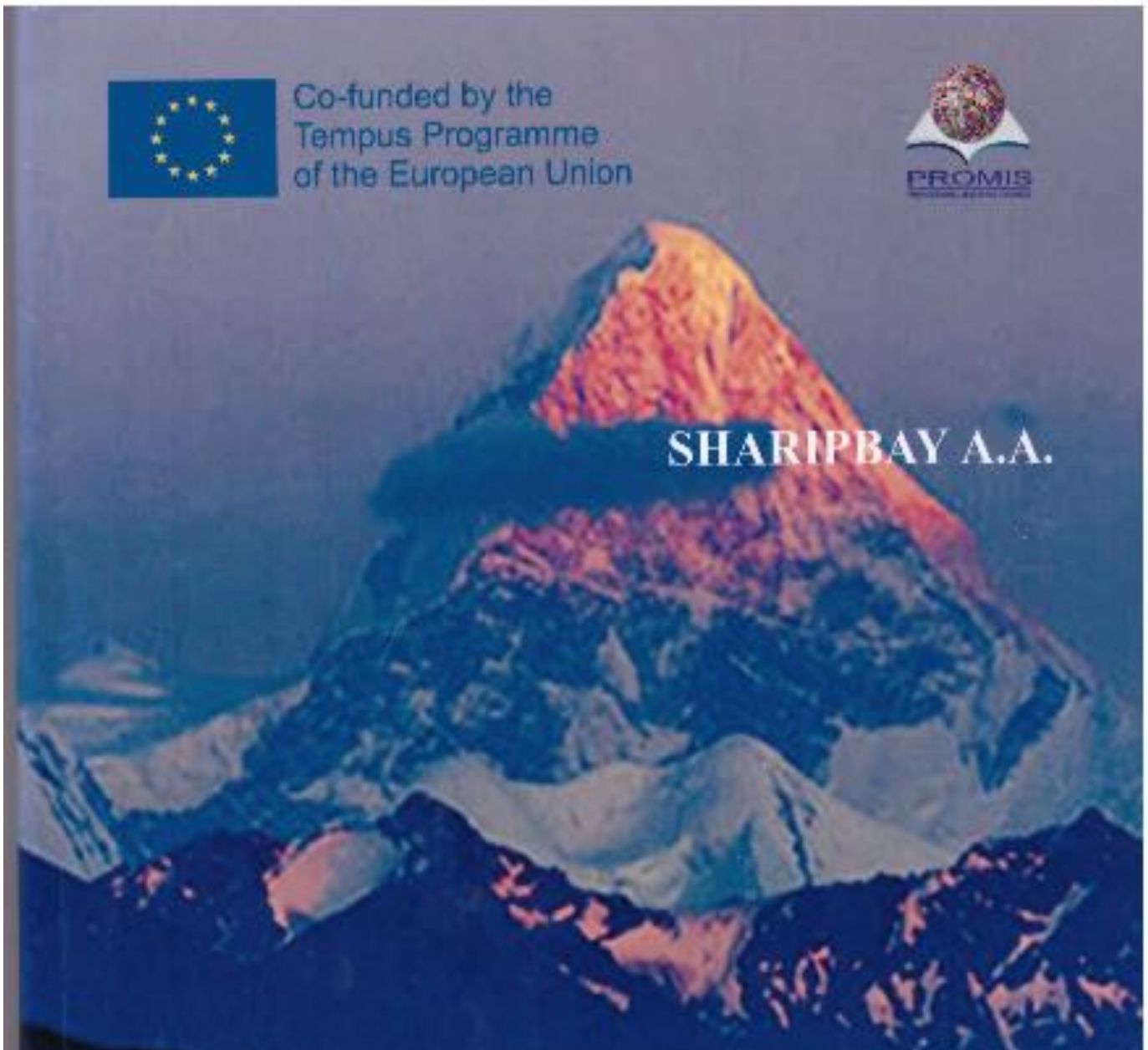
Co-funded by the
Tempus Programme
of the European Union



SHARIPBAY A.A.

MATHEMATICS FOR COMPUTER SCIENCE

(Training manual)

Co-funded by the
Tempus Programme
of the European Union

SHARIPBAY A.A.

MATHEMATICS FOR COMPUTER SCIENCE

(Training manual)

Astana 2017

UDC 004.4(075.8)
LBC: 32.973.26-018.1я73
SH-25

Reviewers:

Kaziev Galym Zulkhurnaevich – D.T.S., Professor, Director of the National Informations Technologies JSC.

Atanov Sabyrzhan Kubeisinovich – D.T.S., Professor of the Chair of Computer Science and Software of the Lev Gumilyov Eurasian National University.

Uskenbayeva Raisa Kabievna – D.T.S., Professor, Prorector of the International Informations Technologies Unversity.

Autors:

SH-25 Sharipbay A.A. – D.T.S., Professor, Laureate of State prize of the Reppublic of Kazakhstan, Director of the SRI Artificial Intelligence, Professor of the Chair of Computer Sciense and Information Security of the Lev Gumilyov Eurasian National University.

SH-25 Sharipbay A.A. Mathemetics for Computer Sciense – Training manual, Astana, 2017, -126 pages.

ISBN 978-601-326-012-9

The content of the training manual "Mathematics for Computer Science" is compiled in accordance with the Syllabus of the compulsory discipline of the same name for the professional master's educational program "Computer Science as a Second Competence" in the specialty 6M060200-Informatics.

The manual is intended for studying the mathematical foundations of the science of computer science by undergraduates of the oriented educational program.

The manual can also be used by students, undergraduates and doctoral students of other specialties, as well as all those who independently wish to study the mathematical basis of computer science.

The training manual was published with funds from the International Project TEMPUS-544319-TEMPUS-1-2013-1-FR-TEMPUS-JPCR-(2013-4530001/00) “Professional Master’s Degree in Computer Science as a second Competence in Central Asia”.

UDC 004.4(075.8)
LBC: 32.973.26-018.1я73
ISBN 978-601-326-012-9

Content

1. SETS

The purpose of the lecture is to give the concept of set and their properties, subset. Finite and infinite sets. Euler-Wenn diagram. Set-theoretical operations. Union. Intersection. Subtraction. Equivalence. Number sets. Sets of natural, integer, real numbers. Set of interval numbers

Sets are one of the most fundamental concepts in mathematics. Developed at the end of the 19th century. The set theory is now a ubiquitous part of mathematics, and can be used as a foundation from which nearly all of mathematics can be derived.

Definition 1.1. *A* set is a collection of distinct objects, considered as an object in its own right.

The objects can be real, physical things, or abstract, mathematical things, and are called elements of the set.

The elements of the set will be shown in curly brackets {and} and do not repeat.

For example: the numbers 1, 2, and 3 are distinct objects when considered separately, but when they are considered collectively they form a single set of size three, written {1,2,3}.

The names of the sets are denoted by capital Latin letters and their elements - by small Latin letters or *A*rabic numerals. In both cases it is possible to use indexes.

Record $a \in A$ ($a \notin A$) means that the element belongs to a (not belongs) set *A*.

A set is defined in two ways: by listing all the elements or describing of the properties of elements.

Sets may be described in many ways: by roster, by set-builder notation, by interval notation, by graphing on a number line, and/or by Wenn diagrams

For example:

1) the set of small Latin letters, which denote the vowel sounds English is $V_s = \{a, e, i, o, u\}$;

2) The set of natural numbers less than the number 100

$N_{100} = \{n | \ n \in N \ \text{и} \ n \leq 100\}$.

Definition 1.2. The number of elements of *A* is denoted by $|A|$ is called the *cardinality* of the set *A*.

*A*mong all sets there are two special sets:

1. Ø - the empty set that does not contain a single element.

2. U - universal set (universe), containing all the elements of this type.

Regarding the theory of the universe is the set containing all elements as objects considered in this theory.

For example, the universe is:

1) in number theory - the set of all integers;

2) in the theory of language - the set of all words in a given alphabet;

3) in geometry - the set of all points of *n*-dimensional geometric space.

Definition 1.3. If the number of elements is finite (non-negative integer *k* exists, equal to the number of elements of the set), then it is called *a finite set*, otherwise it is called *an infinite set.*

In particular, the empty set is a finite set, the number of items is equal to zero, i.e | Ø | = 0.

In the future, we will consider only finite sets.

If finite sets $X_1, \ldots, X_n$ are disjoint ($Xi \cap Xj = Ø$), then

$|X_1 \cup X_2 \cup \ldots \cup X_n| = |X_1| + |X_2| + \ldots + |X_n|$.

If $X_1, \ldots, X_n$ is the final set, then

$|X_1 \times X_2 \times \ldots \times X_n| = |X_1| \times |X_2| \times \ldots \times |X_n|$.

Definitions 1.4. Let we are given two sets *A* and *B*, then over them can be determined next operations:

(1) *Union* consists of elements *A* or *B*, written as

$A \cup B = \{x: x \in A \vee x \in B\}$.

(2) *Crossing* consists of elements *A* and *B*, written as

$A \cap B = \{x: x \in A \,\&\, x \in B\}$.

(3) *Supplement* consists of the elements of the universe *U*, and does not include the elements of *A*, written as

$$\overline{A} = \{x| \quad x \in U \,\&\, x \notin A\}.$$

(4) *Difference* consists of the elements of *A* and not an element of *B*, written as

$A \setminus B = \{x: x \in A \,\&\, x \notin B\}$.

(5) *Symmetric difference* consists only of elements *A* or only elements of *B*, written as

$A \triangle B = \{x| (x \in A \,\&\, x \notin B) \vee (x \in B \,\&\, x \notin A)\}$

(6) *Direct product* consists of all ordered pairs of elements *A* and *B*, written as

$A \times B = \{(a, b) | \ a \in A \ \& \ b \in B\}$

Operation (1) - (3) may be represented by Euler-Wenn diagram (Fig. I.1), wherein a universe *U* depicted rectangle, and a plurality of *A* and *B*, a circumference. To highlight the result of shading applied.

This shows that the sets *A* and *B* are subsets of *U*, and they are written as $A \subseteq U$ and $B \subseteq U$ (see. I.2.2.).

Operations (1) - (3) can be determined not only on the two sets, but over n sets $A_1, A_2, ..., A_n$, where $n \in N$ & $n>2$.

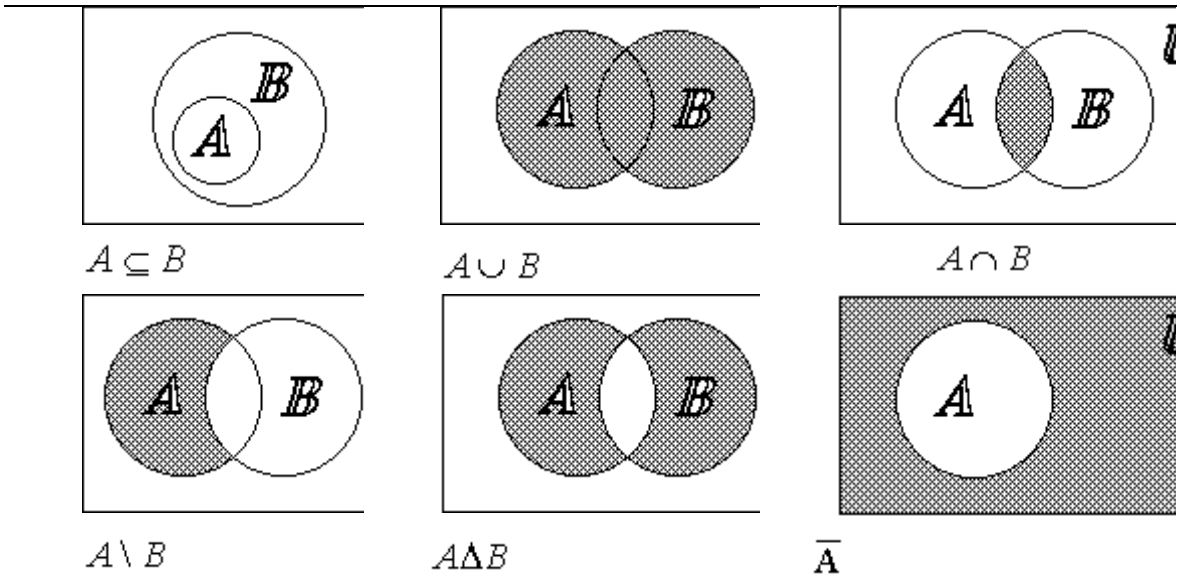


Fig. 1. Euler- Wenn diargam.

Examples 1.1. Let $A = \{a, b, c, d, e, f\}$, $B = \{c, d\}$, then:

1. $A \cup B = \{a, b, c, d, e, f\}$;
2. $A \cap B = \{c, d\}$;
3. $B \times B = \{(c, c), (c, d), (d, c), (d, d)\}$;
4. $A \setminus B = \{a, b, e, f\}$;
5. $A \triangle B = \{a, b, f\}$.
6. $\overline{A}$ depends on what is the universe *U*. For example, if $U = \{a, b, c, d, e, f, h\}$, then $\overline{A} = \{h\}$.

Now you can show the way of the task table sets and operations on them. Suppose that U, $A \subseteq U$ and $x \in U$.

Definition I.5. *Indicator* (*Characteristic function*) for set A is called $I_A(x)$ and defined as:

$$I_A(x)=\begin{cases}1, & if \quad x\in A\\ 0, & if \quad x\notin A\end{cases}$$

Thus: $I_A: U \to \{0,1\}$.

For $A \subseteq U$ and $B \subseteq U$ has the following properties:

$I_A(x)=I_B(x) \Leftrightarrow A=B$;

$I_A(x) \le I_B(x) \Leftrightarrow A \subseteq B$;

$I_{\bar{A}}(x)=1-I_A(x)$;

$I_{A\cup B}(x)=I_A(x)+I_B(x)-I_A(x)\cdot I_B(x)$;

$I_{A\cap B}(x)=I_A(x)\cdot I_B(x)$;

$I_{A\setminus B}(x)=I_A(x)-I_A(x)\cdot I_B(x)$;

$I_{A\setminus B}(x)=I_A(x)-I_A(x)\cdot I_B(x)$;

$I_{A\Delta B}(x)=I_A(x)+I_B(x)-2I_A(x)\cdot I_B(x)$.

Indicators conveniently given by a table 1.1.

Table 1.1. Indicators.

$x\in A$	$x\in B$	$x\in A\cup B$	$x\in A\cap B$	$x\in A\setminus B$	$x\notin A$	$x\in A\Delta B$
0	0	0	0	0	1	0
0	1	1	0	0	1	1
1	0	1	0	1	0	1
1	1	1	1	0	0	0

Set Operations have the following properties:

I. *The union, intersection and difference*:

1) $A \cup \varnothing = A$ - property of zero;

2) $A \cup A = A$ - idempotence;

3) $A \cup B = B$, if all elements of A is contains in B;

4) $A \cup B = B \cup A$ - commutes;

5) $(A \cup B) \cup C = A \cup (B \cup C) = A \cup B \cup C$ - associativity;
6) $A \cap \varnothing = A$ - property of zero;
7) $A \cap A = A$ - idempotence;
8) $A \cap B = A$, if all elements of A is contains in B;
9) $A \cap B = B \cap A$- commutes;
10) $(A \cap B) \cap C = A \cap (B \cap C) = A \cap B \cap C$ - associativity;
11) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ - distributivity;
12) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ - distributivity;
13) $A \cap (B \setminus C) = (A \cap B) \setminus (A \cap C)$ - distributivity;
14) $A \cup \overline{A} = U$ - property of the additions;
15) $A \cap \overline{A} = \varnothing$ - property of the additions;
16) $\overline{A \cup B} = \overline{A} \cap \overline{B}$ - the law of de Morgan;
17) $\overline{A \cap B} = \overline{A} \cup \overline{B}$ - the law of de Morgan;
18) $\overline{\overline{A}} = A$ - involutivity;
19) $A \setminus \varnothing = A$ - property of the difference;
20) $A \setminus A = \varnothing$ - property of the difference;
21) $A \setminus B = A \cap = \varnothing$ - property of the difference;
22) $B \setminus A = B \cap = B \setminus (B \cap A)$ - property of the difference;

II. Symmetric difference and direct product:
1) $A \triangle \varnothing = A$ - property of zero;
2) $A \triangle A = \varnothing$ - idempotence;
3) $A \triangle B = (A \cup B) \setminus (A \cap B)$ - property symmetric difference;
4) $A \triangle B = B \triangle A$ - commutes;
5) $(A \triangle B) \triangle C = A \triangle (B \triangle C) = A \triangle B \triangle C$ - associativity;
6) $(A \cup B) \times C = (A \times C) \cup (B \times C)$ - distributivity;
7) $A \times (B \cup C) = (A \times B) \cup (A \times C)$ - distributivity;
8) $(A \cap B) \times C = (A \times C) \cap (B \times C)$ - distributivity;
9) $A \times (B \cap C) = (A \times B) \cap (A \times C)$ - distributivity;
10) $(A \setminus B) \times C = (A \times C) \setminus (B \times C)$ - distributivity;
11) $A \times (B \setminus C) = (A \times B) \setminus (A \times C)$ - distributivity.

Definitions 1.5. Suppose we have two sets A and B of the same type. Then we can enter the following relationship:

1) $A = B$: A is equal to B, if A and B are composed of the same elements, i.e. A and B are subsets of each other;

2) $A \subseteq B$: A is contained in B, if all elements of A belong to the B or A is equal to B, it means that A is a subset of B;

3) $A \subset B$: A strictly contained in B, if all elements of A belong to B and A is not equal to B, i.e. some elements of B do not belong A, it means that A is a proper subset of B.

Similarly, you can determine the relationship includes $A \supseteq B$ strictly includes $A \supset B$.

It is easy to notice that the above relationships entered =, $\subseteq$ and $\subset$ are subsets of the direct product $A \times B$.

Thus, we can assume that any relationship - is a subset of the direct product generated by the law.

Note 1.1. The empty set Ø is a proper subset of any finite set.

Examples 1.2.

1) If $A = \{a, b, c\}$, $B = \{b, a, c\}$, then $A = B$;

2) if $A = \{1,2,3,4\}$, $B = \{3,1,4,2\}$, then $A \subseteq B$;

3) if $A = \{1,2,3\}$, $B = \{3,1,4,2\}$, then $A \subset B$

For the convenience of working with numbers they are logically divided into sets and identified:

1. *Real numbers.* The set of real numbers is represented by the letter R. Every number (except complex numbers) is contained in the set of real numbers. When the general term "number" is used, it refers to a real number. All of the following types or numbers, may also be considered as real numbers.

2. *Integer numbers.*The set of integers is represented by the letter Z. An integer is any number in the infinite set,

$Z = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}$.

Integers are sometimes split into 3 subsets, Z^+, Z^- and 0. Z^+ is the set of all positive integers {1, 2, 3, ...}, while Z- is the set of all negative integers {..., -3, -2, -1}. Zero is not included in either of these sets. Z^{nonneg} is the set of all positive integers including 0, while Z^{nonpos} is the set of all negative integers including 0.

3. *Natural numbers*. The set of natural numbers is represented by the letter *N*. This set is equivalent to the previously defined set, Z^+. So a natural number is a positive integer.

$N = \{ 1, 2, 3, 4, ... \}$

4. *Whole Numbers.* The set of whole numbers is represented by the letter *W*. This set is equvalent to the previously defined set, Z^{nonneg}. So a whole number is a member of the set of positive integers (or natural numbers) or zero.

$W = \{ 0, 1, 2, 3, 4, ... \}$.

5. *Prime numbers.* The set of prime numbers is represented by the letter *P*. A prime number is an integer that is divisible only by itself and one. Examples of prime numbers are 3, 5, 7, 11, 13, 17, 19.

