma = \{0, 1\}$, $F = \{q_5, q_6\}$



δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	$\varnothing$
q_2	$\varnothing$	$\{q_4\}$
q_3	$\{q_5\}$	$\varnothing$
q_4	$\varnothing$	$\{q_6\}$
$q_5 \rightarrow$	$\{q_5\}$	$\{q_5\}$
$q_6 \rightarrow$	$\{q_6\}$	$\{q_6\}$

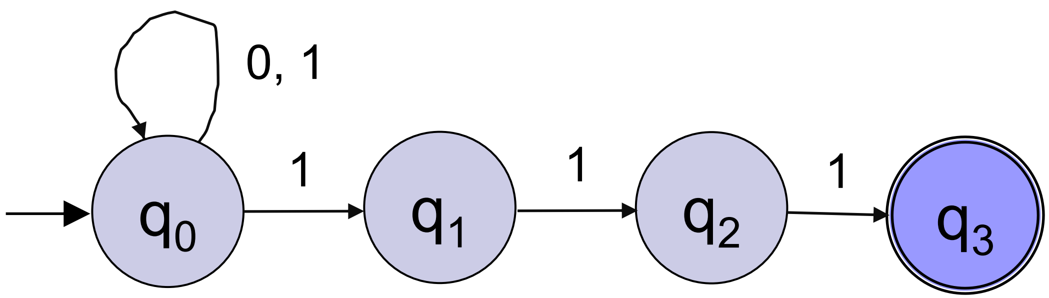
Example 4

Design a finite automata: deterministic and non-deterministic accepting binary strings ending with three ones.

Example 4

Words belonging to L(M):
111, 10111, 11010111, …

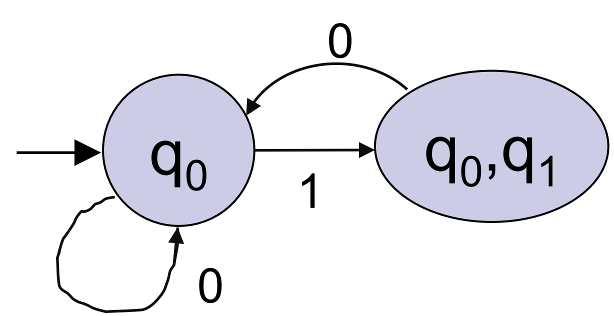
$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_0, q_1, q_2, q_3\}$,
$\Sigma = \{0, 1\}$, $F = \{q_3\}$