6. *Rational Numbers.*The set of rational numbers is represented by the letter *Q*. A rational number is any number that can be written as a ratio of two integers. The set of rational numbers contains the set of integers since any integer can be written as a fraction with a denominator of 1. A rational number can have several different fractional representations. For example, 1/2 is equivalent to 2/4 or 132/264. In decimal representation, rational numbers take the form of finite or infinite periodic fractions. Some examples of rational numbers are:

$$\frac{4}{7} \qquad 30 = \frac{30}{1} \qquad \frac{1}{4} = 0.25\overline{0} \qquad \frac{42}{99} = 0.424242 = 0.\overline{42}$$

Irrational Numbers. The set of irrational numbers is represented by the letter I. Any real number that is not rational is irrational. These are numbers that can be written as decimals, but not as fractions. They are infinite non-periodic decimal fractions. Some examples of irrational numbers are:

$$\pi = 3.1415926535... \quad \sqrt{2} = 1.4142135623... \quad \sqrt{7} = 2.6457513110...$$

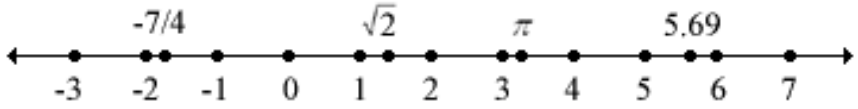
Note. Any root that is not a perfect root is an irrational number. So any roots such as the following examples, are irrational.

$$\sqrt{2},\ \sqrt{3},\ \sqrt{5},\ \sqrt{6},\ \sqrt{7},\ \sqrt{8},\ \sqrt{10},\ ...\ \sqrt[3]{2},\ \sqrt[3]{13},\ \sqrt[3]{20},\ \sqrt[8]{10002229}$$

Note 1.1. Between the sets of numbers have the following relationship: $P \subseteq N \subseteq W \subseteq Z \subseteq Q \subseteq R$.

The Real Number Line

Every real number can be associated with a single point on the *real number line*

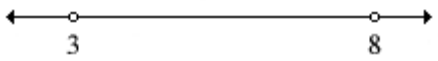


Intervals.

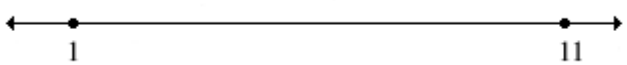
An interval is a set that consists of all real numbers between a given pair of numbers. It can also be thought of as a segment of the real number line. An endpoint of an interval is either of the two points that mark the end of the line segment. An interval can include either endpoint, both endpoints or neither endpoint. To distinguish between these different intervals, we use interval notation.

An open interval does not include endpoints. The exclusion of the endpoints is indicated by round brackets () in interval notation. When the interval is represented by a segment of the real number line, the exclusion of an endpoint is illustrated by an open dot. For example, the interval of numbers between the integers 3 and 8, excluding 3 and 8, is written as $(3, 8) = \{x: 3 < x < 8 \}$

in interval notation. As a segment of the real number line, it would be represented by the line below.

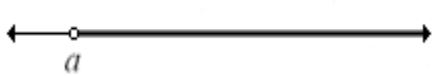
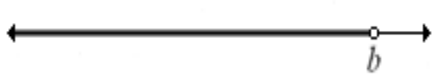


A closed interval includes the endpoints. The inclusion of the endpoints is indicated by square brackets [ ] in interval notation. When the interval is represented by a segment of the real number line, the inclusion of an endpoint is illustrated by a closed dot. For example, the interval of numbers between the integers 1 and 11, including both 1 and 11, is written as $[1, 11] = \{x: 1 \leq x \leq 11\}$ in interval notation. As a segment of the real number line, it would be represented by the line below.



One endpoint of an interval can be included, while the other is excluded. The interval $[a, b)$ represents all numbers between a and b, including a but not b. Similarly, the interval $(a, b]$ would represent all of the numbers between a and b, including b but not a. These intervals are shown in more detail in the table below.

Infinite intervals are those that do not have an endpoint in either the positive or negative direction, or both. The interval extends forever in that direction. Infinite intervals are summarized in the table below.

Interval Notation	Number Line Sketch	Set-builder Notation
(a, b)		$\{x \mid a < x < b\}$
$(a, b]$		$\{x \mid a < x \leq b\}$
$[a, b)$		$\{x \mid a \leq x < b\}$
$[a, b]$		$\{x \mid a \leq x \leq b\}$
(a, ∞)		$\{x \mid x > a\}$
$(-\infty, b)$		$\{x \mid x < b\}$
$[a, \infty)$		$\{x \mid x \geq a\}$
$(-\infty, b]$		$\{x \mid x \leq b\}$
$(-\infty, \infty)$		R

In mathematics, an (real) interval is a set of real numbers with the property that any number that lies between two numbers in the set is also included in the set. For example, the set of all numbers x satisfying $0 \leq x \leq 1$ is an interval which contains 0 and 1, as well as all numbers between them. Other examples of intervals are the set of all real numbers R, the set of all negative real numbers, and the empty set.

Notation for intervals

The interval of numbers between a and b, including a and b, is often denoted $[a, b]$.

To indicate that one of the endpoints is to be excluded from the set, the corresponding square bracket can be either replaced with a parenthesis, or reversed:

$$(a,b) =]a,b[= \{x \in R: \quad a < x < b\},$$

$$[a,b) = [a,b[= \{x \in R: \quad a \leq x < b\},$$

$(a,b] =]a,b] = \{x \in R: \quad a < x \leq b\}$, $[a,b] = [a,b] = \{x \in R: \quad a \leq x \leq b\}$.

Note that (a, a), $[a, a)$ and $(a, a]$ each represents the empty set, whereas $[a, a]$ denotes the set $\{a\}$. When $a > b$, all four notations are usually taken to represent the empty set.

Infinite endpoints.

In both styles of notation, one may use an infinite endpoint to indicate that there is no bound in that direction. Specifically, one may use $a = -\infty$ or $b = +\infty$ (or both). For example, $(0, +\infty)$ is the set of all positive real numbers, and $(-\infty, +\infty)$ is the set of real numbers.

The extended real number line includes $-\infty$ and $+\infty$ as elements. The notations $[-\infty, b]$, $[-\infty, b)$, $[a, +\infty]$, and $(a, +\infty]$ may be used in this context. For example $(-\infty, +\infty]$ means the extended real numbers excluding only $-\infty$.

Integer intervals.

The notation $[a .. b]$ when a and b are integers, or $\{a .. b\}$, or just $a .. b$ is sometimes used to indicate the interval of all integers between a and b, including both. This notation is used in some programming languages.

An integer interval that has a finite lower or upper endpoint always includes that endpoint. Therefore, the exclusion of endpoints can be explicitly denoted by writing $a .. b - 1$, $a + 1 \quad .. b$, or $a + 1 \quad .. b - 1$. Alternate-bracket notations like $[a .. b)$ or $[a .. b[$ are rarely used for integer intervals.

An open interval does not include its endpoints, and is indicated with parentheses. For example (0,1) means greater than 0 and less than 1. A closed interval includes its endpoints, and is denoted with square brackets. For example [0,1] means greater than or equal to 0 and less than or equal to 1.

Classification of intervals

The intervals of real numbers can be classified into eleven different types, listed below; where a and b are real numbers, $a < b$:

empty: $[b, a] = (a, a) = [a, a) = (a, a] = \{\} = \emptyset$,

degenerate: $[a, a] = \{a\}$,

proper and bounded:

open: $(a, b) = \{x \mid a < x < b\}$,

closed: $[a, b] = \{x \mid a \leq x \leq b\}$,

left-closed, right-open: $[a, b) = \{x \mid a \leq x < b\}$,
left-open, right-closed: $(a, b] = \{x \mid a < x \leq b\}$,
left-bounded and right-unbounded:
left-open: $(a, \infty) = \{x \mid x > a\}$,
left-closed: $[a, \infty) = \{x \mid x \geq a\}$,
left-unbounded and right-bounded:
right-open: $(-\infty, b) = \{x \mid x < b\}$,
right-closed: $(-\infty, b] = \{x \mid x \leq b\}$,
unbounded at both ends: $(-\infty,+\infty) = R\}$.

Intervals of the extended real line.

In some contexts, an interval may be defined as a subset of the extended real numbers, the set of all real numbers augmented with −∞ and +∞.

In this interpretation, the notations $[-\infty, b]$, $[-\infty, b)$, $[a, +\infty]$, and $(a, +\infty]$ are all meaningful and distinct. In particular, $(-\infty, +\infty)$ denotes the set of all ordinary real numbers, while $[-\infty, +\infty]$ denotes the extended reals.

This choice affects some of the above definitions and terminology. For instance, the interval $(-\infty, +\infty) = R$ is closed in the realm of ordinary reals, but not in the realm of the extended reals.

Properties of intervals

The intersection of any collection of intervals is always an interval. The union of two intervals is an interval if and only if they have a non-empty intersection or an open end-point of one interval is a closed end-point of the other $((a,b) \cup [b,c] = (a,c])$.

Examples 1.1.

1. Let $A = \{1, 2\}$, $B = \{a, b\}$, $C = \{+, -\}$. Then distributivity $(A \cup B) \cup C = A \cup (B \cup C) = A \cup B \cup C$ defined as $(\{1,2\} \cup \{a, b\}) \cup \{+, -\} = \{1,2\} \cup (\{a, b\}) \cup \{+, -\}) = \{1,2\} \cup \{a, b\} \cup \{+, -\}$.

2. *By roster*: A roster is a list of the elements in a set, separated by commas and surrounded by curly braces:

1) {2,3,4,5,6} is a roster for the set of integers from 2 to 6, inclusive; 2) {1,2,3,4,…} is a roster for the set of positive integers. The three dots indicate that the numbers continue in the same pattern indefinitely.

By set-builder notation: Set-builder notation is a mathematical shorthand for precisely stating all numbers of a specific set that possess a specific property. $\{x \in Z: 2 \le x \le 6\}$ is set-builder notation for the set of integers from 2 to 6, inclusive, where $\in$ = "is an element of" Z -the set of integers numbers. The statement is read, "all *x* that are elements of the set of integers, such that, *x* is between 2 and 6 inclusive."

$\{x \in Z | \ x > 0\}$. The statement is read, "all x that are elements of the set of integers, such that, the *x* values are greater than 0, positive." (The positive integers can also be indicated as the set Z+). It is also possible to use a colon (:), instead of the | , to represent the words "such that". $\{x \in Z | \ 2 \le x \le 6\}$ is the same as $\{x \in Z | \ x > 0\}$

By interval notation: An interval is a connected subset of numbers. *Interval notation* is an alternative to expressing your answer as an inequality. Unless specified otherwise, we will be working with real numbers.

Exercises 1.1. Let $A = \{1, 2, 4\}$, $B = \{3, 4, 5, 6\}$. Run this:

1) $A \cup \varnothing$;
2) $A \cup A$;
3) $A \cap B$;
4) $A \times B$;
5) $A \setminus B$;
6) $A \cup$;
7) $A \triangle B$;
8) $(A \cup B) \times C$;
9) $(A \setminus B) \times C$.

Questions 1.1:

1. What is a set?
2. How is sets determined?
3. What is the universal set?
4. How is the subset determined?
5. What is the Euler-Wenn diagram?
6. What is the Euler-Wenn diagram for the association?
7. How is the direct product of the sets determined?

8. How is the difference of the sets?

9. How is the symmetric difference of the sets?

10. How is the indicator function of the sets?

11. What is the law of de Morgan?

12. If A = {1,2,3}, B = {3,4}, then, what means {1,2,3,4}?

13. If A = {1,2,4}, B = {4,3,2}, then, what means {2,4}?

14. If A {1,2,3,4}, B={3,4,5,6}, then, what means {2,4}?

15. What is the distributivity?

16. What is the associativity?

17. What is commutativity?

Tests I.1:

1. Determine | L |, if L is consist up of the Latin lowercase letters: L=*{a, b, c, d, e, f, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z},*

A) 1

B) 26

C) 28

D) 0

E) 30

2. *If D = {d | d* - an integer and *0* is performed $\leq d \leq 9\}$, i.e., *D = {0,1,2,3,4,5,6,7,8,9}* then what is the | *L |?*

A) d

B) 9

C) 10

D) 0

E) 1

3. What is the ratio of accessories?

A) ∩

B)

C)

D) Ø

E) &

I.3.3 tests.

4. What result is obtained after performing $A \cap V$ for given sets *A = {0, 2, 4, 6}* and *B = {- 2, -1, 0, 1, 2}*

A) {0, 2}

B) {0, 2, 4, 6}

C) {-2, -1, 0, 1, 2}

D) {0, 2, 4, 6, -2, -1, 0, 1, 2}

E) Ø

5. What happens after the result of the operation $A \cup V$ for given sets *A = {1, 3, 5}* and *B = {2, 4, 6, 8}*

A) {1, 2, 3, 4, 5, 6, 8}

B) {1, 3, 5}

C) {2, 4, 6, 8}

D) Ø

E) {1, 3, 8}

6. What will the result after performing $A \setminus B$ for given *sets A = {a, b, c, d, e, f, g}* and *B = {d, e, f}*

A) {a, b, c, d}

B) {a, b, c, d, e, f}

C) {d, e, f}

D) {d, e}

E) Ø

2. NOTATIONS AND CODING OF INFORMATION

The purpose of the lecture is to give the rules of notation (designation) and coding of information for computer storage and processing.

Outline of the lecture is to explore methods and reporting systems of variables using numbers, letters and other symbols.

It is known that information can be considered a reflection of the properties and relations of tangible and intangible objects and subjects of the world. Information can be designated and understood, i.e every piece of information must have its form and its content.

Typically, for designation (notation) of data certain characters and their sequences are used. Herewith, a man distinguishes characters by their mark, and the computer - by their codes, consisting of a sequence of 0 and 1 as the physical storage devices in computer (memory cells and registers) can be only in two states, which correspond to 0 or 1. Using a number of similar physical devices, you can store in memory of computer any information using binary code as a sequence of 0 and 1. Therefore, any notation (numeric, text, graphics, sound, etc.) of information, which operates modern computers, is encoded (converted) in binary code and is decoded (converted back) in notation for ease of individual's perception. Binary code of information is stored in RAM and occupies one byte, when a character outputs to a printer or to computer screen decoding occurs, i.e. converting the character code in its image.

Traditionally for one character encoding the amount of information equal to 1 byte, i.e. I = 1 byte = 8 bits is used. If we consider the characters as possible events, we can calculate how many different characters can be encoded in a single byte:

$N = 2^I = 2^8 = 256$ bits.

Thus, each character is assigned to a unique decimal code between 0 and 255, or the corresponding binary code from 00000000 to 11111111.

While entering symbolic information to the computer, its binary encoding happens, code of the character is stored in RAM and occupies one byte, when a character outputs to a printer or to computer screen decoding occurs, i.e. converting the character code in its image. In general terms, encoding of information can be defined as transfer of information provided by message in primary alphabet to a sequence of codes. It should be understood that any data -

it is somehow encoded information. Information may be presented in different forms: in the form of numbers, text, graphics, sound and etc. Conversion from one form into another is coding. It should be understood that any data it is somehow encoded information. Information can be presented in different forms: as numbers, text, drawing, etc. Transfer from one form to another is encoding.

If the value is determined at the time of construction of the general rules of interpretation of the language of communication, this value will be constant, or alternatively – variable.

Definition 2.1. Notation is:

1) a system of figures or symbols used in a specialized field to represent numbers, quantities, tones or values;

2) the act or process of using such a system.

In computer science the treatment of numerical values may require a different systems of numbers calculation. The bases of these systems can be 2,3,4, ...

Definition 2.2. *Number system* is a way of writing numbers with numbers and sets of rules. There are several ways of recording numbers using digits.

Any number system satisfies the following rules:

– the possibility of recording the numbers in a given range;

– each sequence of digits defines only a single numeric value;

– perform simple operations.

All number system are divided into: positional number systems and nonpositional number systems.

1. In positional number system the values of digits depend on their position in the record number. If same digit in the record of number occurs more than once, then it determines a different value. For example, in the three-digit number 333 a left-most digit 3 identifies three hundreds, the average digit 3 - three tens, and the rightmost digit 3 - three units.

2. In nonpositional number system the values of digits do not depend on their place in the record of numbers. For example, the record of number by Roman digits: in record of number LXXXVIII - eighty-eight the digit L is fifty, X - ten, V - five, I - one.

The positional number system is characterized by its base. The base defines the number of digits used in the system. For example, the number of digits in decimal system is equal to ten, in octal system – eight, in binary system - two etc.

At any positional number system with the base q the predetermined number A can be represented as follows:

$$A_{(q)} = a_{n-1}q^{n-1} + \cdots + a_1 q^1 + a_0 q^0 + a_{-1}q^{-1} + \cdots + a_{-m}q^{-m} \qquad (\boldsymbol{1})$$

where a_i - number of digits used in the number system, n - number of places in the integer part, m - number of places in the fractional part (i=n–1,..., 1, 0, –1,....,–m).

Among them, we are interested in the decimal number system and binary number system.

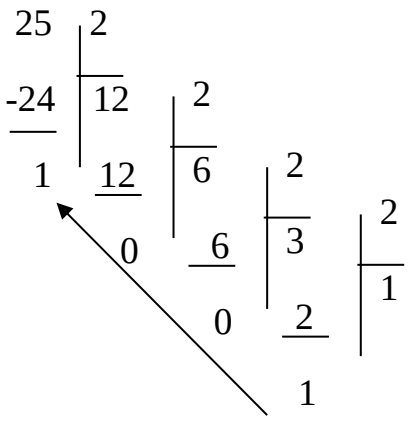
Table 2.1 equivalent decimal and binary digits are given.

Table 2.1. Equivalent decimal and binary digits

Decimal digit	Binary digit
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

From this table you can see that the recording of the same numerical value in different number systems require a different number of digits. For example, the decimal number16 in binary notation will be 10000. To transfer a given integer number from number system with the base p to the number system with the base q this number should be repeatedly divided to q, while the remainder will not be less than q.

To take the resulting quotient as the most significant place of the number with a base of q, and to take residues as values of remaining places in a direction starting from the last residue to the first residue and form a chain from left to the right. For example, transferring the number 25 in the decimal system to the binary system will be as follows:



Namely, $25_{(10)} = 11001_{(2)}$. To verify the number 11001 we decompose according to the equation (1) as follows:

$$11001_{(2)} = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 16 + 8 + 0 + 0 + 1 = 25_{(10)}.$$

In order to transfer of stated right fractional number with the base *p* to the base *q* on the basis of we need to multiply this number by q several times, while the value of the digit of the fractional part is not equal to zero, or until the specified accuracy. As the value of place of a right fraction with a new base q we need to form a chain from left to right in the direction from the first appeared integer part until last appeared integer part.

Performing operations and their properties in the decimal system and the binary system are similar. The properties of these operations are identical to the properties of operations on decimal numbers.

Example 2.1:

1. A four-digit number 1952 of the decimal system is expressed as follows:

$$1952_{(10)} = 1*10^3 + 9*10^2 + 5*10^1 + 2*10^0.$$

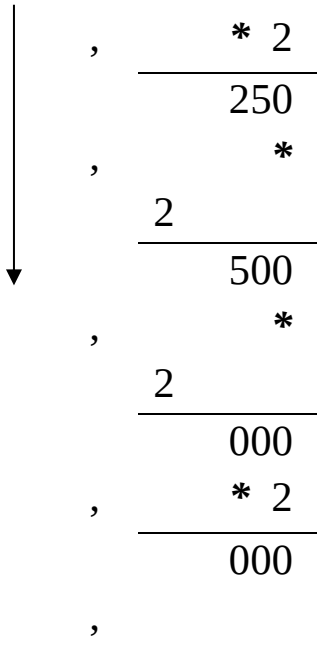
2.The number of the decimal system with three-digit integer part and three-digit fractional part 596.174(10) is expressed as follows:

$$596.174_{(10)} = 5*10^2 + 9*10^1 + 6*10^0 + 1*10^{-1} + 7*10^{-2} + 4*10^{-3}.$$

3.The number of binary system with four-digit integer part and a three-digit fractional part 1010.101(2) is expressed as follows:

$$1010.101_{(2)} = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}.$$

1. The transfer of a fractional number 0.625 in decimal number system to binary system will look like this:



So the result is: $0.625_{(10)} = 0.1010_{(2)}$.