NFA

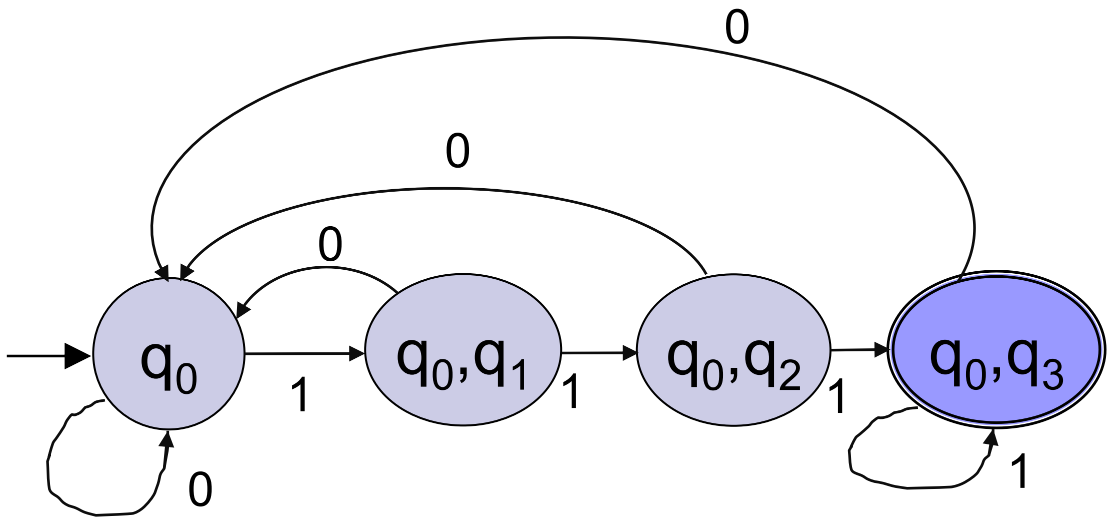
The project of the equivalent DFA

1) The initial part of the graph

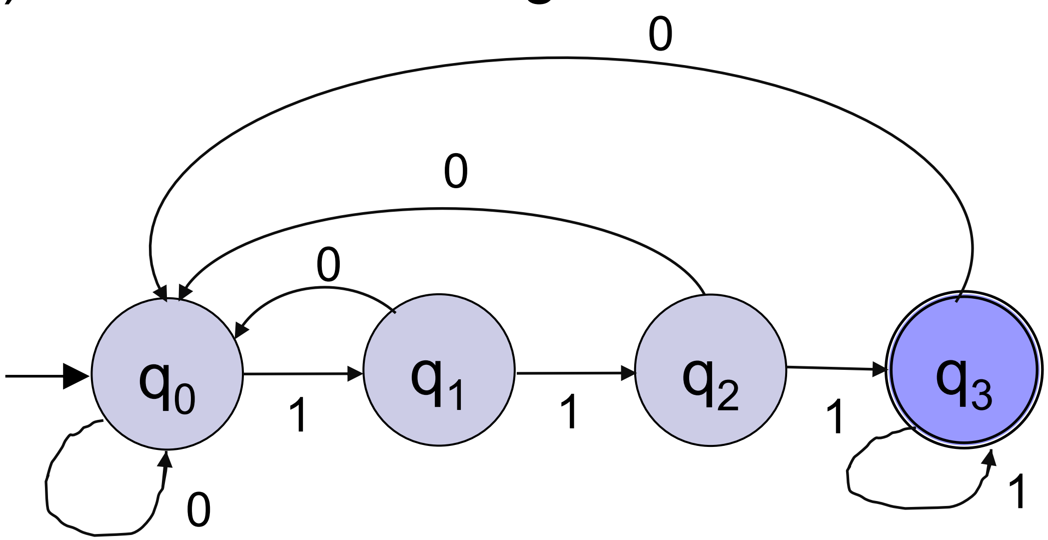


Example 4

2) The equivalent DFA



3) DFA after ordering



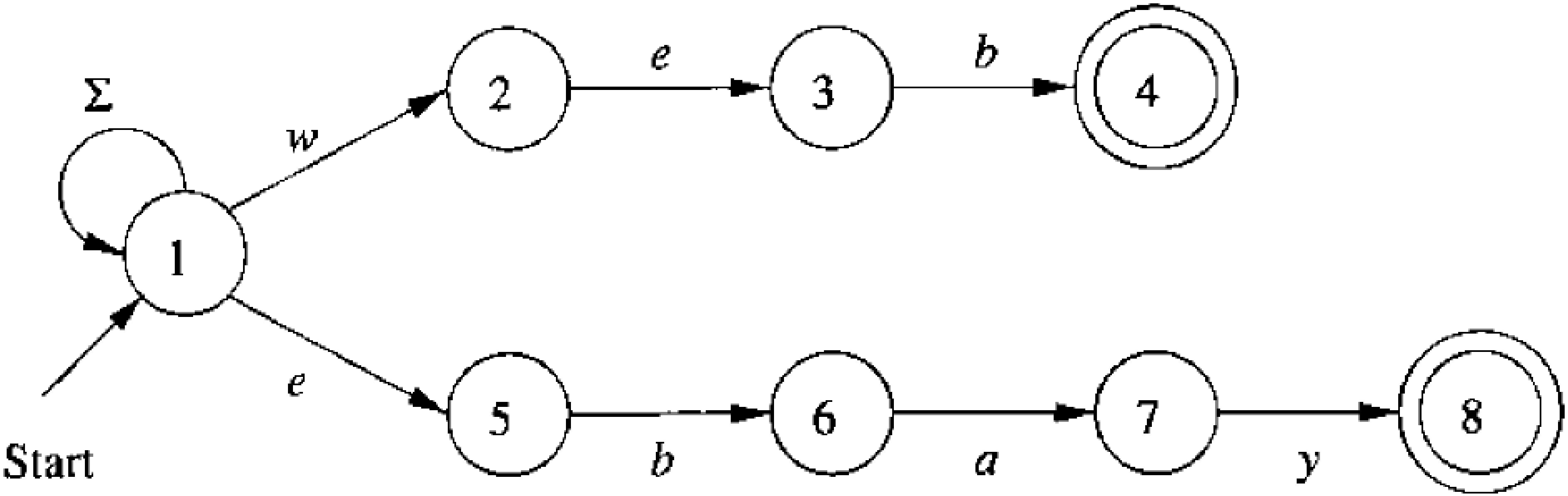
DFA

Example 5 – Text search

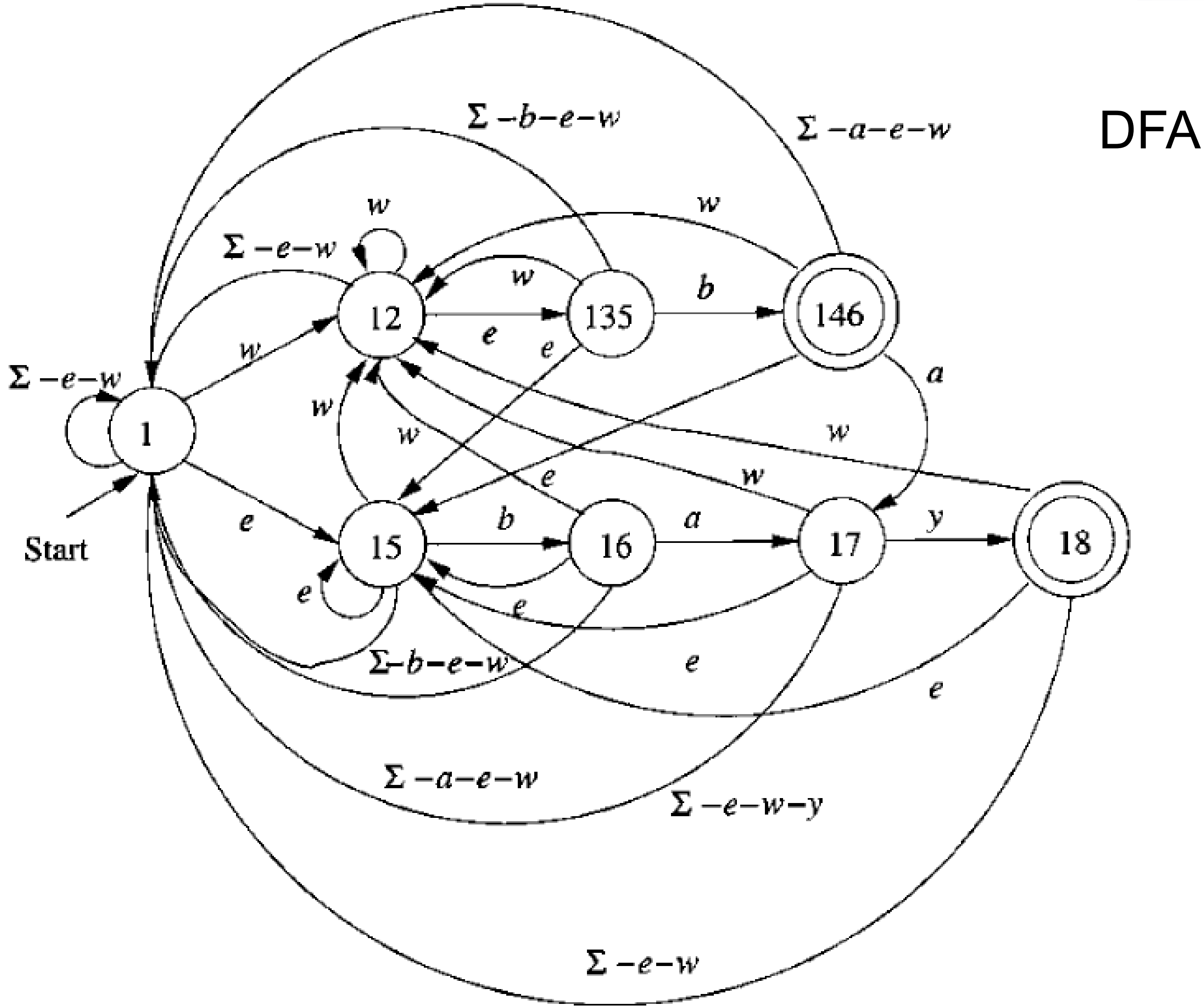
Design a finite state machine, that searches for the words „web” and „ebay” in the text.

Example 5- Text search

NFA



An NFA that searches for the words „web” and „ebay”



Conversion of the NFA that searches for the words „web" and „ebay" to a DFA

Equivalence of DFA & NFA

Theorem 1

Each deterministic finite automaton is a non-deterministic finite automaton, ie

$$DFA \subset NFA$$

Theorem 2

Let L be the language accepted by the nondeterministic finite automaton. Then there is a deterministic finite automaton accepting L.

NFA with ε-moves

Definition of NFA with ε-moves (ε-NFA):

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

where:

Q – finite set of states,

Σ – finite input alphabet,

δ – transition function mapping:

$Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$, (where: ε -empty word),

q_0 – initial state, $q_0 \in Q$

F – set of final states.

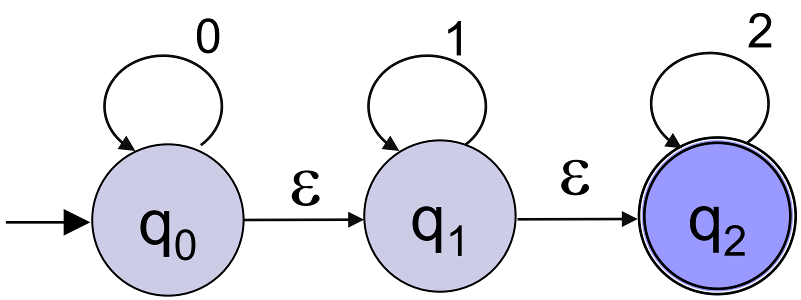
Example 5

Design a non-deterministic finite automaton that accepts the language consisting of words that contain any number of zeros, followed by any number of ones, and then any number of twos.

Example 5

Words belonging to L(M):
012, 002, 112, 011222, …

$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_0, q_1, q_2\}$,
$\Sigma = \{0,1,2\}$, $F = \{q_2\}$



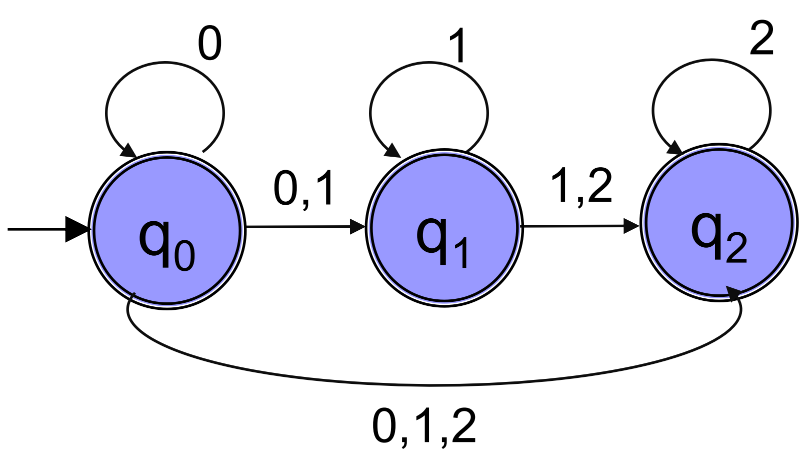
NFA with ε-moves

δ	0	1	2	ε
q_0	$\{q_0\}$	$\varnothing$	$\varnothing$	$\{q_1\}$
q_1	$\varnothing$	$\{q_1\}$	$\varnothing$	$\{q_2\}$
$\mathbf{q_2}$	$\varnothing$	$\varnothing$	$\{q_2\}$	$\varnothing$

Example 5

The project of the equivalent NFA

δ	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	$\varnothing$	$\{q_1, q_2\}$	$\{q_2\}$
$\mathbf{q_2}$	$\varnothing$	$\varnothing$	$\{q_2\}$



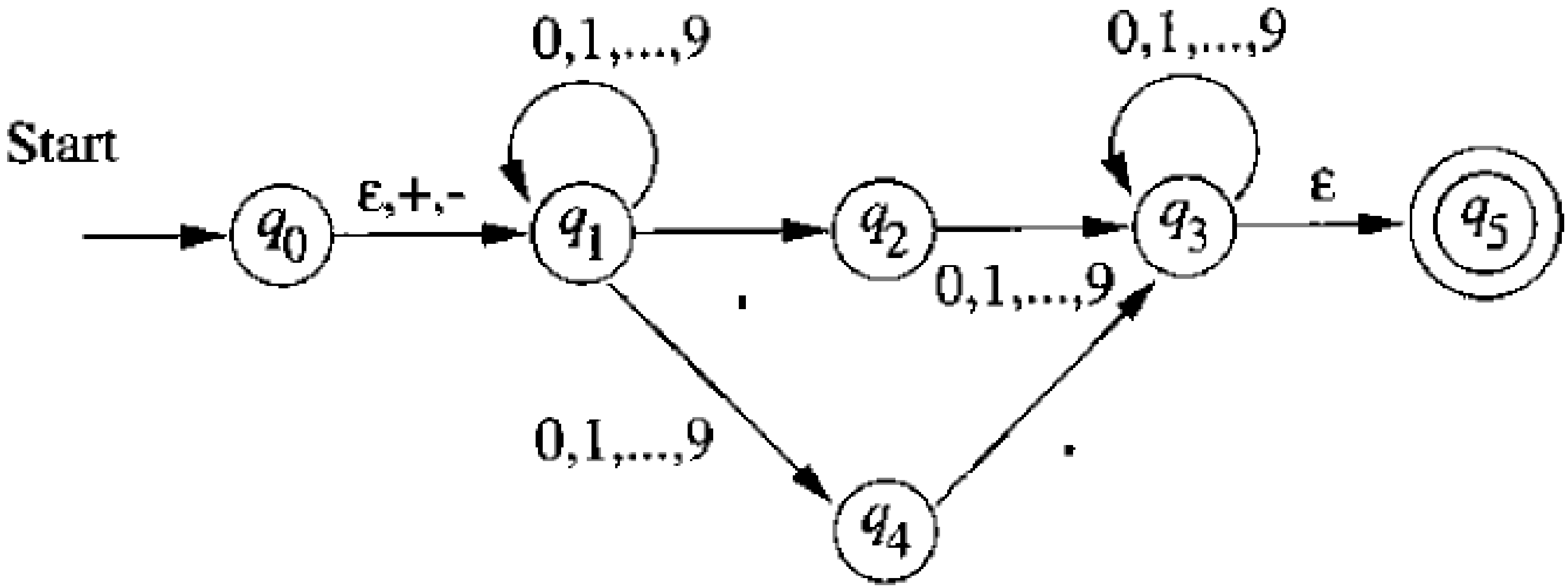
Example

Design a finite automaton that accepts decimal numbers consisting of:

1. An optional + or − sign,
2. A string of digits,
3. A decimal point, and
4. Another string od digits.

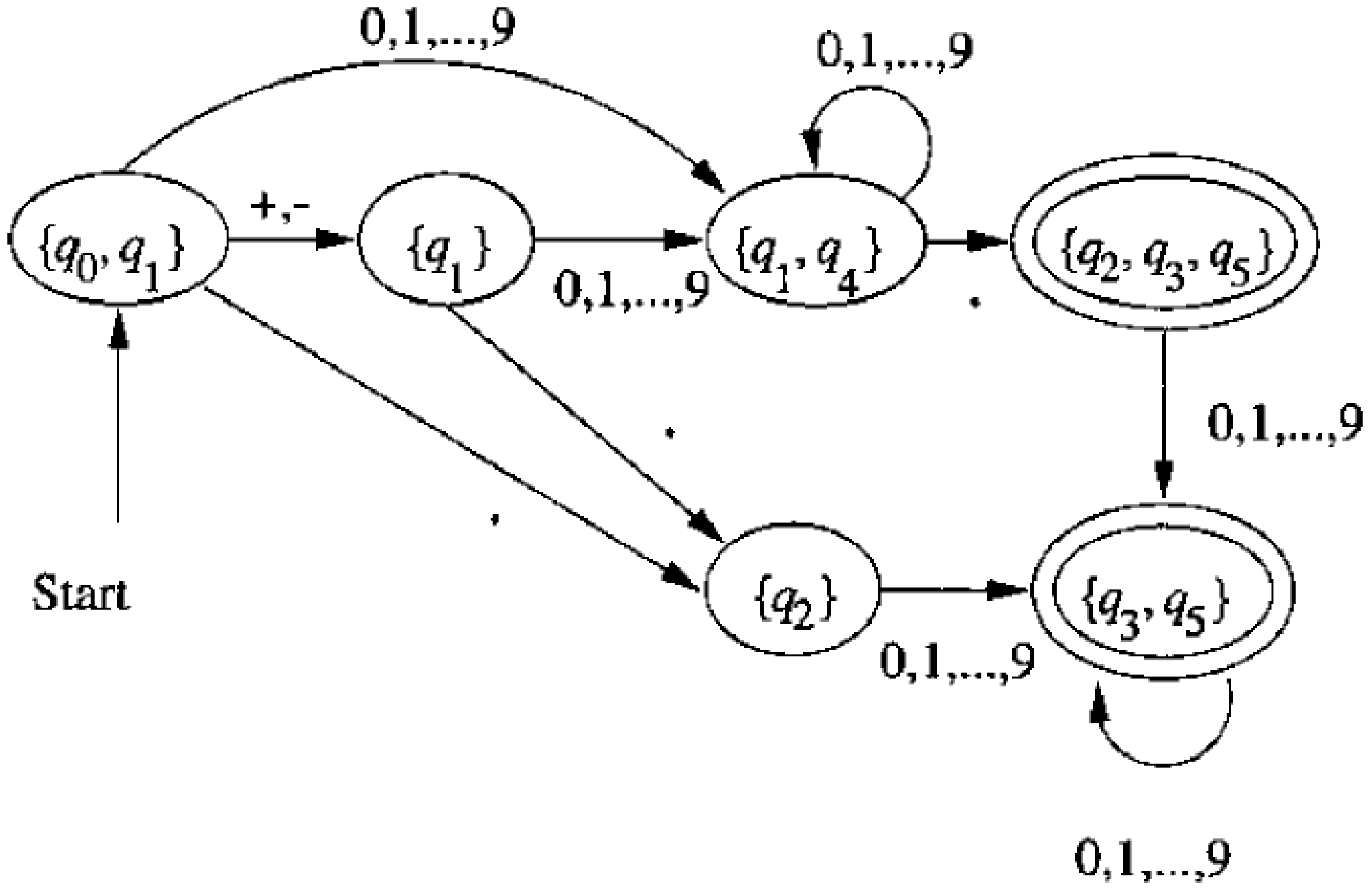
Either this string of digits (4) or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.

Example



An ε-NFA accepting decimal numbers.

Example



An NFA accepting decimal numbers.

Equivalence of NFA-ε and NFA

Theorem 3

If the language L is accepted by a non-deterministic finite automaton with ε-moves, it is also acceptable to the NFA without ε-moves.

Regular expressions

A regular expression is an expression that describes a set of strings.

Regular expressions are used by many text editors, utilities, and programming languages to search and manipulate text based on patterns.

The origins of regular expressions come from automata theory and formal language theory.

Regular expressions describe regular languages in formal language theory.

Regular expressions - definition

Given a finite alphabet Σ, the following regular expressions over Σ are defined:

1. **∅** is regular expression denoting *empty set*.
2. **ε** is regular expression denoting set {ε} (*empty string*).
3. For each *a* in Σ, ***a*** is regular expression denoting set {a} (*literal character*).
4. If ***r*** and ***s*** are regular expressions denoting *R* and *S* languages respectively, then (***r*** + ***s***), (***rs***) and (***r****) are regular expressions denoting sets: $R \cup S$ (set union), *RS* (*concatenation*) and R^* (*Kleene star*) respectively.

Examples

Given $R = \{a, b\}$

$S = \{c, d\}$

then

$R \cup S = \{a, b, c, d\}$ (set union),

$RS = \{ac, ad, bc, bd\}$ (*concatenation*)

$R^* = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, aab, aba, \ldots\}$

(*Kleene star*)

Regular expressions (RegEx)- notation

Metacha-racter	Description
.	any single character
[abc]	a single character that is contained within the brackets ("*a*", "*b*" or "*c*")
[^abc]	a single character that is not contained within the brackets
[a-z0-9]	any single character from a given range
\w	any single character, such as letter, digit or underline
\W	any single character, other than letter, digit and underline
\s	any single white character
\S	any single character other than white one
\d	any digit
\D	any single character other than digit

<table>
<tr><th>Metacha-racter</th><th>Description</th></tr>
<tr><td>^</td><td>matches the starting position within the string</td></tr>
<tr><td>$</td><td>matches the ending position of the string</td></tr>
<tr><td>*</td><td>repeats the previous item zero or more times</td></tr>
<tr><td>+</td><td>repeats the previous item once or more</td></tr>
<tr><td>?</td><td>makes the preceding item optional</td></tr>
<tr><td>{n}</td><td>repeats the previous item exactly n times</td></tr>
<tr><td>{n,}</td><td>repeats the previous item at least n times</td></tr>
<tr><td>{n,m}</td><td>repeats the previous item between n and m times</td></tr>
<tr><td>|</td><td>alternate</td></tr>
<tr><td>( )</td><td>grouping</td></tr>
<tr><td>\</td><td>a backslash escapes special characters to suppress their special meaning</td></tr>
<tr><td>. $ ^ { [(|)] } * + ? \</td><td>special characters</td></tr>
</table>