Opportunities of all positional number systems are the same. The difference between them is only in the methods of designation of number values, but types of operations on numbers and their properties are the same.

However, among them, the decimal number system is the most common. Therefore, we first look familiar to us decimal system of numbers, operations on them, and the properties of these operations, because they are suitable for other systems of notation of numbers.

Integer numbers will be presented by Arabic numerals, in front of their negative values "-" sign will be written, and at front of their positive values "+" sign can be written.

Real numbers depending on their the method of representation are divided into two groups: real numbers with fixed-point and floating-point. Representation of real numbers with the fixed point consists of integer and

fractional parts. The integer part is placed on the left of the fractional part, and they are separated by point ".".

To indicate positive or negative values "+" or "-" in front of them recorded. Both parts are represented by Arabic numerals.

Presentation of real numbers with floating-point consists of parts called the mantissa, the basis of the number system and order.

If we denote the mantissa by M, the order by p, the basis of the number system by q, the real numbers are as follows: $M * q^p$.

To understand, examples of real numbers with floating point in the table 2 are reviewed.

Table 2. Examples of real numbers with floating point.

№	Example	Mantissa	Order	Value
1.	$-12.*10^3$	–12	3	–12000
2.	$0.3*10^{+2}$	0.3	2	30
3.	$254*10^{-2}$	254	–2	2.54
4.	$1.5*10^1$	1.5	1	15
5.	$+2.17*10^2$	2.17	+2	217

One and the same real number with floating point can be represented in different ways. For example, the same number of 3.14 may be recorded:

$$314.*10^{-2} = 31.4*10^{-1} = 3.14*10^0 = 0.314*10^1 = 0.0314.*10^2 = \ldots$$

To have a single entry for the submission of real number with floating-point we need to normalize it to the following condition:

$$q^{-1} \le |M| < 1,$$

where $|M|$ - the absolute value.

For example, real numberw with floating point in a normalized form are as follows:

$0.1364*10^4$ and $0.617*10^{-7}$.

In order to simplify the arithmetic operations in the computer special codes to represent numbers are used. We consider direct code, inverse code and additional code of numbers.

Direct code of binary number is itself a binary number, and a sign of the binary number is written by dinary digit: "-" sign - the number 1, "+" sign - digit 0. For example, a negative binary number 1011_2 in direct code is written as 1.1011.

Representation of numbers in a computer, compared with forms well known since high school, has two important differences:
- numbers are recorded in the binary number system;
- for recording and processing of numbers a finite number of places are assigned (in the ordinary - non-computer arithmetic has no limit).

Addition and multiplication of binary numbers is done according to the table of addition and multiplication:

Addition of binary numbers	**Multification of binary numbers**
0 + 0= 0	0 · 0 = 0
0 +1 = 1	0 · 1 = 0
1 + 0 = 1	1 · 0 = 0
1 + 1 = 10	1 · 1 = 1

Arithmetic device in computer performs an action not with the binary numbers according to the rules of binary arithmetic, but with their binary codes according to the rules of arithmetic binary codes.

Differences between the rules of arithmetic of binary codes from ordinary arithmetic is in limit of discharge grid. In other words, for the record of number in the computer memory a fixed number of places is allocated. Computer memory has byte structure, however, the size of one addressed cell is typically several bytes: 2, 4, 8 bytes.

All the information on the computer is represented in binary code. From the whole set of codes, we consider the direct, inverse and additional codes.

To record integer binary number in the direct code binary numbers are complemented by sign pool, which is assumed to be equal to "0" for positive numbers and "1" - for negative. In manual recording of numbers with sign, the sign pool, for convenience, is separated from significant pools by point.

For example, the decimal number (+12) in direct binary code is written as (0.1100), and a decimal number - so (-12) - (1.1100).

Direct code is used for storage of numbers in the computer memory, as well as for operations of multiplication and division.

Other forms of presenting numbers with sign are the inverse and additional codes. These codes allow you to replace the subtraction of integers with their addition, based on the principle: a - b = a + (-b).

Positive numbers recorded in direct, reverse and additional codes are the same.

Thus, positive decimal number 12 in direct, inverse and additional binary codes can be written as follows: (0.1100).

To convert a negative number from direct code into reverse, one shoud be saved in sign pool and numbers of significant pools should be reversed, i. e. "1" is replaced by "0" and "0" to "1".

Additional code of negative number is obtained from the inverse code of number by adding "1" to the least significant digit of this number.

Rules of adding in additional code:

1. Addition is made accroding to the rules of addition of binary numbers, including the sign pool.
2. If as a result of adding the transfer occurs (overflow) from sign pool, the transfer is ignored (discarded).
3. If the sign of addition does not coincide with the signs of additives (this situation can arise only when the signs are the same), there is an overflow of digit grid of computer and the result should be declared invalid.

Addition in reverse binary code differs from adding in additional code on only one rule: if as the result of the addition there was the transfer from sign pool, i.e., overflow has occurred, it is necessary to add "1" to the least significant digit.

Example 2.1.

1. +5 - positive integer 5
2. 3.14 - positive real number with fixed-point, the integer part 3, and the fractional part 14.
3. 0.2 - positive real number with fixed-point, the integer part 0 and fractional part 2.
4. -1.001 - negative real number with fixed-point, the integer part 1 and the fractional part of 001.

5. 0.0 - positive real number, the integer part 0 and the fractional part 0.

Example 2.2. Write a decimal number (-12) in direct, inverse, and the additional binary codes in six-digit cell:

1.01100 - direct code;
1.10011 - reverse code;
1.10100 - additional code.

In this example, one place is assigned to the sign of number, five places to the number itself, to the point in the discharge grid no place stands out. The number itself is shifted to the right edge, and the excess discharge (in direct code) recorded as "0". Then direct code is inverted to transfer to reverse.

Transfer of numbers from reverse (additional) code into direct code performed on the same rules as to reverse (additional) code from direct.

Example 2.3. To perform this operation: 15 - 7 in direct, reverse, and additional code:

	Decimal number	**Direct code**	**Reverse code**	**Additional code**
Data	15 – 7	0.1111 – 1.0111	0.1111 + 1.1000	0.1111 + 1.1001
Intermediate result	8		10.0111 + 1	1 0.1000
Final result	8		0.1000	0.1000

Example 2.4. To perform this operation: 7 – 15 in direct, reverse, and additional code:

	Decimal number	**Direct code**	**Reverse code**	**Additional code**

Data	–15 +7	0.1111 1.0111	1.0000 + 0.0111	1.0001 + 0.0111
Intermediate result	–8		1.0111	1.1000
Final result	–8		1.1000	1.0111 + 1 1.1000

Exercise 2.

1. Determine the real numbers with floating point:

1)40,23;

2)–5;

3)$3.3*10^{-2}$;

4)5.1+6i;

5)0.14+7i.

2**.** Move a specified number from one number system to another:

1) 10000001 from binary to decimal system.

2) 129 from decimal to octal system.

3) 1952 from decimal to hexadecimal system.

3. Arrange the arithmetic operations so that it is true the following equation in the binary system: 1100 ? 11 ? 100=100000.

Questions 2.

1. What is a number system?

2. For what groups real numbers are divided?

3. Can the same numeric value be represented in the different number systems?

4. What are the types of numeric values?

5. For what groups real numbers are divided?

Test 2.

1. In what system data is coded in ANSI?

A) in binary system

B) in ternary system

C) in octal system
D) in decimal system
E) in hexadecimal system

2. In what system data is coded in in Unicode?
A) in hexadecimal system
B) in ternary system
C) in octal system
D) in decimal system
E) in binary system

3. How many bytes are used for encoding in Unicode?
A) 2
B) 1
C) 3
D) 5
E) 4

4. What is the number system?
A) A recording method using the numbers and a set of rules.
B) Possibility to record values of the numbers in a given range.
C) Each sequence of numbers identifies only one numerical value.
D) Easiness of performing of operations.
E) Values of numbers do not depend on their position in the record of number.

5. Which number system is the smallest?
A) binary.
B) octal.
C) hexadecimal.
D) Ternary.
E) Decimal.

3. BASES OF MATHEMATICAL LOGIC

Statements and logic connectives. The logic form of the statement: the subject, a predicate, connectives, premises. Conclusions: deductive, inductive. Concepts of the proof. Logic connectives: disjunction, conjunction, negation,

implication, equivalence. Truth tables. Logic functions. Concepts of a tautology and the

Statements

The content of any science make statements (propositions) about the objects of her subject domain. Propositional logic is abstracted from the specific content of the statements and studies the structure of complex sentences and their logical connections.

Statement is the declarative proposition, wich can be true or false. Examples of statements: "Snow is white", "2> 3", "If there is rain, then I take an umbrella", etc.

Statements can be linked to each other by means of logical connections, "not", "and", "or", "implication", "equivalent."

Mathematical logic, we will study with the help of mathematical methods in a some meta-language, which is different from the subject language of the studied logic. Subject language of propositional logic consists of the alphabet and formulas:

Alphabet:

(1) P, Q, R, ... - variables for simple statements (propositional letters);

(2) $\neg$, &, $\vee$, $\rightarrow$, $\leftrightarrow$ - symbols on the statements of operations (logical ligament);

(3) (,) - auxiliary characters (braces).

The formulas or complex statements:

(1) P, Q, R, ... - propositional letters - elementary formula (atoms);

(2) if A, B - formula, $\neg$A, A&B, A$\vee$B, A$\rightarrow$B, A $\leftrightarrow$B - formula.

In the definition of the formulas used metaletters A, ie characters that do not belong subject language.

Examples of formulas: $\neg$P, (P&Q), (R$\rightarrow$ (P $\vee$ R)).

Subformulas - is part of the formula, is the formula itself.

Set Language, we have built a formal system. Now imagine it as meaningful propositional algebra, for this we give the meaning symbols of alphabet and

formulas. Propositional letters, and logical operations are defined in the field of two elements {T, F}, T - True, F - False:

		P&Q	P∨Q	¬P	P→Q	P↔Q
		T	T	F	T	T
		F	T	F	F	F
		F	T	T	T	F
		F	F	T	T	T

The value of the formula E [P_1, ... , P_n] at this interpretation of its constituent propositional letters

γ : {P_1, ... , P_n} $\Rightarrow${T,F} we define by induction on the structure of the formula:

E = P : E[γ] = γ (P);

E = ¬A : E[γ] = ¬A[γ];

E = A&B : E[γ] = (A&B)[γ] = A[γ] & B[γ];

E = A∨B : E[γ] = (A∨B)[γ] = A[γ] ∨ B[γ];

If in the formula the operation ¬ is used only one, the formula is called the *formula with negation.*

Tautology (universally valid formula, logical law) - a formula, true for all interpretations of its constituent propositional letters, in other words, - the column of values, which contains only true values (denoted by the symbol ⊨)

Basic tautology.

1a. ⊨A→(B→A)

1б. ⊨(A→B)→((A→(B→C))→(A→C))

2. ⊨A→(B→A&B)

3a. ⊨A&B→A

3б. ⊨A&B→B

4a. ⊨A→A∨B

4б. ⊨B→A∨B

5. ⊨(A→C)→((B→C)→(A∨B→C))

6. ⊨(A→C)→((A→¬C)→ ¬A)

7. ⊨¬¬A→A

8. $\models(A \rightarrow B) \rightarrow ((B \rightarrow A) \rightarrow (A \leftrightarrow B))$

9a. $\models(A \leftrightarrow B) \rightarrow (A \rightarrow B)$

9б. $\models(A \leftrightarrow B) \rightarrow (B \rightarrow A)$

10. $\models(A \rightarrow (\neg A \rightarrow C)$

Logic functions called n-place operation on the set {0,1}.

Alphabet:

(1) $x,y,...,x_1,x_2,...$ - individual variables;

(2) $f,g,...,f_1,f_2,...$ - functional symbols.

Term:

(1) $x,y,...,x_1,x_2,...$ - individual variables are terms;

(2) If $f^{(n)}$ - a functional symbol, $t_1,...,t_n$ - terms, then

$f^{(n)}(t_1,...,t_n)$ - term.

The value of the term:

(1) if t - object variable x, then Val $t = \gamma(x)$;

(2) if $t = f^{(n)}(t_1,...,t_n)$, then Val $t = f^{(n)}$(Val t_1,..., Val t_n).

Function:

$f^{(n)}(x_1,...,x_n)$ can be represented by the term $t(v_1, ..., v_m)$, if $\{v_1, ..., v_m\} \subseteq \{x_1,...,x_n\}$ and $t[\gamma] = f^{(n)}[\gamma]$ for all interpretations $\gamma : \{x_1,...,x_n\} \Rightarrow \{0,1\}$.

Examples 3.1.

1.The four-digit number of the 1952 decimal system is expressed thus:

$$1952_{(10)} = 1*10^3 + 9*10^2 + 5*10^1 + 2*10^0$$

2.The number of a decimal system with a three-digit integer part and a three-digit fractional part 596.174 (10) is expressed as follows:

$$596.174_{(10)} = 5*10^2 + 9*10^1 + 6*10^0 + 1*10^{-1} + 7*10^{-2} + 4*10^{-3}$$

3. The number of a binary system with a four-digit integer part and a three-digit fractional part 1010.101 (2) is expressed as:

$$1010.101_{(2)} = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$$

Examples 3.2.

1. 3.14 - positive real number with fixed-point, the integer part 3, and the fractional part 14. 2. 5 - positive integer 5.

3. 0.2 - positive real number with fixed-point, the integer part 0 and fractional part 2.

4. -1.001 - negative real number with fixed-point, the integer part 1 and the fractional part of 001.

5. 0.0 - positive real number, the integer part 0 and the fractional part 0.

Tests 3.

1. What will be important expression 2>5 ∨ 2<6?

A) 2

B) 1

C) 5

D) 6

E) 0

2. What order of operations an expression D∨¬F ∧ G?

A) first ¬F, then ¬F * G, and at the end D∨¬F ^ G.

B) first ¬F ^ G, and at the end D∨¬F ^ G.

C) first ¬F, and at the end D∨¬F ^ G.

D) first ¬F, then ¬F ^ G.

E), first ¬G, then ¬F ^ G, and at the end D∨¬F ^ G.

3. Which one is De Morgan's law?

A) $\neg(\neg p) \equiv p$

B) $p \equiv p$

C) $\neg(p \vee q) \equiv \neg p \wedge \neg q$

D) $p \wedge \neg p \equiv 0$

E) $p \vee \neg p \equiv 1$

4. LAWS OF LOGIC

Lecture objective: explain the concept and definitions of the laws of logic and review their types.

Lecture plan: study the law of double negation, commutation law, distribution law, law of exclusion of constants, law of contradiction, law of excluded middle, the duality principle, logical corollary, rules of logical corollary, modus ponens rule.

Laws of logic **consist of the following tautologies:**

1) $\models A \vee \neg A$ (*law of excluded middle*)
2) $\models A \to A$ (*law of identity)*
3) $\models \neg(A \vee B) \sim \neg A \& \neg B$ (*first de Morgan's law*)
4) $\models \neg(A \& B) \sim \neg A \vee \neg B$ (*second de Morgan's law*)
5) $\models A \& A \sim A$, $\models A \vee A \sim A$
6) $\models A \to B \sim \neg A \vee B$
7) $\models (A \leftrightarrow B) \sim (A \to B) \& (B \to A)$
8) $\models (A \to B) \sim (\neg B \to \neg A)$ (contraposition law)
9) $\models A \& B \sim B \& A$ (*conjunction commutability*)
10) $\models A \vee B \sim B \vee A$ (*disjunction commutability*)
11) $\models A \& (B \& C) \sim (A \& B) \& C$ (*conjunction associativity*)
12) $\models A \vee (B \vee C) \sim (A \vee B) \vee C$ (*disjunction associativity*)
13) $\models A \& (B \vee C) \sim (A \& B) \vee (A \& C)$ (*first law of distributivity*)
14) $\models A \vee (B \& C) \sim (A \vee B) \& (A \vee C)$ (*second law of distributivity*)
15) $\models A \& (A \vee B) \sim A$, $\models A \vee (A \& B) \sim A$ (*absorption laws*)
16) $\models A \& И \sim A$, $\models A \& Л \sim Л$, $\models A \vee И \sim И$, $\models A \vee Л \sim A$.
17) $\models A \to (B \to C) \sim A \& B \to C$.

Let E be a formula with close negations which does not contain other operations except $\neg, \&, \vee$. The E^X formula is the result of substituting all conjunctions in E with disjunctions and each proposition letter with its negation. Then $\models \neg E \sim E^X$.

The duality principle. Let E, F not contain other operations except $\neg$, $\&$, $\vee$ and let them be formulas with close negations. The formulas E', F' obtained from E, F by simultaneous substitution of all $\&$ with $\vee$ and $\vee$ with $\&$ are called

dual with regard to the formulas E and F correspondingly. Then the following relations exist:

a) if $\models \neg E$, then $\models E'$. b) if $\models E$, then $\models \neg E'$.

c) if $\models E \sim F$, then $\models E' \sim F'$. d) if $\models E \rightarrow F$, then $\models F' \rightarrow E'$.

Logical corollary. Let there be formulas $A_1,A_2,...,A_m$ and B. If from the simultaneous truth of the formulas $A_1,A_2,...,A_m$ there follows the truth of the formula B, then the formula B is a logical corollary of the formulas $A_1,A_2,...,A_m$; this is indicated as $A_1,A_2,...,A_m \models B$, $(m \geq 1)$, where $A_1,A_2,...,A_m$ are premises and B is a corollary.

Logical corollary rules. For computation of relations one single rule called modus ponens is used which represents a procedure of transition from two formulas of the type $A, A \rightarrow B$ (premises) to the formula B (corollary):

$$\frac{A,\ A \rightarrow B}{B}$$ (modus ponens)

Corollary rules must satisfy the requirement that true premises lead to true corollaries.

Predicates are logical functions $J^{(n)}(x_1,...,x_n)$ *given in a non-empty space* D *and acquiring value in the set* {И,Л}.

The predicate $J^{(n)}(x_1,...,x_n)$ becomes an expression after its variables are attributed to the elements of the set D.

***Alphabet*:**

(1) $x,y,z,...,x_1,x_2,...$ – object variables;

(2) $P^{(n)}(x_1,...,x_n),...$ – predicate letters $(n=0,1,...)$;

(3) &, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$ – logical connectives and quantors;

(4) (,) – auxiliary symbols.

Formulas:

(1) $P^{(n)}(x_1,...,x_n)$, …– elementary formulas or atoms;

(2) if A, B are formulas, then $A\&B$, $A \vee B$, $\neg A$, $A \rightarrow B$, $A \leftrightarrow B$ – are formulas as well;

(3) if $A(x)$ is a formula with a free variable x, then $\forall x A(x)$, $\exists x A(x)$ are formulas.

Free and bound variables. All variables existing in the space of action of the quantor at such variables are called *bound variables, otherwise they are called free variables.*

Formula interpretation. The value of the formula $E[P_1,...,P_m; x_1, ...,x_n]$ for interpretation of the predicate letters τ: $P^{(n)} \Rightarrow J^{(n)}$ and attribution of γ: $\{x_1,...,x_n\} \Rightarrow D$ $(D \neq \varnothing)$ to object variables is denoted $E[\tau,\gamma]$. Let us define induction for construction of the formula E:

1) $E = P^{(n)}(x_1,...,x_n)$, then $E[\tau,\gamma] = J[\gamma]$;

2) $E = (A\&B)[P_1,...,P_m; x_1,...,x_n]$, then $E[\tau,\gamma] = A[\tau,\gamma] \& B[\tau,\gamma]$.

Analogously for other logical connectives.

3) $E=\forall x_1 A[P_1,...,P_m;x_1,...,x_n]$, then $E[\tau,\gamma] = \forall x_1 A[\tau,x_1,\gamma]=И$,

where γ: $\{x_2,...,x_n\} \Rightarrow D$, if $A[\tau,a,\gamma] = И$ for any $a \in D$.

4) $E=\exists x_1 A[P_1,...,P_m; x_1, ...,x_n]$, then $E[\tau,\gamma] = \exists x_1 A[\tau,x_1,\gamma] = И$,

where γ: $\{x_2,...,x_n\} \Rightarrow D$, if $A[\tau,a,\gamma] = И$ for some $a \in D$.

The formula $E[P_1,...,P_m; x_1,...,x_n]$ is called a universally valid formula or tautology if for any space $D \neq \varnothing$, for any interpretations τ of predicate letters and any attributes γ to object variables in interval D, $E[\tau,\gamma] = И$.

Logical foundations of computer consist of logic algebra which emerged in mid-19th century in the works of English mathematician John Boole. Its creation was due to an attempt to solve traditional logical problems by algebraic methods using logical operations such as $\neg$, &, $\vee$ denoting words and word combinations "*not*", "*and*", "*or*"". With help of these logical operations a logical expression of any complexity may be constructed.

Hardware implementation of the mentioned logical operations is realized by means of the following logical elements of computer shown in figure 4.

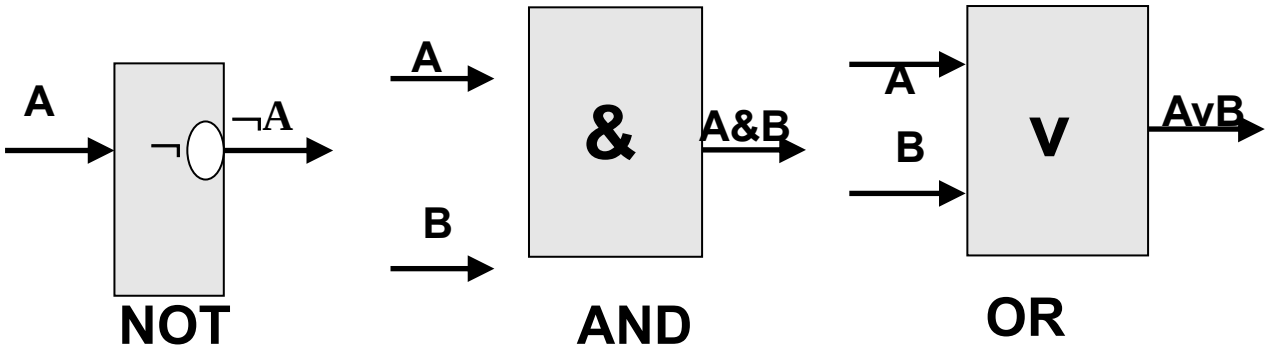


Figure 4. Logical elements of computer.

Examples 4.1.

Let us show that the formula $P(x,y) \rightarrow Q(x)$ is not 1-valid and, consequently, not universally valid.

Solution. D={1} is one-element set, I_1 and I_2 – interpretations of the letter P, and J_1 and J_2 – interpretations of the letter Q:

x	y	I_1	I_2	J_1	J_2
1	1	И	Л	И	Л

Truth-table of the formula $P(x,y)\to Q(x)$:

x	y	$P(x,y)$	$Q(x)$	$P(x,y)\to Q(x)$
1	1	И	И	И
1	1	И	Л	Л
1	1	Л	И	И
1	1	Л	Л	И

Examples 4.2.

Let us show that the formula $\forall x\exists y P(x,y)\to\exists y\forall x P(x,y)$ is not universally valid.

Solution. Let D={1,2}, then the interpretations of the predicate letter $P(x, y)$ may be given by means of the following table:

X	Y	J_1	J_2	J_3	J_4	…	J_7	…
1	1	И	И	И	И	…	И	…
1	2	И	И	И	И	…	Л	…
2	1	И	И	Л	Л	…	Л	…
2	2	И	Л	И	Л	…	И	…

In particular, for interpretation J_7 we obtain: for x=1: $\exists y J_7(1,y)\equiv$И; for x=2: $\exists y J_7(2,y)\equiv$И, then $\forall x\exists y J_7(x,y)=$И. For y=1: $\forall x J_7(x,1)=$Л, for y=2: $\forall x J_7(x,2)=$Л, then $\exists y\forall x J_7(x,y)=$Л. It follows that $\forall x\exists y J_7(x,y)\to\exists y\forall x J_7(x,y)$ = Л.

Examples 4.3.

Let us show that the formula $\forall x(\exists x P(x)\to P(x))$ is not 2-valid.

Solution. D={1,2}, J_1, J_2, J_3, J_4 – interpretations of the letter P :

x	J_1	J_2	J_3	J_4
1	И	И	Л	Л
2	И	Л	И	Л

Truth-table of the formula x ($\exists x P(x) \rightarrow P(x)$):

	$P(x)$	$\exists x P(x)$	$\exists x P(x) \rightarrow P(x)$	$\forall x(\exists x P(x) \rightarrow P(x))$
	J_1 J_1	И	И И	И
	J_2 J_2	И	И Л	Л
	J_3 J_3	И	Л И	Л
	J_4 J_4	Л	И И	И

Examples 4.4.

Let P be a false statement 1 = 5, Q is a false statement as well 3 = 7 and R is a true statement 4 = 4. Demonstrate that conditional statements: «if P, then Q» and «if P, then R» are both true.

Solution. If 1 = 5, then adding 2 to both parts of the equality we obtain 3 = 7. Therefore, the statement «if P, then Q» is true. Now let us subtract 3 from both parts of the equality 1 = 5 obtaining –2 = 2. Therefore, $(-2)^2 = 2^2$, i.e. 4 = 4. Therefore, «if P, then R» is true as well.

Problems 4.

1. Translate each of the following arguments into logical symbols and analyze the correctness of the result:

1) I would pay for television repair only if it functioned. It does not. For this reason, I will not pay.

2) If he had told her nothing, she would never have found it out. And if she had not asked him, he would not have told her. But she found it out. Therefore, she asked him.

3) He said he would come if it did not rain. But it is raining. Therefore, he will not come.

2. Check the correctness of argument: Ivanov will not do this work if Petrov does it. Petrov and Sidorov will do this work if and only if Ivanov does it. Sidorov will do this work, and Ivanov will not. Therefore, Petrov will not do this work.

3. Which formulas yield the following formula sequences: A⊃(B⊃C), A, B⊃C, B, C.

Questions 4.

1. Are the following expressions equivalent?
   1) A∧B and A and B?
   2) A∧B and not only A, but also B?
   3) A∧B and B, even though A?
   4) A∧B and B, in spite of A?
   5) A∧B and both A, and B?
2. Are the following expressions equivalent?
   1) A∨B and A or B?
   2) A∨B and A or B?
   3) A∨B and A, if not B?
   4) A∨B and A and B?
   5) A∨B and A or B?
3. Are the following expressions equivalent?
   1) A∼B and A, if and only if B?
   2) A∼B and if A, then B, and vice versa?
   3) A∼B and A, if B, and B, if A?
   4) A∼B and A equivalent to B?
   5) A∼B and A if and only if B?
4. For which of the statements *X*: *X=1, X=6, X=5, X=3, X=4* are the relations (*X*>3) & (*X*<5) true?
5. For which of the words “Informatics”, “Psychology”, “Economics” will the statement “The first letter is consonant, and the second letter is a vowel” be true?
6. Which of the following statements are true, and which are false?
   (a) The sum of interior angles of any triangle is 180°.
   (b) All cats have a tail.

(c) There is an integer x satisfying the equation $x^2 = 2$.

(d) There is an even prime number.

(e) Snow is white.

(f) The Earth revolves around the Moon.

(g) Paris is the capital of France.

(h) To govern is to know.

Tests 4.

1. What characterizes the law of excluded middle?

1) Implication of two statements is equivalent to the inverse implication of their negations.

2) Any statement is either false or true, no third possibility exists.

3) Any statement is the logical corollary of itself.

4) To negate a negation of a statement is equivalent to its assertion.

2. Interpretation is:

5) Concepts whose application to logical calculation expressions depends in great measure on the choice of interpretation.

6) Juxtaposition of every elementary expression p with a certain true value.

7) Concepts whose application to logical computation depends in great measure on the choice of interpretation.

8) Relation between objects which means that the state or properties of any of them change if the state or properties of others are changed.

3. Is the logical connective «or»:

1) connective?
2) exclusive?
3) divisive?
4) auxiliary?
5) negating?

4. What characterizes the law of double negation:

1) Any statement is either false or true, no third possibility exists.

2) Any statement is the logical corollary of itself.

3) To negate a negation of a statement is equivalent to its assertion.

4) Any statement is the logical corollary of itself.

5. GRAPHS

The purpose of the lecture: to consider the concept of the graph, the types of graphs and their properties.

Outline of the lecture: to explore formal definitions and ways to represent graphs, to analyze different types of graphs and types of applications of graphs for various tasks.

Definitions 6.1:

The graph is a dynamic networking connected structure of data represented by of a plurality of pairs called vertices and edges. Each vertex can be connected with several other vertices or with itself by means of edges and vertices, which do not form a hierarchy. Formally, a graph is defined as a set of pairs of G = (X, A), where X - the set of vertices, A - the set of edges, actually is a relation on a set X, i.e. $A \subseteq X \times X$. If $x_i \in X$ and $x_j \in X$ – vertices , then (x_i, x_j) – edges.

There are several types of graph. If from each vertex of the graph originates equal number of edges and if equal number of edges goes in each vertex, such a graph is a regular graph. If for each edge of the graph direction is defined, the graph is called a directed graph. If each edge of the graph has a weight, a graph is called weigthed graph, i.e., you can define a function w : E, where R - the set of real numbers, w -weight of graph and w≥0.
Matrix of adjacency is one of the ways to represent a graph in the form of a matrix.

Matrix of adjacency of a graph G with a finite number of n vertices (numbered from 1 to n) is a square matrix A of size n, wherein the value of element a_{ij} equals to number of edges from the i-th vertex in the j-th vertex. Sometimes, especially in the case of an undirected graph, the loop (the edge of the i-th vertex in itself) counts as two edges, i.e., the value of the diagonal element a_{ij} in this case equals to double number of loops around the i-th vertex.

Matrix of adjacency of a simple graph (not containing loops and multiple edges) is a binary matrix which contains zeros on the main diagonal. In graph theory are used following:

– Incidence matrix. This matrix A with n rows corresponding to the vertices and m columns corresponding to the edges. For a directed graph column

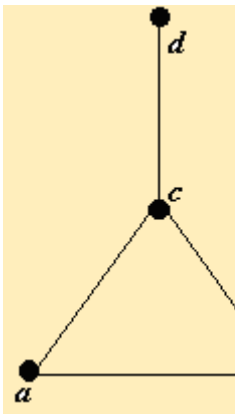
corresponding to the arc (x, y) contains - 1 in the row corresponding to vertex x, and 1 in the row corresponding to the vertex y. In all others 0. Loop, i.e. arc (x, x) may be represented by a different value in the row x, e.g., 2. If an undirected graph, the column corresponding to the edge (x, y) contain 1, the corresponding x and y and zeros in all other rows.

– The matrix of adjacency. This is a matrix n × n where n - the number of vertices, where a_{ij} =1, if there is an edge going from vertex x to vertex y and a_{ij}=0 otherwise, i.e.:

a_{ij} - the number of edges connecting vertices v_i and v_j, and in A) in some applications of each loop (an edge $\{v_i, v_i\}$ for some i)) is counted twice;

B) adjacency matrix of empty graph, does not contain any edges, consists of zeroes.

Below are examples of incidence matrix of and adjacency matrix for continuous graph shown in Figure 6.1



	u	v	w
a	1	1	0
b	1	0	1
c	0	1	1
d	0	0	0

	a	b	c	d
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

Figure 6.1 Incidence matrix Adjacency matrix

Given a graph $G=(X, A)$, where $X=\{x_i\}$, i = 1, 2, ..., n – the set of vertices, $A=\{a_j\}$, j = 1, 2, ..., m – the set of arcs.

Subgraph $G'=(X', A')$ of the original graph G is a graph G ', for which $X' \subseteq X$ и $A' \subseteq A$. Examples of subgraphs are shown in Fig. 6.2, b, and original graph - Fig. 6.2 a.

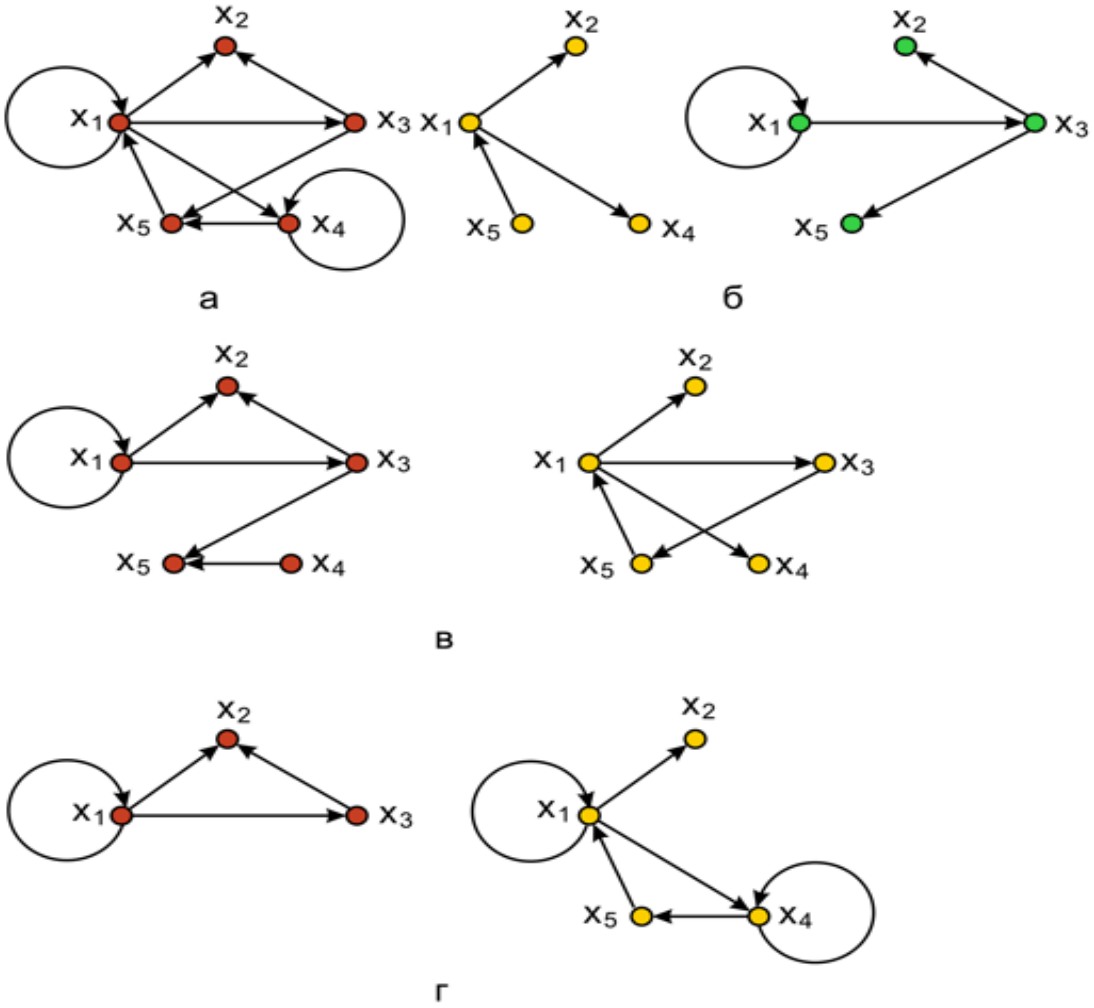


Figure 6.2. Types of subgraphs: а - the original graph; б - subgraphs; в - spanning subgraph; г - induced subgraphs

If A - adjacency matrix of the graph G, the matrix A^n has the following property: item at the i-th row, j-th column is equal to the number of paths from the i-th vertex to the j-th consisting of exactly n edges.

The path in a graph is a sequence of edges leading from one vertex to another vertex, such that every two neighboring edges have a common vertex and no edge occurs more than once, that is, formal path in a graph is a sequence of vertices $(x_1, x_2, x_3, ..., x_{m-1}, x_m)$, that pairs $\{(x_1, x_2),(x_2, x_3),...,(x_{m-1}, x_m)\}$ will be edges. Two vertices $x_i \in X$ and $x_j \in X$ in the graph is called connected (disconnected), if it exists (do not exist) the path leading from x_i to x_j. This path can be in both directions. If every two vertices in the graph are connected, then this graph is a connected graph. If the graph contains at least one pair of

disconnected vertices, the graph is disconnected. If all pairs of vertices connected in both directions, so the graph is strongly connected graph.

The path with no repeated edges is called a chain and the chain without repeated vertices called simple.

Chain in which the end vertices coincide is called *a cycle*, and the cycle in which no recurring peaks other than the end, called *simple*, i.e. the path way back to the same vertex, then that path is called the *closure (cycle)*, i.e. in the closure of the initial and final vertices are the same. If the closure does not pass through one of the vertices of the graph more than once, it is called a *simple closure*. If the closure originates from a single vertex and directly enters into the top back, it is called *a loop*, i.e, the loop has a unique vertex.

The length of the path is the number of edges of this path. If the weights of the edges are their length, then the path length is calculated as follows:

$$w(x_1, x_2, x_3, ..., x_{m-1}, x_m) = \sum_{i=1}^{m-1} w(x_i, x_{i+1}).$$

In the graphs you can perform the following tasks: *a comparison of the two graphs, finding the shortest path from one vertex to another, finding the number of closed paths* and etc.

A tree is a graph in which all vetices are connected, and the paths are not closed, i.e., connected graph is without cycles and without loops.

The tree vertices are divided into the following types:

1) *the root* – a vertex, from which originates one or more edges, but enter no edge, i.e., a vertex, which does not have a single ancestor, but it can have many descendants;

2) *branch* - the vertex, to which enters a single edge, but many egdes can originate from it, i.e., the veretx which has a single ancestor and can have many descendants;

3) *sheet* - the vertex, to which enters only one edge, but originate no edge, i.e. the vertex which has a single ancestor, but does not have any descendants.

In the tree the direction of path passes through the branches from the root to the leaves. Inside the tree can be a few trees, which will be called *subtrees*. You can now give the following recursive definition (referring to itself):

1. *A recursive basis*: the set {v}, consisting of only one vertex v is a tree where its unique vertex is both the root and leaf.

2. *Recursive step:* if v - vertex and $A_1, A_2, ..., A_n$ - the trees, then it is possible to build a new tree in which the root is the vertex v, and edges – originates from this vertes and enters the roots of $A_1, A_2, ..., A_n$ trees.