Examples

Post code checking: ^\d{2}-\d{3}$

www adress checking: ^www\.[-\w]+(\.[-\w]+)+$

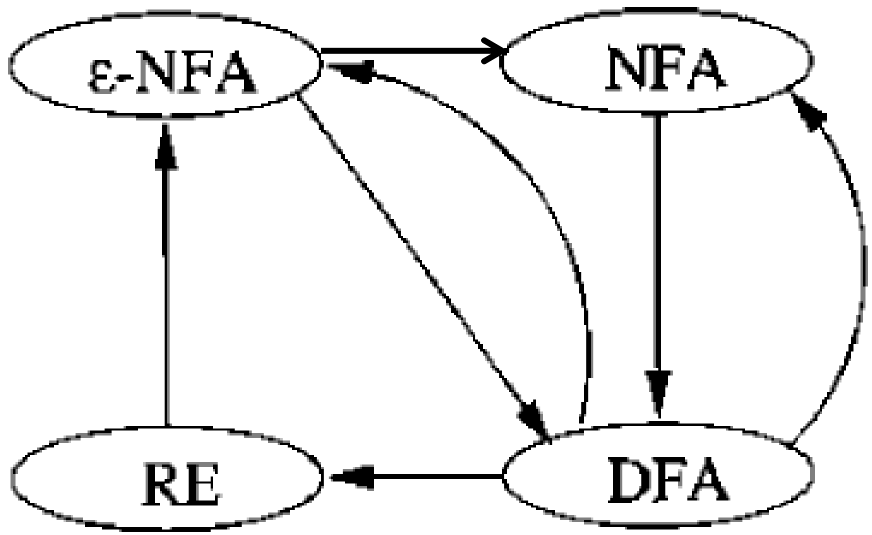
IP adress checking:

^0*(25[0-5]|2[0-4]\d|1?\d\d?)(\.0*(25[0-5]|2[0-4]\d|1?\d\d?)){3}$

Equivalence of regular expressions and finate automata

1. Every language L accepted by a finite automaton is also defined by regular expression.
2. Every language L defined by regular expression is also defined by a finite automaton.

Equivalence of regular expressions and finate automata



Different notations of regular languages.

Converting regular grammar into regular expression

Grammar defining e-mail adress:

S::= A@A.W
A::= W{.W}
W::= L{L}
L ::= a | b | c | d | e | … | x | y | z

Grammar after reduction:
S::= L{L}{. L{L}} @ L{L}{. L{L}}. L{L}
L ::= a | b | c | d | e | … | x | y | z

Converting regular grammar into regular expression

Grammar defining e-mail adress:

S::=(a|b|c|…|z){(a|b|c|…|z)}{.(a|b|c|…|z){(a|b|c|…|z)}}
@(a|b|c|…|z){(a|b|c|…|z)}{.(a|b|c|…|z){(a|b|c|…|z)}}
.(a|b|c|…|z){(a|b|c|…|z)}

Regular expression defining e-mail adress:

^[a-z]+(\.[a-z]+)*@[a-z]+(\.[a-z]+)*\.[a-z]+$

Turing Machine

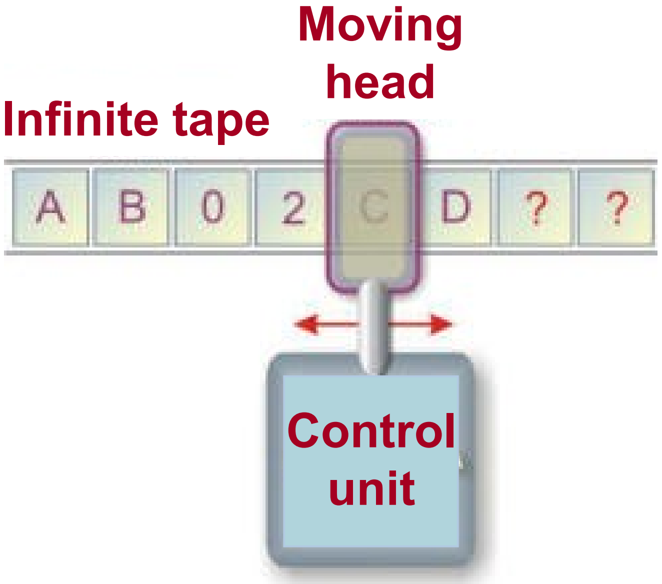
Alan Turing

Turing Machine is a very simple abstractive mathematical model of a computer.

Turing Machine

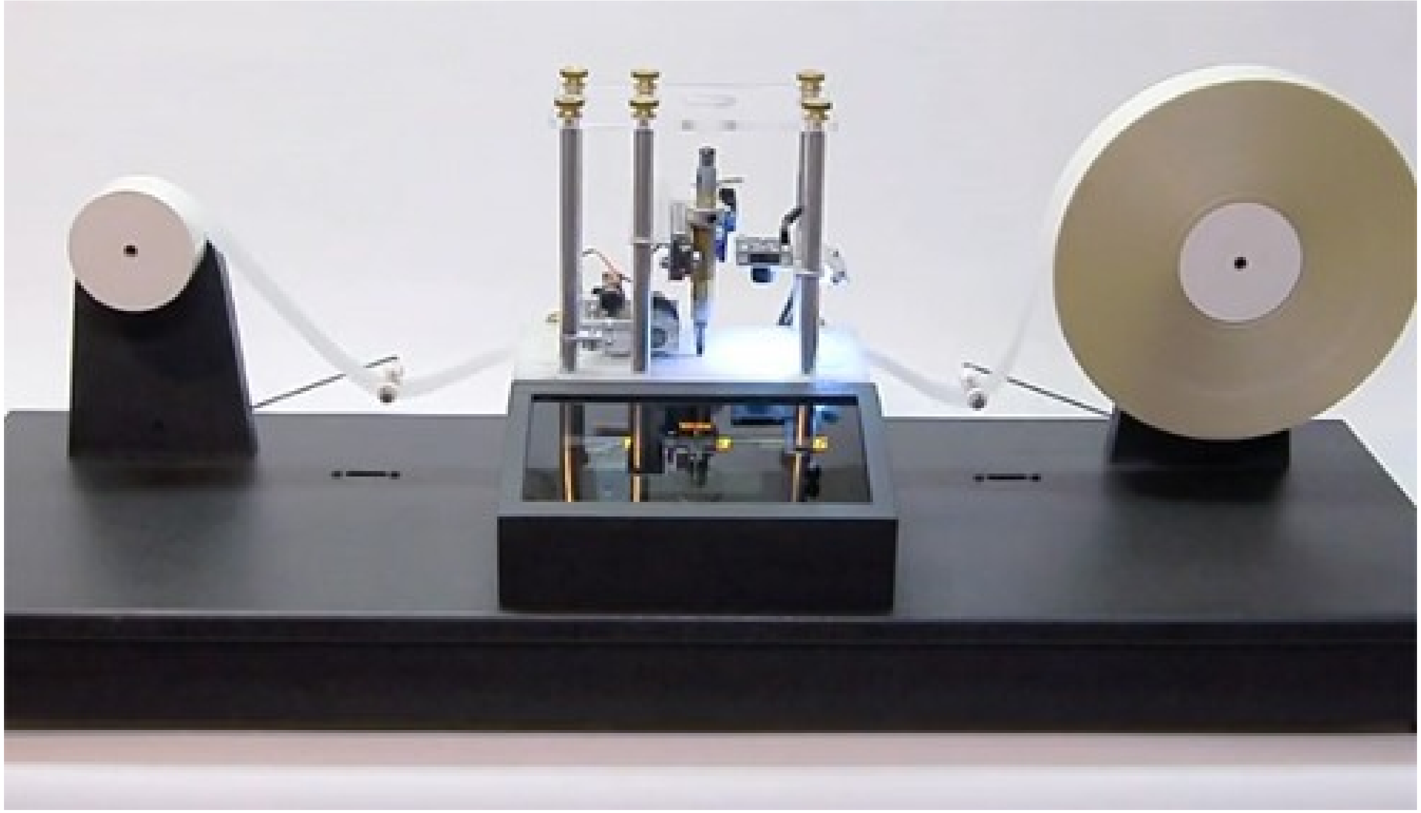
Basic components:

- Infinite tape – memory
- Moving head – input/output system
- Control unit – processor