3. *Recursive conclusion:* Trees obtained only by rules 1 and 2. This definition of a tree can be represented in Figure 6.3 as follows:

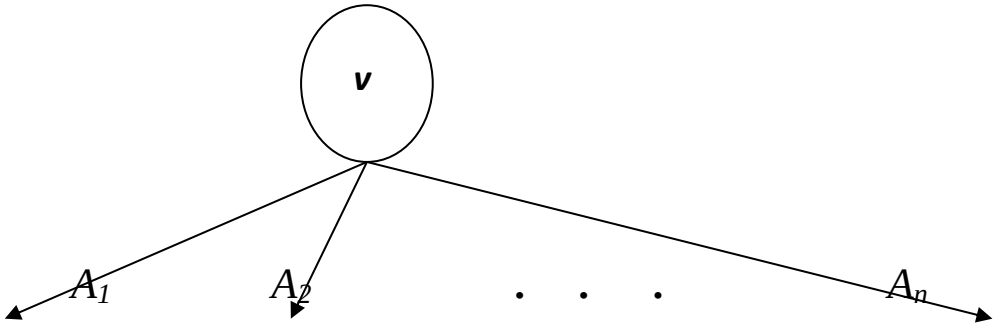


Figure 6.3. Determination of tree

From this definition it is clearly evident that the tree is a hierarchical connected dynamic structure of data represented by single root vertex and its descendants. The maximum number of descendants of each vertex and determines the size of a tree.

Among the trees stands out, the so-called *binary trees*. It can be defined as follows:

Binary Tree - a tree in which each node has at most two descendants. This node is called the *parent node* and the descendants are called *left heir* and *right heir*. We give a recursive definition of a binary tree. A binary tree is the following set of vertices:

– either contains nothing (the empty set);
– or consists of a root, which is connected with two binary trees, called left-hand subtree and right-hand subtree.

Thus, the binary tree is either empty or consists of data and two subtrees, each of which may be empty. If in some vertex two subtrees are empty, then it is a leaf. Formally, a binary tree is defined as follows: <binary tree> :: = nil | (<data> <binary tree> <binary tree>) where nil - empty.

The following tasks are solved in trees: *tree traversal, search for tree, adding a new node to the tree, destroying the tree tops, comparisons of trees* and others.

Binary trees are used in the search algorithms: each vertex of binary search tree corresponds to an element of a sorted set, all his left descendants the left to fewer elements, and all his right descendants to a great element. Each node in the tree is uniquely identified by a sequence of non-recurring vertices from the root and until it – by path. The path length is a level of node in the hierarchy tree. For practical purposes, generally two subspecies of binary trees are used: binary search tree - binary search tree (BST) and binary heap.

Binary search tree has the following properties:
– the left subtree and the right subtree are binary search trees;
– all the vertices of the left subtree of v arbitrary vertex has value of key of data that is less than the value of key of data of the vertex v itself;
– all the vertices of the right subtree of the same vertex v has value of key of data that is greater than the value of key of data of vertex v.

Clearly, data from each node should have keys on which the comparison operation is determined.

Binary heap or sorting tree has the following properties:
– value at any vertex is not less than the values at the vertices of its descendants;
– leaf depth (distance until the root) does not differ by more than one layer;
– the last layer is filled from left to right.

Such heap is called *max-heap*. There are also heaps, where the value in each vertex, conversely, no more than the values of its descendants. Such heaps are called *min-heap*.

Examples 6.2:

1. A binary relation over finite objects can be represented as a directed graph as shown in Figure 6.4. The following shows the relationship divisibility of integers from 1 till 12: 2 and 3 divided by 1; 4 and 6 is divided into two; 6 is divisible by 2 and 3; 12 divided by 4 and 6.

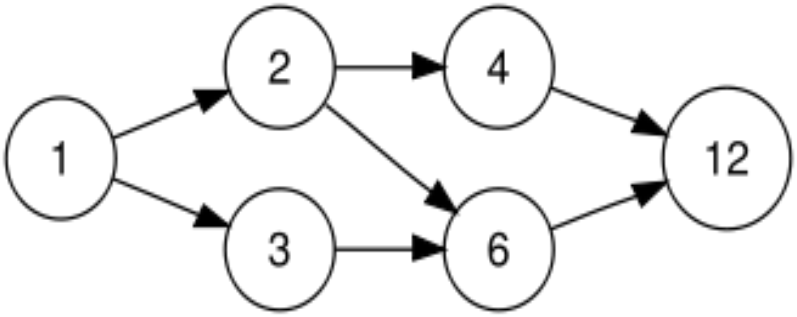


Figure 6.4. Representation of binary relation

2. Presentation of a binary tree shown in Figure 6.5.

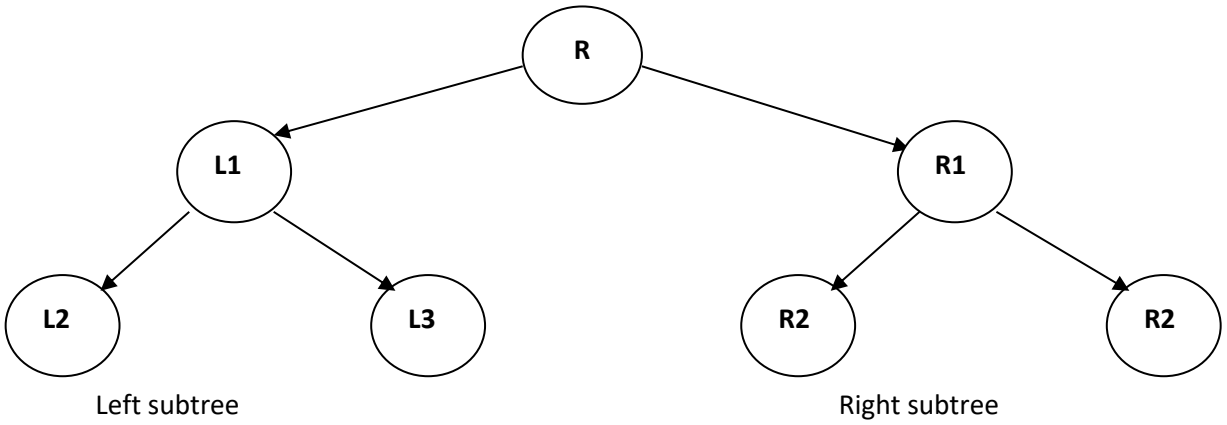


Figure 6.5. A binary tree.

3.Bypass of binary tree of arithmetic expression

((3 + 1) * 3 / (9-5) 2 + (3 * (7-4) 6)

from the top to the bottom and from the left to the right is shown in Figure 6.6.

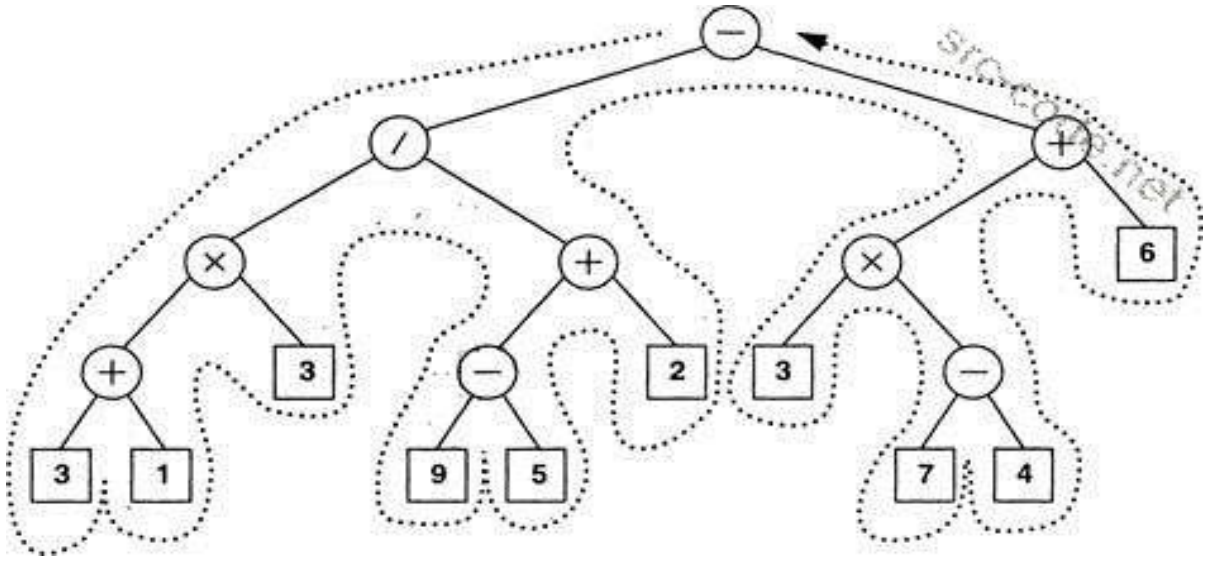


Figure 6.6. Bypass of tree

Exercises 6.1:

1. Build a directed weighted graph for describing the structure of identifier.
2. Build the tree for the expression ((a / (b + c)) + (x * (y - z))).
3. Determine the adjacency matrix A of an undirected graph that contains a loop around the vertex one, which depending on the application element a_{11} may be considered equal to one (as shown below), or to two.

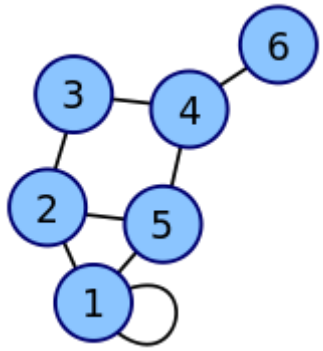


Figure 6.1. Undirected graph

Help:

1. Without loss of generality, to facilitate the construction of the desired graph we will consider not letters, but only one letter not numbers, only one number, which will serve as weight for required weighted graph.
2. In the corresponding binary tree, leaves are operands, and other vertices are operations.
3. The adjacency matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Exercises 6.2:

On a finite set N = {1, 2, 3, 4, 5} is given binary relation. R = {(1,2), (1,4), (1,5), (2,3), (3,2), (3,4), (4,4), (4,5) , (5,3), (5,4)}. Record domain and the range of values for this relation. Draw a graph of this relation. Make up adjacency and incidence matrix for it.

Question 6:

1. How path is formed in the graph?

2. What edges are called multiple?

3. What vertex is called an isolated?

4. What is the level of the isolated vertices?

5. What means the level of vertex?

6. What graph is called a cyclic?

7. What is the incidence matrix?

Test 6:

1. What are the types of graphs?

A) directed graph, undirected graph;

B) directed graph, defined graph;

C) specified graph, undirected graph;

D) specified graph, unsepcified graph;

E) unspecified graph, undirected graph.

2. What is a tree?

A) graph without loops and cycles;

B) graph without weights;

C) graph without networks and cycles;

D) weighted graph;E) directed graph.

3. What is a binary tree?

A) tree in which each vertex has at most two descendant;

B) tree, which has two vertices;

C) tree, which has no cycle;

D) tree, which has no loop;

E) tree, in which one vertex has no direct descendants.

6. FORMAL GRAMMARS

6.1. General information

In this section, formal grammars will be considered mechanisms of language generation, the relationship of derivability and language generated by formal grammar, examples are proposed, tasks are given, questions and tests are formulated. For the preparation of educational materials, sources were used [1-9,11-18,21,25,27-32].

An important class of mechanisms for the generation of languages is formed by formal grammars (Formal Grammar), which were first introduced in 1959 by the American linguist Chomsky [24].

The formal grammar that generates the language L uses two disjoint sets of symbols:

1) A finite set of terminals (terminals) – constants T, from which chains of the language L are formed;

2) A finite set of nonterminals – variables N disjoint with the set T, which denote grammatical concepts, categories, etc. language L.

3) The process of generating L strings is described by a finite set of rewriting rules P, each of which consists of pairs of strings (α, β). In such a pair, the first component α is a string containing at least one nonterminal, and the second component can be any string formed from terminal and / or nonterminal symbols. It can also be an empty chain.

Agreements 6.1.1. The following agreements are accepted:

(1) lowercase latin cursive letters *a*, *b*, ..., *z* and Arabic numerals 0, 1, ..., 9 designate terminals;

(2) uppercase latin cursive letters *A, B, ..., X, Y, Z* denote nonterminals, while *S* denotes the initial nonterminal symbol;

(3) lowercase Greek letters α, β, ..., ω denote strings that can contain both terminals and nonterminals, here ε is an empty string;

(4) the substitution rule, which is a pair of chains (α, β) from the set P, is written as $\alpha \to \beta$;

(6) rules of the form $\alpha \to \varepsilon$ are called ε (epsilon) -rules;

(7) these agreements also apply to letters with subscripts and superscripts;

(8) rules of the form $\alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m \to \beta$ is a cancellation of m rules of the form $\alpha_1 \to \beta$, $\alpha_2 \to \beta$, …, $\alpha_m \to \beta$ or: $\alpha_1 \to \beta$, $\alpha_2 \to \beta$, …, $\alpha_m \to \beta$

(9) rules of the form $\alpha \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ is an abbreviation of n rules of the form $\alpha \to \beta_1$, $\alpha \to \beta_2$, …, $\alpha \to \beta_n$ or: $\alpha \to \beta_1$, $\alpha \to \beta_2$, …, $\alpha \to \beta_n$

(10) a rule of the form $\alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_{m \to} \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$ is an abbreviation of m × n rules obtained from agreement (6) and (7)

Definition 6.1.1. A formal grammar is the following quadruple $G = <T, N, P, S>$, where:

T is a non-empty finite set of terminal symbols (terminals);

N is a nonempty finite set of nonterminal symbols (nonterminals), and $T \cap N = \emptyset$, $\emptyset$ is the empty set;

P is a non-empty finite set of permutation rules of the form $\alpha \to \beta$, where $\alpha \in (T \cup N)^* \times N \times (T \cup N)^*$, $\beta \in (T \cup N)^*$, that is,

$P \subseteq \{(\alpha,\beta): \alpha \in (T \cup N)^* \times N \times (T \cup N)^* \& \beta \in (T \cup N)^*\}$;

S is the initial nonterminal, $S \in N$.

The inference rules of a grammar can be viewed as elementary operations that, when applied in a certain sequence to the original string, generate only correct strings. The very sequence of rules used in the process of generating a certain chain is the output of this chain.

A grammar-defined language is a set of finite strings that consist only of terminals. All these terminal chains are deduced starting with one special chain, consisting of only one initial nonterminal S.

The inference rules of a grammar can be viewed as elementary operations that, when applied in a certain sequence to the original string, generate only correct strings. The very sequence of rules used in the process of generating a certain chain is the output of this chain.

A grammar-defined language is a set of finite strings that consist only of terminals. All these terminal chains are deduced starting with one special chain, consisting of only one initial nonterminal S.

Examples 6.1.1. Grammar G = <T, N, P, S> with parameters: $P = \{S \to E, E \to E+ V, \mid E- V, \mid V, V \to V*F \mid V/F \mid F, F \to a \mid (E)\}$, $N = \{S, E, V, F\}$, $T = \{+, -, /, *, (,), a\}$ generates a parenthetical algebraic expression in infix notation.

To define a language with the help of grammar, the notion of a *derivable string* and an *immediate derivability relation* are used.

Definitions 6.1.2:

1. Let α, β, γ be derivable strings of the grammar G = <T, N, P, S>. Then the outputted strings are recursively defined as follows:

1) S is the outputted string of the grammar G;

2) If αβγ is a deducible chain of G and P has a rule β → δ, then αδγ is also a deducible chain of G.

2. The deduced string of the grammar G that does not contain nonterminal symbols from N is called the *terminal string* generated by the grammar G.

3. If α = γξδ, β = γηδ and α → β, ξ → η are the inference rules of the grammar G, then it is said that an immediate derivability relation is established between the strings α and β, which means that in the grammar G the string β is directly derived from of the chain α by replacing ξ with η, and this relation is denoted by $\alpha \Rightarrow_G \beta$. If the grammar is known in advance, then the exponent G in relation to direct deducibility is omitted and this relation is written as α⇒β.

A notation of the form $\alpha \Rightarrow^k \beta$ is the k-th power of the relation α⇒β, If there are k+1 chains $\alpha_0, \alpha_1, ..., \alpha_k$ such that $\alpha=\alpha_0$, $\alpha_k=\beta$ and $\alpha_{i-1} \Rightarrow \alpha_i$ ($1 \le i \le k$) This sequence of strings is called *the derivation of length k* of the string β from the string α in the grammar G.

If there exists $i \ge 1$ (or $i \ge 0$) the relation $\alpha \Rightarrow^i \beta$ holds, then this is written as $\alpha \Rightarrow^+ \beta$ (or $\alpha \Rightarrow^* \beta$). Here, $\Rightarrow^+$ denotes the transitive closure of the relation ⇒, and ⇒* denotes the reflexive and transitive closure of the relation ⇒. In this case, the notation of the form $\alpha \Rightarrow^+ \beta$ (α⇒*β) reads as: "β is *deducible* from α in a *non-trivial* way" ("β is deducible from α").

Remark 6.1. $\alpha \Rightarrow^* \beta$ if and only if $\alpha \Rightarrow^i \beta$ for some $i \ge 0$, and $\alpha \Rightarrow^+ \beta$ if and only if $\alpha \Rightarrow^i \beta$ for some $i \ge 0$.

Definitions 6.1.3.

1. Each string that is derived from the initial nonterminal of the grammar is called a *sentence form*.

2. Output strings that do not contain nonterminal symbols are called *terminal strings*. Therefore, the language L(G) can be defined as the set of terminal strings deduced in the grammar G.

3. The language L(G) generated by the grammar G is the set of terminal strings that are derived from one initial nonterminal S by applying the substitution rule from the set P, that is, formally written as $L(G) \rightleftharpoons \{\tau:\ \tau \in T^*, S \Rightarrow^* \tau\}$.

This means that any string belonging to the language L(G) is a sentence form.

Examples 6.1.2.

1. Let the grammar $G_1 = <T, N, P, S>$ be given, where $T = \{0,1\}$ is the set of terminals, $N = \{A, S\}$ is the set of nonterminals, $P = \{S \rightarrow 0A1, 0A \rightarrow 00A1, A \rightarrow \varepsilon\}$ is a set of substitution rules. If we consider an inference of the form S⇒0A1⇒00A11⇒0011, then we can see that at the first step the nonterminal S is replaced by the chain 0A1 by the rule S → 0A1, at the second step the chain 0A by the rule 0A → 00A1 is replaced by the chain 00A11, and at the third step nonterminal A is replaced by the empty string ε by the rule A → ε. Thus, we can say that $S \Rightarrow^3 0011$, $S \Rightarrow^+ 0011$, $S \Rightarrow^* 0011$ and the string 0011 belongs to the language $L(G_1) = \{0^n 1^n:\ n>1\}$.

2. A grammar with rules $P_1 = \{S \rightarrow 01S, S \rightarrow 0\}$ and a grammar with rules $P_2 = \{S \rightarrow 0A, A \rightarrow 10A, A \rightarrow \varepsilon\}$ are equivalent.

3. Two grammars for generating algebraic expressions formed by operands *i*, *n* and operations +, * with the same terminal symbols $T = \{i, n, (,), +, *\}$ and nonterminal symbols N = {*S, F, H*} but with different rules: $P_1 = \{S \rightarrow S+F|F,\ S \rightarrow S+F|S*F|F,\ F \rightarrow F*H|H,\ H \rightarrow i|n|(S)\}$ and $P_2 = \{S \rightarrow S+F, S \rightarrow F, F \rightarrow F*H, F \rightarrow H, F \rightarrow H, H \rightarrow i, H \rightarrow n, H \rightarrow (S)\}$

are equivalent.

With the help of formal grammars, it is possible to generate various classes of languages by imposing restrictions on their inference rules:

1. Any grammar whose inference rules do not impose any restriction belongs to class 0 and is called an *unrestricted grammar* (NG), is an unrestricted grammar, and the set of strings generated by this grammar will be a *recursively enumerable language*.

2. A grammar in which all inference rules of the form α → β are constrained $\alpha = \xi H \zeta$, $\beta = \xi \eta \zeta$, $\xi \in (T \cup N)^*$, $H \in N$, $\eta \in (T \cup N)^+$, $\zeta \in (T \cup N)^*$ belongs to class 1 and is called a *context-sensitive grammar* (CSG), and the set of strings generated by this grammar will be a *context-sensitive language*.

3. A grammar in which the constraint $A \in N$, $\alpha \in (T \cup N)^*$ is imposed on all inference rules of the form $A \to \alpha$ is of class 2 and is called a context-free grammar (CFG) and the set of strings generated by this grammar will be Context-free language.

4. A grammar in which all inference rules have the form $A \to \alpha B \beta$ or $A \to \alpha$, where $A, B \in N$, $\alpha, \beta \in T^*$ is of class 3 and is called a *linear grammar* (LG). In a linear grammar, If β = ε, then it will be a *right-linear gramma*r (RLG), and If α = ε, then it will be a *left-linear grammar* (LLR). The set of strings generated by the left-linear grammar will be called the left-linear language, and the set of strings generated by the right-linear grammar - the right-linear language.

Remarks 6.1.2.

1. In some sources, context-free grammar is called context-free grammar (CFG), context-sensitive grammar is called context grammar (CG) or non-truncating grammar or grammar of the immediate components.

2. Each linear grammar is a context-free grammar.

3. Every contextless grammar is a contextual grammar.

4. Each contextual grammar is a grammar without limitation.

5. Any linear language is its own subset of a contextless language, but a contextless language may not be linear.

6. Any context-free language that does not contain an empty string will be its own subset of the context language.

7. Any contextual language is contained in a recursively enumerated language.

8. If L_0, L_1, L_2, L_3 are languages generated by grammars of type 0, 1, 2, 3, respectively, then $L_3 \subseteq L_2 \subseteq L_1 \subseteq L_0$ is true.

Examples 6.1.3.

1. The language $\{a^n b^n c^n, n \geq 0\}$ generated by a grammar with inference rules $S \to aSBC$, $S \to aBC$, $CB \to BC$, $aB \to ab$, $bB \to bb$, $bC \to bc$, $cC \to c$ will be recursively enumerable. Here you can apply the rules in any order, only it is necessary to apply the rule $S \to aBC$ (for fixing n), and the rule $bC \to bc$ should be applied only after there are no B to the right of C (otherwise this B cannot be replaced by b and the output will not end with a terminal chain).

2. The set of Boolean formulas given by the variables a, b, c will be a context-free language, since it is generated by a context-free grammar $\boldsymbol{P}$ = {$S\to\neg S$, $S\to S\wedge F$, $S\to S\vee F$, $S\to F$, $F\to a|b|c$, $F\to (S)$}, $\boldsymbol{N}$={S, F}, $\boldsymbol{T}$={a, b, c, ¬, ∧, ∨, (,)}

3. Let G = <{S}, {a, b}, {$S\to aSa$, $S\to b$}, S>. Then $aSa\Rightarrow^3 aaaaSaaaa$.

4. A grammar with such rules $S\to ASQA$, $S\to AbA$, $A\to a$, $bQ\to bb$, $AQ\to UQ$, $UQ\to UV$, $UV\to QV$, $QV\to QA$ is contextual, but not context-free, since the last five rules do not have the required form.

5. A grammar with the following rules $S\to QS$, $S\to US$, $S\to b$, $Qb\to Ab$, $A\to a$, $QA\to AAQ$, $UAb\to b$, $UAAA\to AAU$ is not context free, since the last 3 rules do not have the required form.

Definitions 6.1.4.

1. If the language $\boldsymbol{L}(\boldsymbol{G})$ generated by the grammar G does not contain any finite string (final word) of terminal symbols, then it is called an *empty language*, that is, $L(G) = \emptyset$.

2. For the language $\boldsymbol{L}(\boldsymbol{G})$ to be non-empty, there must be at least one rule of the form $\xi\to\omega$ and there must be a derivation $S \Rightarrow^* \xi$, where $S\in N$ is the initial nonterminal, $\xi \in(T \cup N)^*\times N\times(T \cup N)^*$, $\omega\in T^*$.

3. If in the grammar G the inference rules form a closed loop, then such a grammar generates an infinite language, i.e. $L(G)=\infty$;

4. If $\xi\in L(G)$ holds for any string ξ and a given grammar G, then ξ is a chain in the language $L(G)$;

5. If for any two grammars $G'$and G'', $L(G')=L(G'')$ is satisfied, then grammars $G'$and $G''$are *equivalent*.

Examples 6.1.4.

1. A grammar with rules $S \to Q$, $U \to abba$ generates an empty language, denoted as $\emptyset$.

2. A grammar with rules $S \to aS$ generates an infinite language denoted as ∞;

3. A grammar with rules $S\to abS$, $S\to a$ and a grammar with rules $S\to aU$, $U\to baU$, $U\to\varepsilon$ are equivalent.

Considering the above, the following algorithmic problems of grammars can be considered:

1. *The problem of emptiness* - for a given grammar *G*, find out whether *L(G)* is an empty language, i.e. *L(G)* = *Ø*?

2. *The membership problem* - for any string ξ, find out whether it belongs to the language *L(G)* generated by a given grammar *G*, that is, ξ∈*L*(*G*)?

3. *The problem of equivalence* - for any two grammars *G'* and *G''* find out whether they will be equivalent, i.e. *L*(*G'*)= *L*(*G''*)?

4. *The problem of closedness* - when applying a multiple operation to languages of a certain type, find out if the result will have the same type?

5. The infinity problem - for a given grammar *G*, find out whether *L*(*G*) will be an infinite language, that is, *L*(*G*) = ∞?

Tasks 6.1.

1. Construct all sentences for grammar with rules:
   $S \to A+B|B+A,\ A \to a,\ B \to b$.
2. Build the output of the given chain *a-b*a+b* for grammar with rules:
   $S \to K|F+S|K-S,\ K \to F|F*K,\ F \to a|b$.
3. Build the output of the given chain *aaabbbccc* for grammar with rules:
   $S \to aSBC|abC,\ CB \to BC,\ bB \to bb,\ bC \to bc,\ cC \to cc$.
4. Describe the language generated by grammar
   $S \to FF,\ F \to aFb,\ F \to ab$.
5. Describe the language generated by grammar
   $S \to Sc,\ S \to A,\ A \to aAb,\ A \to \varepsilon$.
6. Describe the language generated by grammar
   $S \to \varepsilon,\ S \to a,\ S \to b,\ S \to aSa,\ S \to bSb$.
7. Describe the language generated by grammar
   $S \to SA,\ SAA \to ASb,\ ASA \to b,\ A \to a$.
8. Describe the language generated by grammar
   $S \to aSA,\ S \to abc,\ bA \to bbc,\ cA \to Aa$.
9. Describe the language generated by grammar
   $S \to aAS,\ S \to B,\ Aa \to aaA,\ AB \to B,\ B \to a$.
10. Find a linear grammar generating the next languageк $\{a^m b^n c:\ \tau \in \{a,b\}^*, m \geq 0, |\tau|_b = 2\}$.
11. Find a linear grammar generating the next languageк $\{a^n \tau:\ n \geq 1, m \geq 1$.
12. Find a linear grammar generating the next languageк $\{a,b\}^*$ - $a^n b^n c^n$: $n \geq 0$.

13. Find a linear grammar generating the next languageк $\{\alpha a\beta\xi b: \alpha\in\{a,b\}^*, \beta\in\{a,b\}^*$.

14. Find a linear grammar generating the next languageк $\{a,b,c\}^* - \{\tau c\tau: \tau\in\{a,b\}^*\}$.

15. Find Right Linear Grammar Equivalent to Grammar $S\rightarrow KbbaK$, $K\rightarrow Ka$, $K\rightarrow Kb$, $S\rightarrow\varepsilon$.

16. Find Right Linear Grammar Equivalent to Grammar $S\rightarrow aSb$, $S\rightarrow K$, $S\rightarrow J$, $K\rightarrow aK$, $J\rightarrow Jb$, $K\rightarrow a$, $J\rightarrow\varepsilon$.

Questions 6.1.

1. Are the following grammars equivalent

$S\rightarrow ab$, $S\rightarrow aKSb$, $K\rightarrow bSb$, $KS\rightarrow b$, $K\rightarrow\varepsilon$

and

$S\rightarrow aAb$, $A\rightarrow\varepsilon$, $A\rightarrow b$, $A\rightarrow S$, $A\rightarrow bSbS$?

2. Are the following grammars equivalent

$S\rightarrow aD$, $D\rightarrow bba$, $D\rightarrow baDa$, $D\rightarrow aDaDa$

and

$S\rightarrow aaE$, $S\rightarrow abD$, $E\rightarrow bDD$, $D\rightarrow aaEa$, $D\rightarrow abDa$, $D\rightarrow ba$?

3. What class does the grammar belong to?

$S\rightarrow abba$, $S\rightarrow baa$?

4. What class does the grammar belong to?

$S\rightarrow AD$, $A\rightarrow aA$, $A\rightarrow\varepsilon$, $D\rightarrow bDc$, $D\rightarrow\varepsilon$

5. Are the following grammars equivalent

$S\rightarrow AB$, $A\rightarrow a|Aa$, $A\rightarrow a|Aa$

and

$S\rightarrow AS|SB|AB$, $A\rightarrow a$, $B\rightarrow b$?

6. Are the following grammars equivalent

$S\rightarrow cE$, $E\rightarrow ddc$, $E\rightarrow dcEc$, $E\rightarrow cEcEc$

and

$S\rightarrow ccA$, $S\rightarrow cdB$, $A\rightarrow dBB$, $B\rightarrow ccAc$, $B\rightarrow cdBc$, $B\rightarrow dc$?

7. How can an unambiguous grammar describe a language that is generated by an ambiguous grammar $E\rightarrow E+E|E*E|(E)|i$.

Tests 6.1.

1. What will be the language$\{a^{2n-1}, n\geq 1\}$, if it is generated by the grammar

$\boldsymbol{N}=\{S\}$; $\boldsymbol{T}=\{a\}$, $\boldsymbol{P}=\{S\to a, S\to aaS\}$?

A) right-linear language.

B) left-linear language.

C) context-free language.

D) context-sensitive language.

E) recursive language.

2. What will be the language$\{a^{2n-1}\}$, if it is generated by the grammar $N=\{S\}$, $T=\{a\}$, $P=\{S\to a, S\to Saa\}$?

A) left-linear language.

B) right-linear language.

C) context-free language.

D) context-sensitive language.

E) recursive language.

3. What will be the grammar with the rules: $S\to aSa$, $S\to Q$, $Q\to bQ$, $Q\to\varepsilon$?

A) left linear

B) right linear

C) context-free

D) context-sensitive

E) recursive.

4. What will be the languageT^* in the alphabet $T=\{t_1, t_2,\ldots, t_n\}$, if it is generated by the grammar $S\to\varepsilon$, $S\to t_1S$, $S\to t_2S$, ..., $S\to t_nS$?

A) right-linear language.

B) left-linear language.

C) context-free language.

D) context-sensitive language.

E) recursive language.

5. What will be the grammar with the rules: $S\to QQ$, $Q\to cQQ$, $S\to a$?

A) context-free

B) right linear

C) left linear

D) context-sensitive

E) recursive.

6.2. Regular grammars

This section will consider regular grammars, their types and properties, and also offer examples, given tasks, formulated questions and tests. for the preparation of training materials, sources were used [1-9,11-13,28-32].

Let a formal grammar $G = <N, T, P, S>$ be given, where N is a finite set of nonterminals, T is a finite set of terminals, $T \cap N = \varnothing$, P is a finite set of inference rules, S is an initial nonterminal, $S \in N$. Then the following definitions can be given:

Definitions 6.2.1.

1. If in the grammar G for each $A, B \in N, \tau \in T^*$ all inference rules are given in the form $A \to \tau B$ or $A \to \tau$, then it is called a *right-linear grammar*.

2. The set of strings generated by a right-linear grammar is *right-linear language.*

3. A right-linear grammar is found $G = <N, T, P, S>$ in normal form, If each rule in it has the form $A \to \varepsilon$, $A \to a$ or $A \to aB$, where $A \in N$, $B \in N$, $a \in T$.

4. If in the grammar G for each $A, B \in N, \tau \in T^*$ all inference rules are given in the form $A \to B\tau$ or $A \to \tau$, then it is called a *left-linear grammar.*

5. The set of strings generated by the left-linear grammar is *left-linear language.*

6. A left-linear grammar is found $G = <N, T, P, S>$ in normal form, If each rule in it has the form $A \to \varepsilon$, $A \to a$ or $A \to Ba$, where $A \in \boldsymbol{N}$, $B \in \boldsymbol{N}$, $a \in \boldsymbol{T}$.

7. Linear grammar is in normal form, If in linear grammar each rule has the form $A \to \varepsilon$, $A \to \alpha$, $A \to \alpha B$ or $A \to B\alpha$, where $A \in \boldsymbol{N}$, $B \in \boldsymbol{N}$, $\alpha \in \boldsymbol{T}^*$.

8. If in the grammar G for each $A \in \boldsymbol{N}$, $B \in \boldsymbol{N}$, $\alpha \in \boldsymbol{T}^*$, $\beta \in \boldsymbol{T}$ all inference rules are given in the form $A \to \alpha B \beta$ or $A \to \alpha$, then it is called a *linear grammar*.

9. The set of strings generated by a linear grammar is a *linear language.*

10. In a linear grammar in the chain during the inference process there will be no extra nonterminal and If $\beta = \varepsilon$, then it will be right-linear, and If $\alpha = \varepsilon$, then it will be left linear

11. A right-linear grammar $G = <N, T, P, S>$ is called regular, If the initial nonterminal S does not occur in the right-hand side of any rule, that is, each of its rules, except $S \to \varepsilon \in \boldsymbol{P}$, has the form either $A \to aB$, or $A \to a$, where $A, B \in N$, $a \in T$.

12. The set of strings generated by a regular grammar is a regular language; it is equivalent to the regular set defined in I.3.3.

It is now possible to establish the properties of these grammars and languages using the following theorems:

Theorem 7.2.1. Every right-linear grammar is equivalent to some right-linear grammar in normal form.

Theorem 7.2.2. If a right-linear language does not contain an empty word, then it is generated by some right-linear grammar in normal form without ε-rules.

Theorem 7.2.3. Every right-linear grammar is equivalent to some regular grammar.

Theorem 7.2.4. Each linear grammar is equivalent to some linear grammar in normal form.

Theorem 7.2.5. If a linear language does not contain an empty word, then it is generated by some linear grammar in normal form without ε-rules.

Theorem 7.2.6. The language L is linear if and only if the language $L \setminus \{\varepsilon\}$ is linear.

Theorem 7.2.7. Let L be a linear language over the alphabet T. Then there is a positive integer k such that for any chain $\lambda \in \boldsymbol{L}$ of length at least k one can choose chains α, β, γ, δ, τ∈T * for which $\alpha\beta\gamma\delta\tau = \omega$, $\beta\delta \neq \varepsilon$ (that is, β ≠ ε or $\delta \neq \varepsilon$), $|\alpha\beta|+|\delta\tau| \leq k$ and $\alpha\beta^i\gamma^i\delta\tau \in \boldsymbol{L}$ for all $i \in N$.

In this grammar, a right-linear language is generated

using inference rules

Examples 6.2.1.

1. The grammar is set $\boldsymbol{G} = <\boldsymbol{N}, \boldsymbol{T}, \boldsymbol{P}, S>$, где $\boldsymbol{N} = \{S, A\}$, $\boldsymbol{T} = \{a, b\}$, $\boldsymbol{P} = \{S \to aA, A \to aA, A \to b\}$. In this grammar, a left-linear language is generated $\boldsymbol{L}(\boldsymbol{G}) \rightleftharpoons \{a^n b : n=1,2,...\}$ using inference rules $S \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow ... \Rightarrow a...aaab$.

2. The grammar is set $\boldsymbol{G} = <\boldsymbol{N}, \boldsymbol{T}, \boldsymbol{P}, S>$, где $\boldsymbol{N} = \{S, A\}$, $\boldsymbol{T} = \{a, b\}$, $\boldsymbol{P} = \{S \to Aa, A \to Aa, A \to b\}$. In this grammar, a right-linear language is generated к $\boldsymbol{L}(\boldsymbol{G}) \rightleftharpoons \{ba^n : n=1,2,...\}$ using inference rules $S \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow ... \Rightarrow baaa...a$.

3. Consider a right-linear (left-linear) language $\{a^{2n-1}\}$, consisting of chains of the form *a*, *aaa*, *aaaaa*, ... It is generated by a right-linear (left-linear) grammar G = <N, T, P, S>, consisting of the set $\boldsymbol{T}=\{a\}$, $\boldsymbol{N}=\{S\}$ and $\boldsymbol{P}=\{S\to a, S\to aaS\}$ ($\boldsymbol{P}=\{S\to a, S\to Saa\}$). By the look of the rules, you can see that the specified language will be a right-linear (left-linear) language.

4. Consider the language $\boldsymbol{L}=\{a^m b^m a^n b^n: m\geq 0, n\geq 0\}$ over the alphabet $\{a,b\}$. The assertion of Theorem 7.4.7 does not hold for any natural number *k*. Consequently, the language *L* is not linear.

5. The language $\{\omega\in\{a,b\}^*: |\omega|_a=2, |\omega|_b=2\}$ is generated by a linear grammar. The given language does not contain an empty string, in any string the number of occurrences and *a* and *b* must be 2. Therefore, it is generated by a right-linear grammar

Tasks 6.2.1.

1. Find Right Linear Grammar becoming a *L*anguage $\{\tau\in\{a,b\}^*: |\tau|_a \geq 2, |\tau|_b\geq 2\}$.

2. Find Right Linear Grammar equivalent grammar $S\to E$, $S\to bE$, $S\to caE$, $E\to a$, $E\to bS$.

3. Find Right Linear Grammar in normal form without ε-rules, generating the language $\{a^k b^m c^n: k\geq 0, m\geq 1, n\geq 0\}$.

4. Find Right Linear Grammar in normal form without ε-rules, generating the language $\{a,b\}^* - (\{a^n: n\geq 0\} \square \{\{a^k b^m c^n: k\geq 0, m\geq 1, n\geq 0\}\ b^n: n\geq 0\}$.

5. Find a linear grammar in normal form without ε-rules that generates a language $\{a^n b^n c^m: n\geq 1, m\geq 1\}$.

6. Describe the language affected by the following rules:

$S\to 0A|1S|\varepsilon$, $A\to 0B|1A$, $B\to 0S|1B$

7. Describe the language affected by grammar:

$\boldsymbol{T} = \{a, b, d\}$, $\boldsymbol{N} = \{A,B, D\}$, $S = A$,

$\boldsymbol{P}=\{A\to aB, B\to aB, B\to b, B\to bD, D\to d, D\to dD, A\to aD, A\to a\}$

Questions 6.2.1.

1. Are there languages $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ such that $\boldsymbol{L_1}$ is right-linear and $\boldsymbol{L_2}$ is left-linear, and $\boldsymbol{L_1}\cup\boldsymbol{L_2}$ is not linear language?

2. Are there languages $\boldsymbol{L_1}$ and $\boldsymbol{L_2}$ such that $\boldsymbol{L_1}$ is right-linear and $\boldsymbol{L_2}$ is left-linear, and $\boldsymbol{L_1} \cap \boldsymbol{L_2}$ is linear language?