Turing Machine

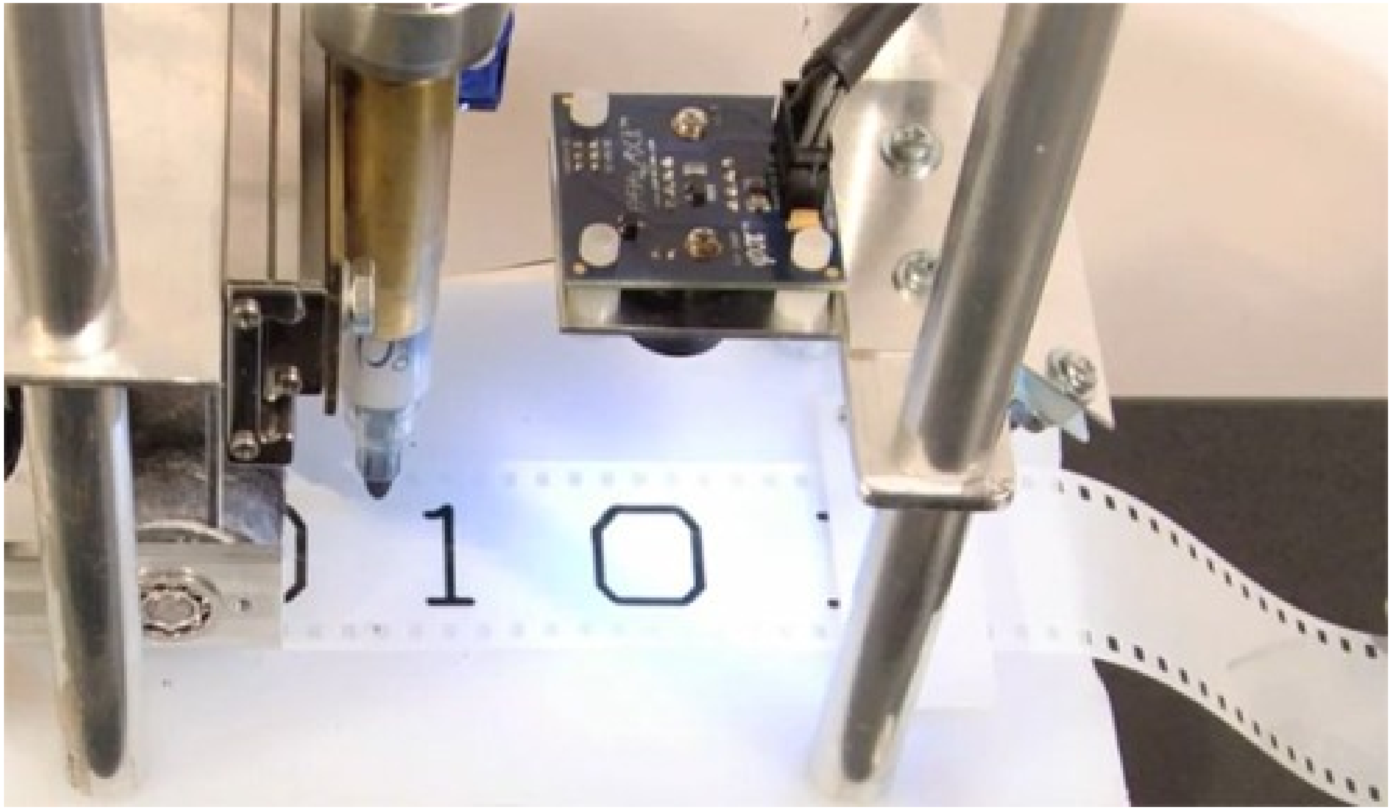
Model of the Turing machine built by Mike Davey



http://gadzetomania.pl/2010/03/27/wykonana-metoda-chalupnicza-maszyna-turinga-wideo

Turing Machine

Model of the Turing machine built by Mike Davey



http://gadzetomania.pl/2010/03/27/wykonana-metoda-chalupnicza-maszyna-turinga-wideo

Turing Machine – basic model

Definition of Turing Machine:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Theta, F)$$

where:

Q – finite set of states,

Σ – finite input alphabet, set of input symbols,

Γ – tape alphabet – finite set of valid tape symbols,

$$\Sigma \subset \Gamma - \{\Theta\}$$

Θ – empty symbol belonging to Γ,

δ – transition function, δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$,

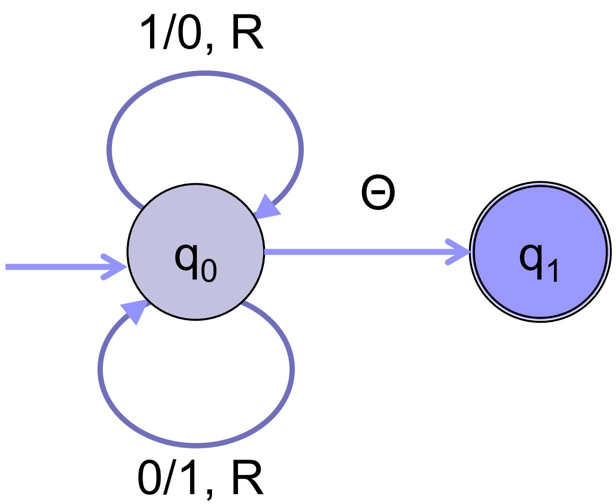
where L and R symbols meaning the head movement direction: left or right.

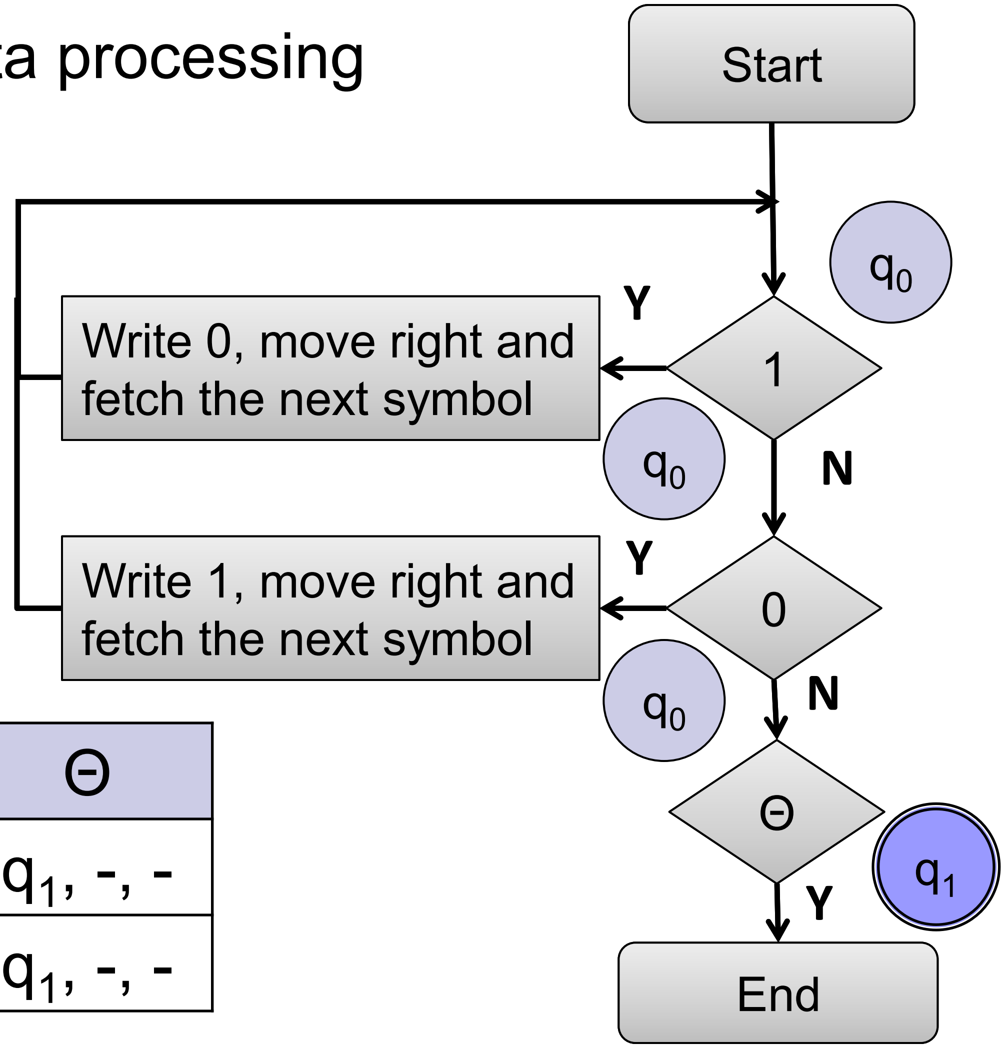
q_0 – initial state, which belongs to Q

F – set of final states.

Turing Machine – program example

Negation of binary value – data processing



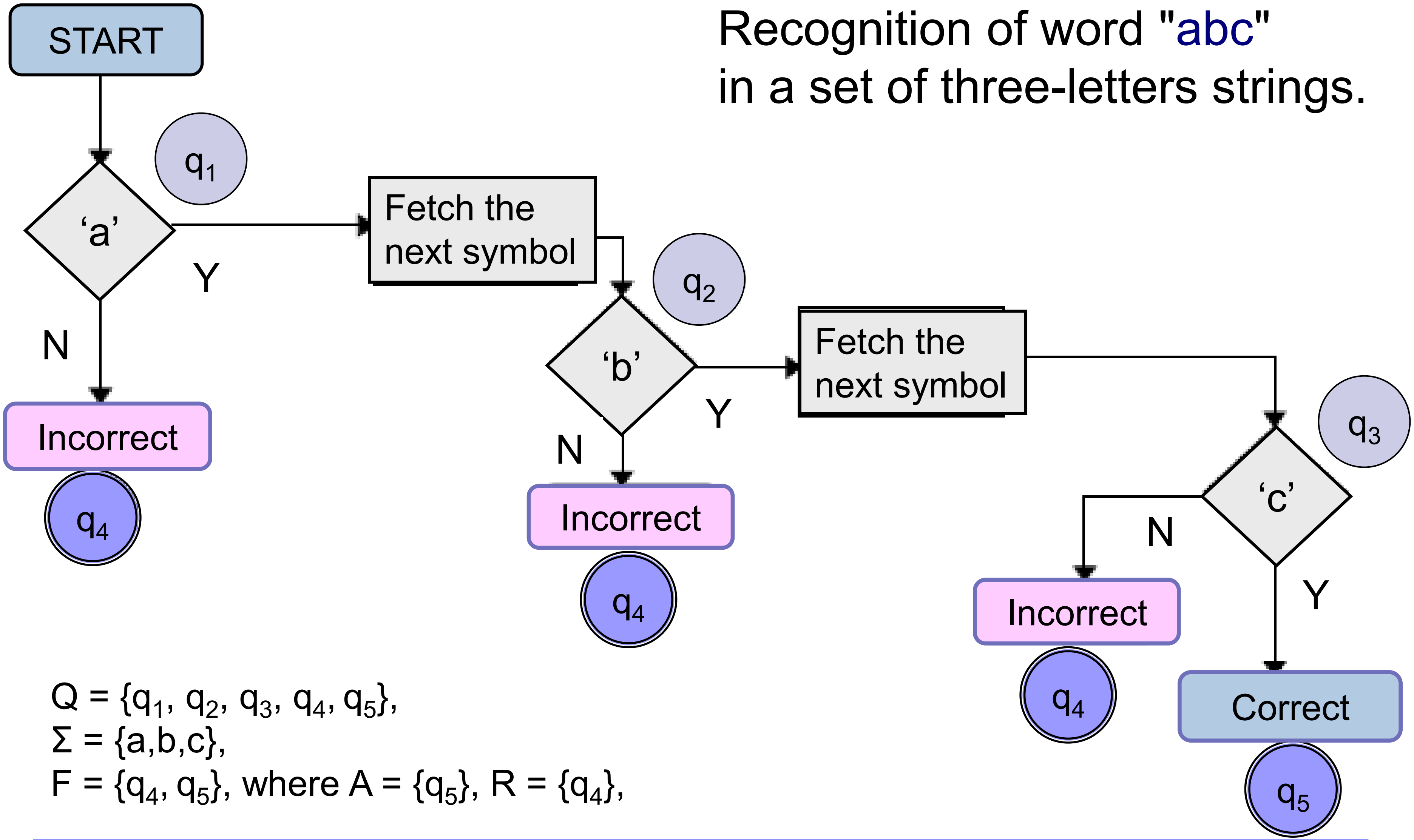


δ	0	1	Θ
q_0	q_0 ,1, R	q_0, 0, R	q_1, -, -
q_1	q_1, -, -	q_1, -, -	q_1, -, -

$\Sigma = \{0,1\}$, $\Gamma = \{0,1,\Theta\}$, $Q = \{q_0,q_1\}$, $F = \{q_1\}$

Turing Machine – example 2

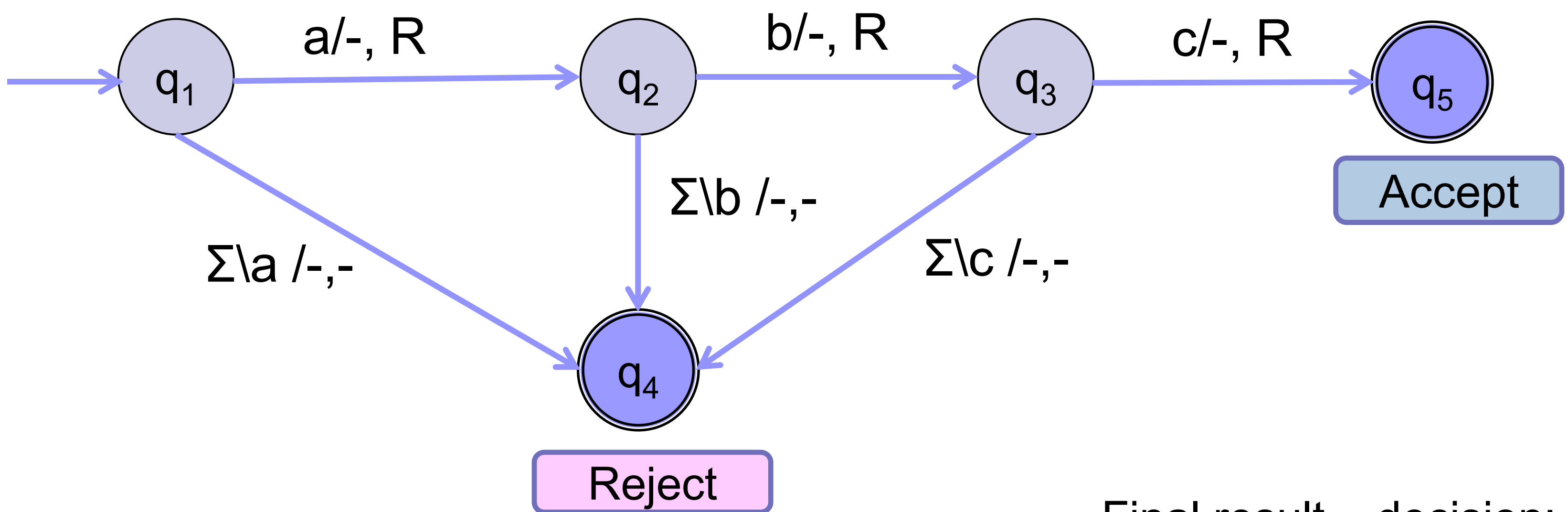
Recognition of word "abc"
in a set of three-letters strings.



$Q = \{q_1, q_2, q_3, q_4, q_5\}$,
$\Sigma = \{a,b,c\}$,
$F = \{q_4, q_5\}$, where $A = \{q_5\}$, $R = \{q_4\}$,

Turing Machine – example 2

Recognition of word "abc„ in a set of three-letters strings - decission.



$Q = \{q_1, q_2, q_3, q_4, q_5\}$,
$\Sigma = \{a,b,c\}$,
$F = \{q_4, q_5\}$, where $A = \{q_5\}$, $R = \{q_4\}$,