3. Is there a right-linear grammar G such that the language $\boldsymbol{L}(\boldsymbol{G})^R$ is not generated by any right-linear grammar that has as many rules as the grammar G?

4. Is there a right-linear grammar G such that the language $\boldsymbol{L}(\boldsymbol{G})^R$ is not generated by any right-linear grammar with $n + 1$ rules (where n is the number of rules in G)?

5. Is there a right-linear grammar G with three nonterminals such that the language $\boldsymbol{L}(\boldsymbol{G})^R$ is not generated by any right-linear grammar with three nonterminals?

6. What type of grammar are the following rules?

$S \to 0A|1S|\varepsilon,\ A \to 0B|1A,\ B \to 0S|1B$

7. Do the following rules apply to right-linear grammar?

$S \to AB,\ A \to Aa|bB,\ B \to a|Sb$

Tests 6.2.1.

1. What grammar is a regular grammar?

A) right-linear;

B) left-linear;

C) context-free;

D) recursive;

E) context sensitive.

2. What will be the language $\{ a^n b, n \geq 1\}$ generated by a grammar of the form $\boldsymbol{N} = \{S, A\}$, $\boldsymbol{T} = \{a, b\}$, $\boldsymbol{P} = \{S \to aA, A \to aA, A \to b\}$?

A) right-linear language;

B) left-linear language;

C) context-free language;

D) context-sensitive language;

E) recursive language.

3. What will be the language generated by the grammar of the form:

$G = <\{S\}, \{a, b\}, S, P>$ and $S \to abS\ a$?

A) regular language;

B) left-linear language;

C) context-free language;

D) context-sensitive language;

E) recursive language.

4. What grammar is $G = <\{S, A, B\}, \{a, b\}, S, P>$, If has the following rules: $S \rightarrow A$, $A \rightarrow aB\varepsilon$, $B \rightarrow Ab$?

A) linear;

B) regular;

C) context-free;

D) context sensitive;

E) recursive.

5. What will be the language T * in the alphabet $T = \{t_1, t_2, \ldots, t_n\}$, If it is generated by the grammar $S \rightarrow \varepsilon$, $S \rightarrow t_1S$, $S \rightarrow t_2S$, ..., $S \rightarrow t_nS$?

A) right-linear language;

B) left-linear language;

C) context-free language;

D) context-sensitive language;

E) recursive language.

6.3. Context-free grammars

This part deals with context-free grammars, discusses the algorithmic problems of context-free languages, provides Examples, gives Tasks, formulates Questions and Tests. For the preparation of teaching materials, sources were used [1-9,11-13,15-22,24,25-32].

Recall that context-free grammars are grammars in which all inference rules are of the form $A \to \alpha$, where $A \in \boldsymbol{N}$, $\alpha \in (\boldsymbol{T} \cup \boldsymbol{N})^*$, i.e. nonterminal A is replaced by the string α in the set of terminals and nonterminals regardless of the context in which A.

Context-free grammars (CFGs) occupy an important place in language theory and serve to Tasks the syntactic structure of the generated string through the sequence of application of inference rules.

Definition 6.3.1. The languages generated by context-free grammars are called context-free languages.

Examples 6.3.1. Let be $\boldsymbol{G_1} = <\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, S>$, where $\boldsymbol{N}=\{S,A,B\}$, $\boldsymbol{T}=\{a,b\}$, $\boldsymbol{P} = \{S \to aB, S \to bA, A \to aS, A \to bAA, B \to bS, A \to a, B \to aBB, B \to b\}$.

The $\boldsymbol{G_1}$ grammar is context-free, since in each of its inference rules the left side consists of a single nonterminal, and the right side consists of a non-empty chain of terminals and nonterminals.

In a $\boldsymbol{G_1}$ grammar, the typical conclusions are:

$S \Rightarrow aB \Rightarrow ab$, $S \Rightarrow aB \Rightarrow abS \Rightarrow abbA \Rightarrow abba$, $S \Rightarrow bA \Rightarrow ba$, $S \Rightarrow bA \Rightarrow bbAA \Rightarrow bbaA \Rightarrow bbaa$.

The applied inference rules that generate the set of all strings consisting of an equal number of characters *a* and *b*.

A grammar can have several equivalent inferences, which apply the same rules in the same places, but in a different order. It is difficult to define the notion of equivalence of two inferences for grammars of an arbitrary form, but in the CFG conditions one can introduce a convenient graphical representation of a class of equivalent inferences, called an inference tree.

Definition 6.3.2. The marked ordered tree D is called an inference (parsing) tree in CFG $G(S) = <T, N, P, S>$, If the following conditions are satisfied:

(1) The root of the tree *D* (a vertex that does not include any arcs) is marked with *S*;

(2) If $\boldsymbol{D_1}$, …, $\boldsymbol{D_k}$ are subtrees dominated by direct descendants of the root of the tree, and the root of the tree $\boldsymbol{D_i}$ is marked with X_i, then the expression $S \rightarrow X_1 X_2 \ldots X_k$ is a rule from the set P.

(3) If X_i is a nonterminal, $\boldsymbol{D_i}$ consists of a single vertex labeled X_i and If X_i is a terminal, then $\boldsymbol{D_i}$ for any i = 1, 2, ..., k must be an inference tree in the grammar $\boldsymbol{G}(X_i) = <\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, X_i>$

(4) If the root of the tree has a single descendant labeled ε, then this descendant forms a tree consisting of a single vertex, and the expression $S \rightarrow \varepsilon$ will be a rule from the set P.

Thus, in the inference tree, each vertex is marked with a symbol from the set $\boldsymbol{N} \cup \boldsymbol{T} \cup \{\varepsilon\}$. In this case, If the internal vertex of the tree is denoted by the symbol A, and its direct descendants are denoted by the symbols $X_1, X_2, \ldots, X_n$, then the expression $A \rightarrow X_1 X_2 \ldots X_n$ will be the rule of the grammar inference.

The inference tree for CFG $\boldsymbol{G} = <\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, S>$ can be constructed as:

1. The vertices of the tree are marked with symbols from the set $T \cup N$ in a strictly defined order.

2. If a vertex labeled X has at least one subordinate vertex, then $X \in N$. In this case, the root of the tree is marked with $S \in N$.

3. If the vertices $X_1, X_2, \ldots, X_k$ are directly subordinate to the vertex S, then the rule $S \rightarrow X_1, X_2, \ldots, X_k$ must belong to the set P.

Note that there is a natural ordering of the vertices of an ordered tree, in which the direct descendants of the vertex are ordered “from left to right”.

Examples 6.3.2. Figure 7.3.1 shows inference trees in the grammar $\boldsymbol{G_2} = \boldsymbol{G}(S)$ with the rules $S \rightarrow aSbS|bSaS|\varepsilon$.

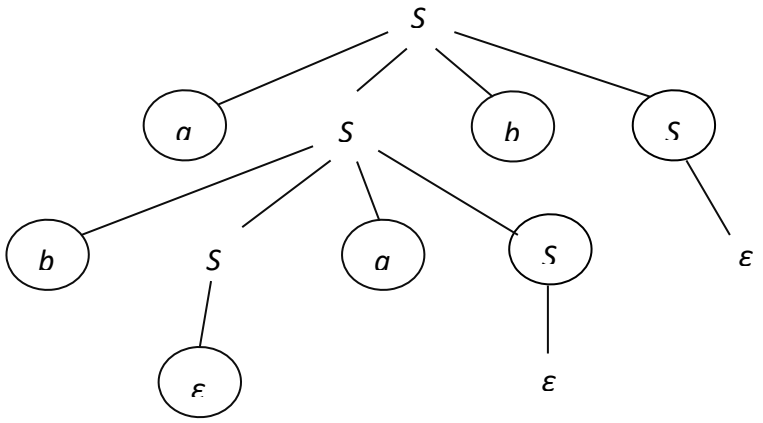


Figure 7.3.1. Examples of grammar output tree.

Let's number the vertices of the output tree from top to bottom and from left to right. Suppose that X is a vertex and X_1, ..., X_k are its direct descendants.Then for the vertices X_i and X_j, If $i<j$, ($i=1,2, ..., k$, $j=1,2, ..., k$), then the vertex X_i and all its descendants are considered to be located to the left of the vertex X_j and all her descendants.

Let D be an inference tree in CFG G = <T, N, P, S>. Then the following new concepts can be introduced:

Definitions 6.3.3:

1. The crown of the inference tree is the chain, which is obtained if we write out the labels of the leaves from left to right;

2. A section of a tree D is a set C of vertices of a tree D such that:

(1) no two vertices from C lie on the same path to D;

(2) no vertex of the tree D can be added to C without violating property (1).

It can be shown that inference trees represent inferences in the sense that for each inference of the derivable chain α in CFG G one can construct an inference tree in G with crown α, and vice versa.

Definition 6.3.4. The crown of a section of a tree D is a chain that is obtained by concatenating from left to right the labels of the vertices that form a certain section.

Examples 6.3.3. The crown of the inference tree section shown in Figure 7.3.2 is the chain *abSaSbS*.

Let $\boldsymbol{G}$ = <$\boldsymbol{T}$, $\boldsymbol{N}$, $\boldsymbol{P}$, S> - CFG. Then $S \Rightarrow^* \alpha$ holds when G contains an inference tree D with crown α. Let $C_0, C_1, C_2, \ldots, C_n$ be a sequence of sections of the tree D such that:

(1) Section C_0 contains only the root of the tree D;

(2) The section C_{i+1} for $0 \le i < n$ is obtained from the section C_i by replacing one nonterminal vertex with its direct descendants;

(3) C_n - crown of tree D. If $S \Rightarrow^* \tau = \alpha_0, \alpha_1, \ldots, \alpha_n$ is the left inference of the terminal chain τ, then each αi has the form $x_i A_i \beta_i$, where $x_i \in \boldsymbol{T}^*$, $A_i \in \boldsymbol{N}$ and $\beta_i \in (\boldsymbol{N} \cup \boldsymbol{T})^*$, $0 \le i < n$. In the left inference, each subsequent chain of the inference α_{i+1} is obtained by replacing the leftmost nonterminal A_i of the previous chain αi

with the right-hand side of some rule. In the right output, the rightmost nonterminal is replaced.

Definitions 6.3.5:

1. If the section C_{i+1} is obtained from C_i by replacing the leftmost nonterminal vertex in C_i with its direct descendants, then the corresponding conclusion $\alpha_0,\alpha_1,\ldots,\alpha_n$ is called the left inference of the chain α_n from α_0 in the grammar G. The right inference is defined similarly, it is only necessary in in the previous sentence, read "rightmost" instead of "leftmost". Note that the left (or right) output is uniquely determined by the inference tree.

2. The string τ is called left derivable in the grammar G, If there is a left derivation $S\Rightarrow^*\tau$, and is written as $S\Rightarrow^*_{G_l}\tau$ (or $S\Rightarrow^*_l\tau$, when it is clear which grammar G is meant).

3. The string τ is called deducible in the grammar G, If there is a right deduction $S\Rightarrow^*\tau$, and it is written $S\Rightarrow^*_{G_h}\tau$ (or $S\Rightarrow^*_{G_h}\tau$). Thus, one step of the left inference is denoted by $\Rightarrow_l$,, and the step of the right inference is denoted by $\Rightarrow_h$.

If $S\Rightarrow^*\tau = \alpha_0,\alpha_1,\ldots,\alpha_n$ is the left inference of the terminal chain τ, then each α_i has the form $x_iA_i\beta_i$, where $x_i\in\boldsymbol{T}^*$, $A_i\in\boldsymbol{N}$ and $\beta_i\in(\boldsymbol{N}\cup\boldsymbol{T})^*$, $0\le i<n$. In the left inference, each subsequent chain of the inference α_{i+1} is obtained by replacing the leftmost nonterminal A_i of the previous chain α_i with the right-hand side of some rule. In the right output, the rightmost nonterminal is replaced.

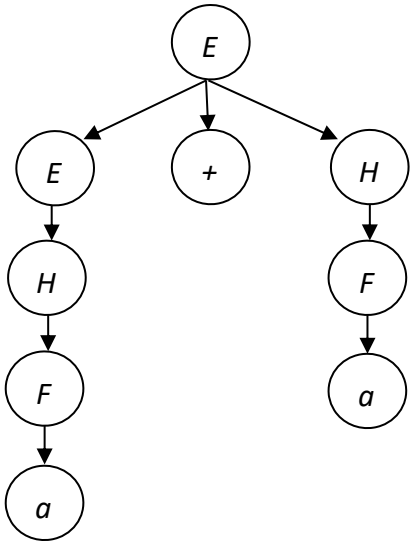
Examples 6.3.4. Consider CFG $\boldsymbol{G}_a$ with rules

$E\to E+H\,|H$, $H\to H*F\,|F$, $F\to(E)\,|a$

The inference tree shown in Figure 7.3.3 serves as a representation of the two equivalent pins of $a + a$ chain:

1) left output $E\Rightarrow E+H\Rightarrow H+H\Rightarrow F+H\Rightarrow a+H\Rightarrow a+F\Rightarrow a+a$,

2) right output $E\Rightarrow E+H\Rightarrow E+F\Rightarrow E+a\Rightarrow H+a\Rightarrow F+a\Rightarrow a+a$.



Definition 6.3.6. CFG G is called ambiguous, If there is at least one terminal chain $\tau \in \boldsymbol{L}(\boldsymbol{G})$, which is the crown of two or more different derivation trees in G. That is, some terminal chain τ $\tau \in \boldsymbol{L}(\boldsymbol{G})$ has two or more different left (right) inference, otherwise CFG G is called unambiguous.

Examples 6.3.5. CFG $\boldsymbol{G}_a$ from Example 7.3.5 is ambiguous, since there is a terminal chain $a + a$ has two or more different left (right) outputs:

left terminal $E \Rightarrow E+H \Rightarrow H+H \Rightarrow F+H \Rightarrow a+H \Rightarrow a+F \Rightarrow a+a$,

right output $E \Rightarrow E+H \Rightarrow E+F \Rightarrow E+a \Rightarrow H+a \Rightarrow F+a \Rightarrow a+a$.

Definition 6.3.7. Let $\gamma \in \boldsymbol{T}^*$, $\delta \in \boldsymbol{T}^*$, $\omega \in \boldsymbol{T}^*$ and $X \in \boldsymbol{N}$. Then the nonterminal symbol X is called useful in CFG $\boldsymbol{G} = <\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, S>$, If it can participate in the derivation of the form $S \Rightarrow^* \gamma X \delta \Rightarrow^* \omega$, otherwise it is called useless

Definition 6.3.8. The symbol $X \in N \cup T$ is called unreachable in CFG $G = <T, N, P, S>$, If X does not appear in any water derivable chain.

Definition 6.3.9. We call CFG $G = <T, N, P, S>$ a grammar without ε-rules (or non-shortening), If either

(1) P does not contain ε-rules, or

(2) there is exactly one ε-rule $S \rightarrow \varepsilon$ and S does not occur in the right-hand sides of the remaining rules from P.

Definitions 6.3.10. Let CFG $G = <T, N, P, S>$ be given. Then:

1. A nonterminal $A \in N$ in G is called a cyclic symbol, If for it there is a derivation $A \Rightarrow \zeta A \xi$, $\zeta \in (\boldsymbol{T} \cup \boldsymbol{N})^*$, $\xi \in (\boldsymbol{T} \cup \boldsymbol{N})^*$.

2. A cyclic symbol is called effective, If $A \Rightarrow \alpha A \beta$, where $|\alpha A \beta| > 1$, otherwise the cyclic symbol is called fictitious.

3. A grammar G is called a cyclic grammar if it contains at least one cyclic symbol.

4. The grammar G is called a grammar without cycles, If for the nonterminal $A \in N$ there are no conclusions of the form $A \Rightarrow^+ A$;

5. A grammar G is called a reduced grammar, If it is without loops, without ε-rules, and without useless symbols.

Grammars with ε-rules or loops are sometimes more difficult to parse than grammars without ε-rules, since in any practical situation useless symbols unnecessarily increase the size of the parser. Therefore, for some parsing algorithms, we will require that the grammars appearing in them be reduced. Let us prove that this requirement nevertheless allows us to consider all CF-languages without cycles.

Definition 6.3.11. The A-rule of a KS-grammar is a rule of the form A → α (do not confuse the A-rule with the ε-rule, which has the form B → ε).

Examples 6.3.6. Eliminate the rule $A \rightarrow aAA$ from the grammar G having two rules $A \rightarrow aAA \mid b$. Applying the lemma, setting $\alpha = a$ and $\beta = A$, we obtain a grammar G' with the rules $A \rightarrow aAAA|abA|b$.

Tasks 6.3.

1. Construct the reduced grammar equivalent to a grammar with the following rules:

$S \rightarrow aABS|bCACd,$
$A \rightarrow bAB|cSA|cCC,$
$B \rightarrow bAB|cSB,$
$C \rightarrow cS\ |c.$

2. Construct the reduced grammar equivalent to a grammar with the following rules:

$S \rightarrow aAB|E,\ A \rightarrow dDA|\varepsilon,$
$B \rightarrow bE|f,$
$C \rightarrow cAB|dSD|a,$
$D \rightarrow \varepsilon A,$
$E \rightarrow fA|g.$

3. Build the given grammar that generates identifiers consisting of letters and numbers.

4. Build an inference tree for chain 10.1001 in CFG with the rules*:* $S \rightarrow S0 \mid S1 \mid D0 \mid D1,\ D \rightarrow H,\ H \rightarrow 0 \mid 1 \mid H0 \mid H1.$

5. Build an inference tree for the chain *if a then b = a + b + b in* CFG with the rules: $S \rightarrow$ *if* B then $S|B = E,\ E \rightarrow B|B+E,\ B \rightarrow a|b$

6. Build *a CFG* generating language $\{a^{2n}b^m c^{2k} | m=n+k,\ m>1\}$, build an output tree and left-sided output for *aabbbcccc.*

7. Build the CFG generating the language $\boldsymbol{L}= \{1^{3n+2}\ 0^n : n\geq 0\}$, build the inference tree and left-sided inference for 1111111100.

8. Build in the alphabet $T = \{a, b\}$ languages $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, in which the letter b is repeated n times $\boldsymbol{L}_1 = \{ab^n : n>= 0\}$, $\boldsymbol{L}_2 = \{b^n\ a : n >= 1\}$.

9. Construct a *CFG* that generates a language that consists of strings that begin with # and end with!, Between which there is a non-empty string of + and - signs that does not contain two identical symbols standing side by side.

10. Construct a *CFG* that generates correct logical expressions using the conjunction & and the disjunction ∨, which can be connected by the relations: >, <, =.

Questions 6.3:

1. How is the left (right output) determined in CFG?
2. What type of language $\boldsymbol{L}(\boldsymbol{G})\rightleftharpoons\{a^n b^n c^n :\ n>1\}$?
3. What type of language $\boldsymbol{L}= \{1^{3n+2}\ 0^n :\ n\geq 0\}$?
4. What is the type of grammar that generates a lot $\{a \vee a^* a\}$?
5. What is the type of grammar that generates a lot $\{a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 :\ a_i \in \{0,1\},\ 1\leq i \leq n\}$?
6. What will be the left pins and right pins in CFG for this chain 1111111100?
7. Does a non-cyclical CFG generate a final language?
8. Does a cyclic reduced CFG containing at least one effective cyclic symbol generate an infinite language?
9. Is it possible to transform the rules

$$A\rightarrow AA|\alpha,\ A\rightarrow A\alpha A|\beta,\ A\rightarrow \alpha A|A\beta|\gamma$$

so as to get ambiguous grammar?