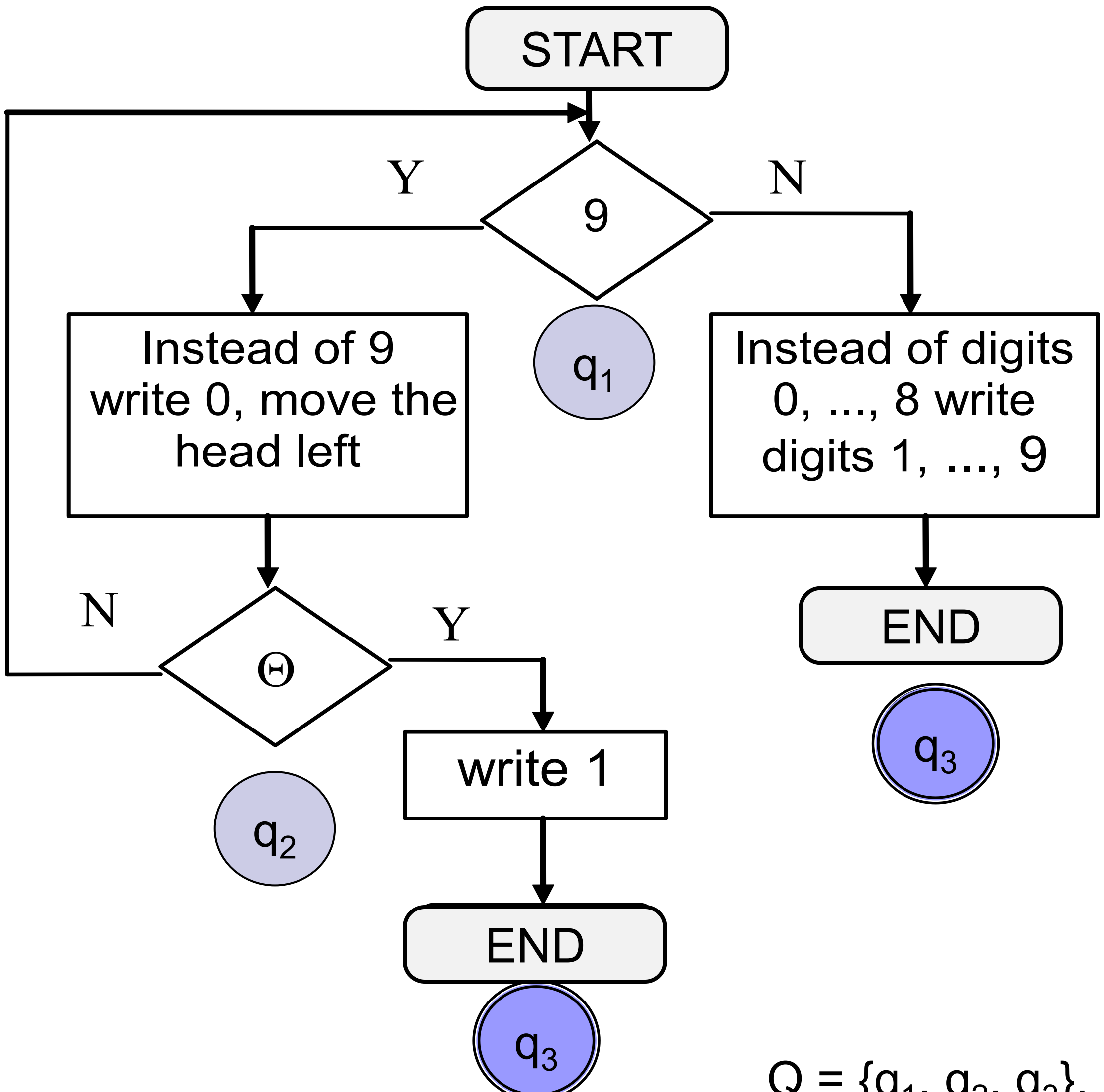
Final result – decision:
A – accept
R – reject

Turing Machine – example 2

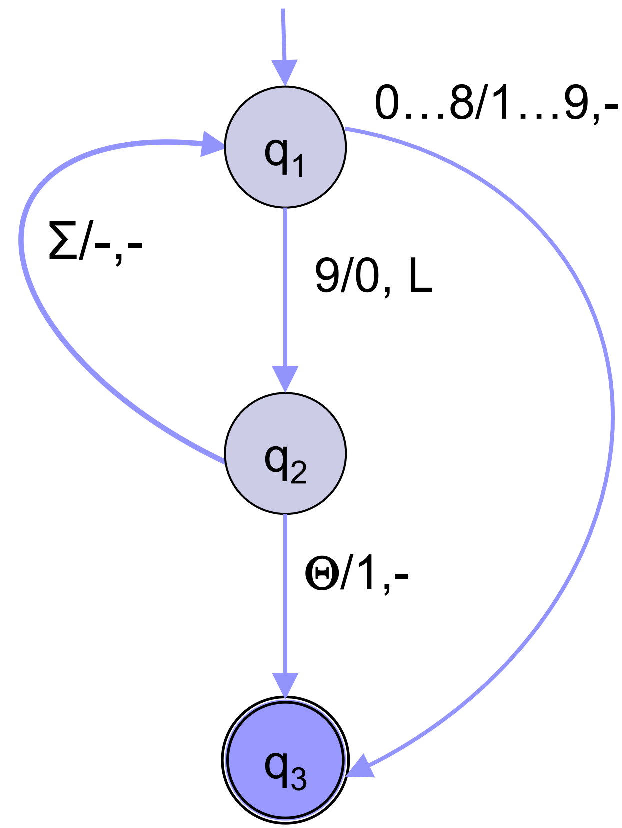
δ	a	b	c
q_1	q_2 , -, R	q_4 , -, -	q_4 , -, -
q_2	q_4 , -, -	q_3 , -, R	q_4 , -, -
q_3	q_4 , -, -	q_4 , -, -	q_5 , -, -
q_4	q_4 , -, -	q_4 , -, -	q_4 , -, -
q_5	q_5 , -, -	q_5 , -, -	q_5 , -, -

where: R – move the head right and fetch the next symbol,
q_4 – rejecting state, q_5 – accepting state.

Turing Machine – example 3



Incrementing a number by 1 - calculations



$Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$, $F = \{q_3\}$

Turing Machine – example 3

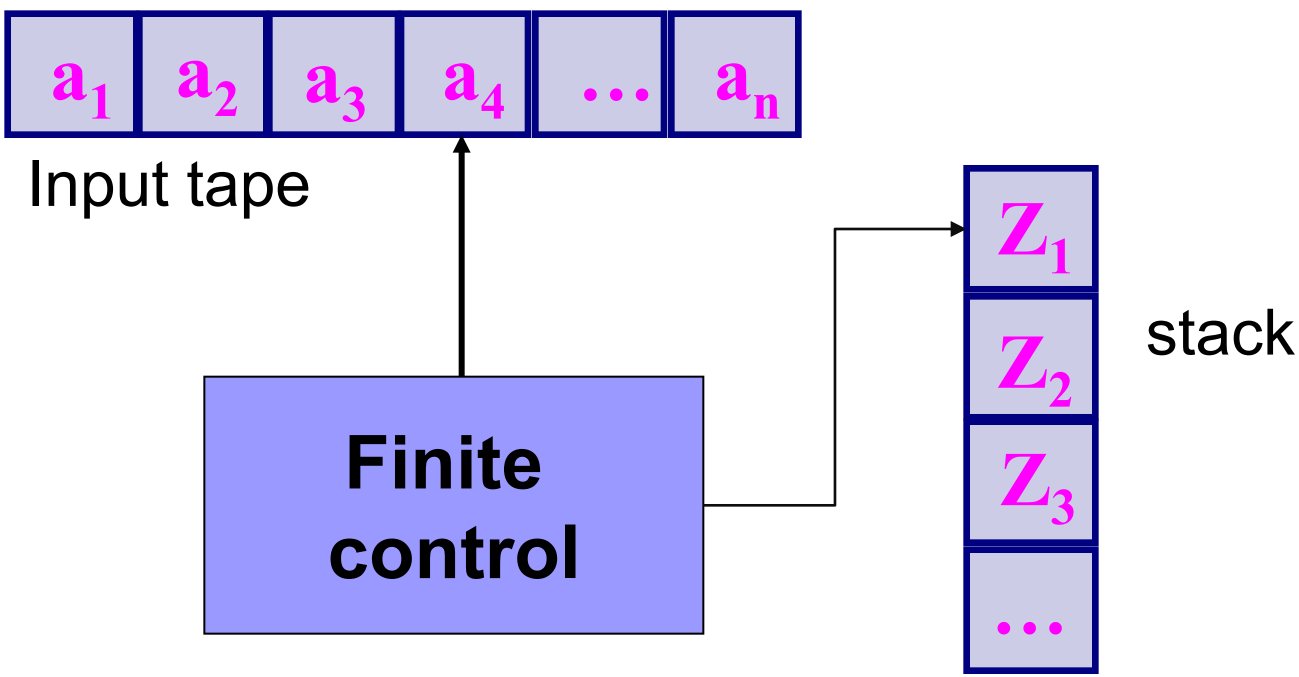
$Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$, $F = \{q_3\}$

δ	9	Θ	0	1	…	8
q_1	q_2,0, L	-	q_3,1,-	q_3,2,-	…	q_3,9,-
q_2	q_1,-,-	q_3,1,-	q_1 ,-,-	q_1 ,-,-	…	q_1 ,-,-
q_3	q_3 ,-,-	q_3 ,-,-	q_3 ,-,-	q_3 ,-,-	…	q_3 ,-,-

Machine State	q_1	q_2	q_1	q_3
Tape State	89	80	80	90
	_↑	↑_	↑_	↑_

Pushdown automaton (PDA)

Pushdown automaton is a finite automaton equipped additionally with a stack control. PDA is a subclass of Turing machines.



Pushdown automaton (PDA)

Definition of PDA:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where:

Q – finite set of states

Σ – finite input alphabet, set of input symbols

Γ – stack alphabet – finite set of valid stack symbols

Z_0 – initial symbol belonging to Γ

q_0 – initial state, which belongs to Q

F – set of final states

δ – transition function, δ:

$$Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^*$$

and Γ^* denotes the set of strings over alphabet Γ

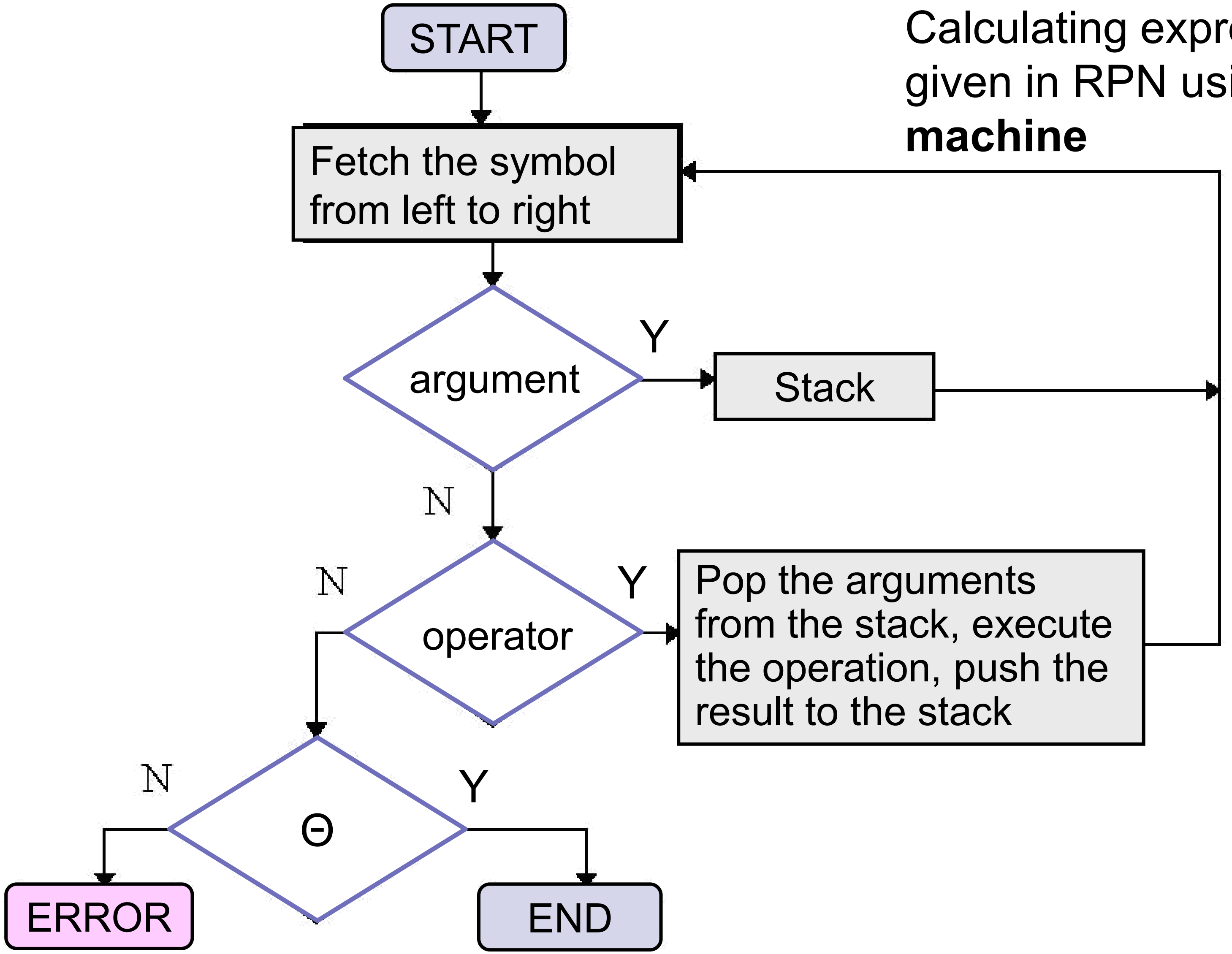
RPN – Reverse Polish Notation

Conventional infix notation	Postfix notation (RPN)
x+y	*xy+*
(*x–y*)+*z*	*xy–z+*
x– (*y+z*)	*xyz+–*
*x**(*y+z*)**w*	*xyz+*w**

Jan Łukasiewicz

RPN – Reverse Polish Notation

Calculating expression value given in RPN using a **stack machine**



RPN – Reverse Polish Notation - example

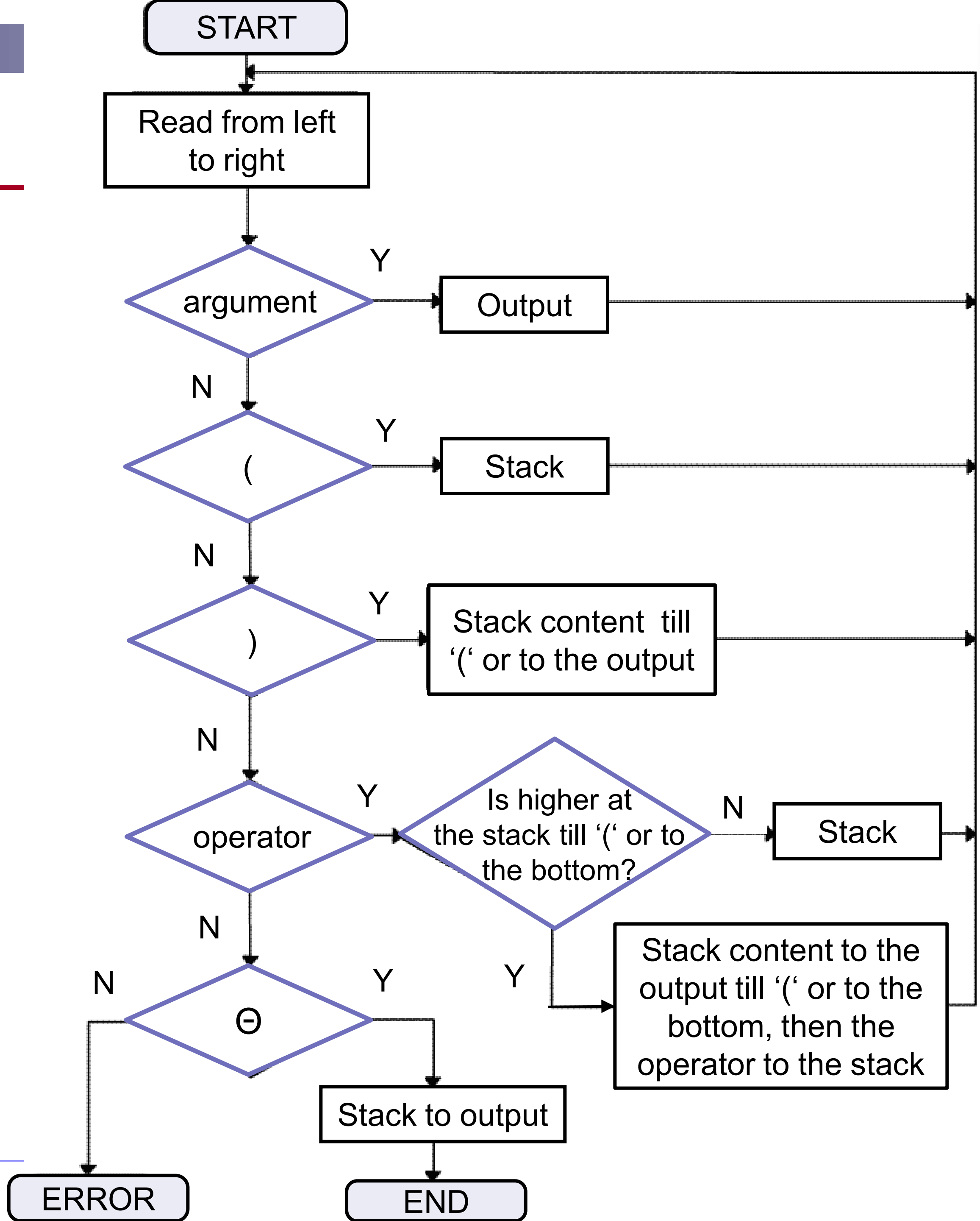
Calculating expression value given in RPN

3 5 * 1 + 2 /

Input	Stack	Output
3	3	
5	3 5	
*	15	
1	15 1	
+	16	
2	16 2	
/	8	
Θ		8

RPN

Conversion of an expression from conventional to RPN notation using a **stack machine**



RPN – Reverse Polish Notation - example

Conversion of an expression from conventional to RPN notation

(3*5+1)/2

Input	Stack	Output
(	(	
3	(	3
*	(*	
5	(*	5
+	(+	*
1	(+	1
)		+
/	/	
2	/	2
Θ		/

3 5 * 1 + 2 /

Literatura

1. Hopcroft J.E., Ullman J.D.: *Wprowadzenie do teorii automatów, języków i obliczeń.* PWN, Warszawa, 2003.
2. Homenda W.: *Elementy lingwistyki matematycznej i teorii automatów.* Oficyna Wydawnicza Politechniki Warszawskiej, W-wa 2005.
3. Krasiński T.: *Automaty i języki formalne.* Uniwersytet Łódzki, Łódź, 2005.