10. Is it possible to construct a cyclic reduced grammar to generate a language $\boldsymbol{L}= \{a^{2n}\ b^m\ c^{2k} :\ m=n+k,\ m>1\}$?

11. Is the language context-free $\{\omega\omega\omega :\ \omega\in\{a, b\}^*\}$?
12. Is the language context-free

$\{\gamma\delta\delta\omega :\ \gamma\in\{a, b\}^+,\ \delta\in\{a, b\}^+,\ \omega\in\{a, b\}^+\}$?

Tests 6.3.

1. What language is generated by a grammar in which all inference rules are of the form $A \to \alpha$, where $A \in \boldsymbol{N}$, $\alpha \in (\boldsymbol{T} \cup \boldsymbol{N})$?

A) context-free;

B) left-hand;

C) right-linear;

D) recursive;

E) context sensitive.

2. What symbol is a nonterminal called, If for it there is a derivation $A \Rightarrow \zeta A \xi$, $\zeta \in (\boldsymbol{T} \cup \boldsymbol{N})^*$, $\xi \in (\boldsymbol{T} \cup \boldsymbol{N})^*$?

A) cyclic;

B) acyclic;

C) periodic;

D) unique;

E) ambiguous.

3. What is the name of the chain, If there is a conclusion $S \Rightarrow^* \tau$, and is written $S \Rightarrow^*_{Gh} \tau$ (or $S \Rightarrow^*_{Gh} \tau$)?

A) legally withdrawable;

B) left-handed;

C) directly withdrawable;

D) re-removable;

E) not deducible.

4. What grammar is G, if there is at least one terminal chain $\tau \in L(G)$, which is the crown of 2 or more different inference trees in G?

A) ambiguous;

B) unambiguous;

C) undefined;

D) cyclic;

E) What is the name of the character $\boldsymbol{X \in N \cup T}$ if it appears in the string?

A) unattainable;

B) achievable;

C) ambiguous;

D) unambiguous;

E) recursive.

7. FINITE AUTOMATONS

Lecture objective: explain the concept of universal automaton and finite automaton.

Lecture plan: study the composition and structure of abstract automaton; give a formal definition of indeterminate and determinate finite automaton and of languages recognized by such automatons.

Usually under the term “automaton” we understand a device which, once turned on, can perform a number of given operations on its own. However, we deal with an abstract automaton used as a mathematical model of any digital (discrete) devices in which all signals are quantized in level, and all actions are quantized in time.

An abstract automaton (hereinafter – automaton) can distinguish a set or transform a set into another set; it consists of a tape, a head unit and a controller device; it may also have working memory.

Tape – a linear sequence of cells, each of which can store only one symbol from a certain finite input (output) alphabet.

The tape is infinite, but at each given moment only a finite number of cells is occupied. Special markers denoting the beginning and end of the tape may occupy the boundary regions to the left and right of the occupied cell area. The marker may be just at one end of the tape or be absent altogether.

Input (output) head unit – a device which can view only one tape cell at any given moment of time. The head unit can shift one cell to the left or to the right, or remain immobile. It is generally assumed that the head unit is read-only, i.e. during the work of the automaton the symbols on the tape do not change. But it is also possible to consider automatons whose head unit both reads and writes. Thus, the head unit may perform both reading and writing operations.

Working memory – an auxiliary storage for reading and writing data. Working memory may be organized as a dynamic data structure (queue or stack).

Controlling unit – a device which governs the automaton’s behavior and has a finite internal memory for storing a finite number of states. It governs the automaton’s behavior by means of a function (relation) which describes how the states change depending on the current state and current input symbol read by the head unit, and the current information extracted

from the working memory if available. The controlling unit also determines the direction of the shift of the head unit and the information to be entered in the working memory.

The automaton is determined by the input of a finite set of states of the controlling unit, finite set of accepted input symbols, the source state and the set of final states, as well as the state transition function which, by the current state and current input symbol being its arguments, indicates all possible next states or values of this function. The work of the automaton may be conveniently described by means of its configuration. The automaton's configuration includes:

- controlling unit's state;
- contents of the input tape and the position of the input head unit;
- contents of the working memory and the position of the working head unit if available;
- contents of the output tape if available.

The automaton's configuration can be initial, current and final.

In its *initial configuration* the internal memory contains a previously entered symbol denoting the initial state of the controlling unit; the controlling unit is in the initial state; the head unit reads the leftmost input symbol on the tape; if working memory is available, it contains preconfigured initial contents.

In its *current configuration* the internal memory contains previously entered symbols of current states of the controlling unit; the controlling unit is in one of its current states; the head unit reads neither the leftmost nor the rightmost current input symbol; if working memory is available it has preconfigured current contents.

In its *final configuration* the internal memory contains previously entered symbols denoting the final states of the controlling unit; the controlling unit is in one of its final states; the head unit views the right end marker or, if the marker is not available, it leaves the input tape; if working memory is available then it satisfies certain conditions.

Prior to its inception the automaton is its initial configuration, i.e. the symbol denoting the initial state of the controlling unit is entered in the internal memory, the input chain is entered in the input tape; if working memory is available, corresponding data is entered in the memory.

The automaton uses a program consisting of a finite sequence of *steps*. Each step consists of the current (initial) and next (final) configuration.

At the step's beginning the memory reads the symbol of the current state of the controlling unit, the input tape reads the current input symbol; the information in the working memory, if available, is also read. Then, depending on the current state and read information the automaton's actions are determined:

(1) Input head unit moves to the right, left or remains in place;

(2) A new symbol is entered in the current cell of the input tape or the previous symbol is not changed;

(3) Some information, if available, is entered in the working memory;

(4) A symbol is entered in the output tape, if the tape is available.

(5) The controlling unit moves into another state and the number (symbol) of this state is entered in the internal memory.

As a result, during one step of the automaton the input head unit can move one cell to the left, right or remain in its place. As the automaton functions, the contents of the input tape cells do not change, but the contents of the output tape cells and the working tape cells can.

If the automaton views the input chain and executes a sequence of steps starting from the initial configuration and finishing in a final configuration, then it recognizes the chain.

A *language* recognized by the automaton is a set of chains that the automaton recognizes.

Examples 7.1:

1. A public pay telephone may serve as an example of automaton: it recognizes the input of a coin and enters the dial number state.

2. An ATM is an automaton: it recognizes an inserted card and enters the pin-code input state.

3. A subway ticket gate is an automaton: it recognizes a token and enters the open gate state.

Finite automatons recognize regular languages. First, formal definitions of indeterminate and determinate finite automatons are given,

then the languages they recognize are described, followed by the proof of their equivalency.

Finite automatons are among the simplest and most widespread recognizing machines. A finite automaton contains *output tape, internal memory, external memory, head unit and controlling unit.*

Finite automaton may be indeterminate or determinate, but its head unit must be one-way only and move only to the right. Their formal definitions are as follows:

Definition 7.1. *Indeterminate finite automaton (IFA)* is determined by the seven element set $M = <Q,T,I,F,\vdash,\dashv,\Delta>$ where:

Q – finite set of states of the controlling unit;

T– finite set of input symbols, $Q \cap T = \varnothing$;

I – set of initial states of the controlling unit, $I \subseteq Q$;

F– set of final states of the controlling unit indicating that the input chain is recognized, $F \subseteq Q$;

$\vdash,\dashv$– tape start and end markers $\vdash,\dashv \notin T$;

Δ–set of relations of transition $\Delta \subseteq Q \times T^* \times \vartheta(Q)$, $\vartheta(Q)$ – set of all subsets of the set Q.

The determined finite automaton (DFA) is a special case of IFA.

Definition 7.2. Finite automaton $M = <Q,T,I,F,\vdash,\dashv,\Delta>$ is called *determined,* if:

(1) The set of initial states I contains exactly one element;

(2) For each transition $<q, \tau, p> \in \Delta$ $|\tau|=1$ holds true;

(3) For each state $q \in Q$ and for each symbol $t \in T$ there exists no more than one state $p \in Q$ with an attribute $<q, t, p> \in \Delta$;

(4) Other symbols are identical to IFA.

Notes 7.1:

1. Sometimes instead of the set of relations of transition Δ taking logical values “true” or “false”, the function of transition δ is used which takes value as a symbol of the set Q, where δ: $Q \times T^* \rightarrow \vartheta(Q)$ – in the case of IFA and δ: $Q \times T^* \rightarrow Q$– in the case of DFA. From the function δ it is easy to arrive at the relation Δ by assuming

$\Delta = \{<q, \tau, \delta(q, \tau)>: \ q \in Q, \tau \in T^*\}$

2. Henceforth we shall use both relations of transition and functions of transition depending on the context without making particular mention. For any $q \in \boldsymbol{Q}, p \in \boldsymbol{Q}$ и $\tau \in \boldsymbol{T}^*$ we may use:

1)For relations of transition: $<q,\tau,\{p\}>$–for IFA, $<q,\tau,p>$–for DFA;

2)For function of transition: $\delta(q,\tau)=\{p\}$–for IFA,$\delta(q,\tau)=p$ – for DFA.

3. If we want to use the function of transition instead of the relation of transition, then in the formal definition KA it is necessary to substitute the symbol Δ with δ, and leave other symbols unchanged at their previous values, i.e. we obtain $\boldsymbol{M} = <\boldsymbol{Q}, \boldsymbol{T}, \boldsymbol{I}, \boldsymbol{F}, \vdash, \dashv, \delta>$.

The KA transition may be illustrated as a *diagram*, in which each state is denoted with a circle and transition with an arrow. An arrow from the state $q \in \boldsymbol{Q}$ to the state $p \in \boldsymbol{Q}$ denoted with a chain $\tau \in \boldsymbol{T}^*$ indicates that $<q, \tau, p>$ (or $\delta(q, \tau) = p$) is a transition within the given IFA. Each initial state may be recognized by a short arrow leading to it. Each final state is indicated with a double circle.

8. Are the following grammars equivalent?

$S \to ab, S \to aKSb, K \to bSb, KS \to b, K \to \varepsilon$

and

$S \to aAb, A \to \varepsilon, A \to b, A \to S, A \to bSbS$

9. Are the following grammars equivalent?

$S \to aD, D \to bba, D \to baDa, D \to aDaDa$

and

$S \to aaE, S \to abD, E \to bDD, D \to aaEa, D \to abDa, D \to ba$?

10. What class does the following grammar belong to?

$S \to abba, S \to baa$?

11. What class does the following grammar belong to?

$S \to AD, A \to aA, A \to \varepsilon, D \to bDc, D \to \varepsilon$

12. Is the grammar with the rules

$S \to AB, A \to a|Aa, A \to a|Aa$

equivalent to the grammar with the rules

$S \to AS|SB|AB, A \to a, B \to b$?

13. Is the grammar with the rules

$S \to cE, E \to ddc, E \to dcEc, E \to cEcEc$

equivalent to the grammar with the rules

$S \rightarrow ccA$, $S \rightarrow cdB$,$A \rightarrow dBB$,$B \rightarrow ccAc$,$B \rightarrow cdBc$,$B \rightarrow dc$?

How should one describe in unambiguous grammar a language generated by the ambiguous grammar $E \rightarrow E+E|E*E|(E)|i$?

Examples 7.2:

1. For KA $\boldsymbol{M}_1$ with one transition and parameters: $\boldsymbol{Q}=\{q,p\}$; $\boldsymbol{T}^*=\{\tau\}$, $\boldsymbol{I}=\{q\}$, $\boldsymbol{F}=\{p\}$, $\delta(q,\tau)=p$ the diagram is shown in the figure 7.1.

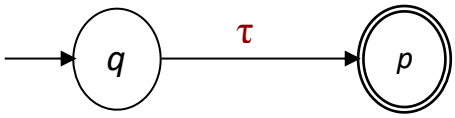


Figure 8.1. Diagram KA $\boldsymbol{M}_1$ with one transition.

2. Let KA $\boldsymbol{M}_2$ have the following parameters: $\boldsymbol{Q}=\{1,2\}$, $\mathbf{T}=\{a,b\}$, $\boldsymbol{I}=\{1\}$, $\boldsymbol{F}=\{2\}$, $\Delta=\{<1, aaa, 1>, <1, ab, 2>, <1, b, 2>, <2, \varepsilon, 1>\}$. As we can see, figure 7.2 shows a diagram of transitions of IFA $\boldsymbol{M}_2$, in which regular expressions aaa, ab, b,ε are used as arc markings. Such conception makes construction of the diagram easier and renders it compact and intuitive.

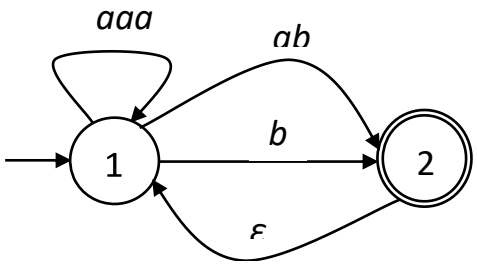


Figure 8.2. Diagram KA $\boldsymbol{M}_2$ with regular expressions

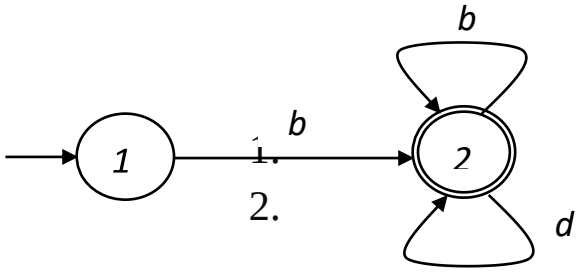


Figure 8.3. Diagram KA $\boldsymbol{M}_3$ for identifier.

KA $\boldsymbol{M}_3$ for recognition of identifiers consisting only of letters and numbers and starting with a letter will have the following parameters:

$\boldsymbol{Q}$={1,2}, $\boldsymbol{T}$={*b*,*d*}, $\boldsymbol{I}$={1}, $\boldsymbol{F}$={2}, δ(*1*,*b*)=*2*,δ(*2*,*b*)=*2*,δ(*2*,*d*)=*2*, where *b* – letter, *d* – number. The diagram KA $\boldsymbol{M}_3$ is shown in the figure 7.3.

Note 7.3.If a diagram contains several transitions with the same starting and ending point, they are called *parallel transitions*. Usually parallel transitions are indicated in a diagram with a single arrow. The markings of transitions are separated with commas. In figure 7.4 a diagram KA $\boldsymbol{M}_4$ is shown with parallel transitions for chains *ab*, *b*.

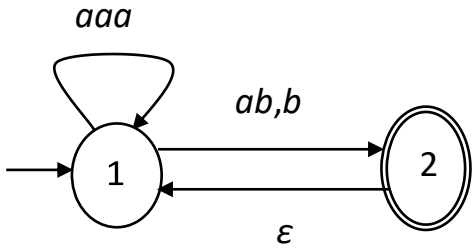


Figure 7.4. Diagram KA $\boldsymbol{M}_4$. with parallel transitions

The KA transitions may be represented as functions by means of a table or commands.

Convention 7.1.Among all KA states the initial state q_s and final state q_f stand out; here s and f are understood not as numeral variables but as mnemonic marks of start (*start*) and end (*final*).

Examples 7.3. In the table *7*.1 the function of transition δ KA $\boldsymbol{M}_5$ is shown determined by the sets $\boldsymbol{Q}$ = {q_s, q_1, q_2, q_3} and $\boldsymbol{T}$= {t_1, t_2, t_3}.

Table *7*.1. Values of the function of transition δ KA $\boldsymbol{M}_5$.

δ		Input		
		t_1	t_2	t_3
State	q_s	q_2	q_2	q_2
	q_1	q_3	q_s	q_s
	q_2	q_2	q_2	q_2
	q_3	q_3	q_2	q_s

The function of transition in the table *7*.1 may be represented as commands in the following way:

$\delta(q_s, t_1) = q_2$, $\delta(q_s, t_2) = q_2$,$\delta(q_s, t_3) = q_2$,

$\delta(q_1, t_1) = q_3$, $\delta(q_1, t_2) = q_s$,$\delta(q_1, t_3) = q_s$,

$\delta(q_2, t_1) = q_2$, $\delta(q_2, t_2) = q_2$,$\delta(q_2, t_3) = q_2$,

$\delta(q_3, t_1) = q_3$, $\delta(q_3, t_2) = q_2$,$\delta(q_3, t_3) = q_s$.

Let КА $\boldsymbol{M}$ be given with initial state $q_s \in \boldsymbol{Q}$,current state $q \in \boldsymbol{Q}$, final state $q_f \in \boldsymbol{Q}$ and unused current input chain $\tau \in \boldsymbol{T}^*$. Then the following description may be given.

Definitions 7.3:

1. If the head unit views the leftmost symbol of the input chain, then the pair $(q_s,\tau) \in \boldsymbol{Q} \times \boldsymbol{T}^*$ is called *initial configuration* КА;

2. If the head unit views the current symbol of the input chain τ, then the pair $(q,\tau) \in \boldsymbol{Q} \times \boldsymbol{T}^*$ is called *current configuration* КА;

3. If the input chain τ has been read completely, then the pair $(q_f, \varepsilon) \in \boldsymbol{Q} \times \boldsymbol{T}^*$ is called *final configuration* КА;

Note 7.4. By its contents the configuration is an "instantaneous description" of КА. Assuming that the initial chain whose belonging to the language under discussion is to be verified is in the tape, then in the configuration (q,τ) the chain τ is the part of the initial chain which remains in the tape.

The step of КА is determined by the state of the controlling unit and the input symbol being viewed at that moment. The step itself consists in the change of state of the controlling unit and the shift of the head unit one cell to the right.

The Step КА $\boldsymbol{M}$ is yielded by the binary relation $\models_{\boldsymbol{M}}$, determined over its configurations in the set $\boldsymbol{Q} \times \boldsymbol{T}^*$. If the automaton is known, then the letter $\boldsymbol{M}$ in the relation $\models_{\boldsymbol{M}}$ may be omitted.

Let $t \in \boldsymbol{T}$ be the leftmost symbol of the input chain still not read and both for $q \in \boldsymbol{Q}$ and $p \in \boldsymbol{Q}$ $<q, t, p> \in \Delta$ holds true; then for the chains $\tau \in \boldsymbol{T}^*$ the relation $(q, t\tau) \models (p, \tau)$ is true which determines the step of the automaton; this means that the automaton is in the state q and the state unit is viewing the symbol t in the input tape*;* then КА $\boldsymbol{M}$ moves into the state p and the head unit moves one cell to the right. If $\tau = \varepsilon$, then the input chain is considered to have been *read completely*.

Examples 7.4. Let $\tau = abba$. Then in the diagram КА $\boldsymbol{M_2}$ in the figure *7*.3 there is a step determined as relation $(1, abba) \models (2, ba)$.

Definition 7.4. $\models^k$ is the k–th degree of relation $\models$, if a chain of k+1 configurations exist

$(q_0,\tau_0), (q_1,\tau_1), (q_2,\tau_2),\ldots, (q_{k-1},\tau_{k-1}), (q_k,\tau_k)$

so that for any i ($1 \le i \le k$) the relation is true

$(q_{i-1},\tau_{i-1}) \models (q_i,\tau_i)$, where $q_0=q_s$, $\tau_0=\tau$, $q_k= q_f,\tau_k=\varepsilon$.

If for any $i \ge 1$ or $i \ge 0$ $(q_0,\tau) \models^i (q_i,\varepsilon)$ holds true, then we may write $(q_0,\tau) \models^+ (q_i,\varepsilon)$ or $(q_0,\tau) \models^* (q_i,\varepsilon)$ correspondingly. Here by $\models^+$ is denoted the transitive closure of relation $\models$, and by $\models^*$ – the reflexive and transitive closure of relation.

Definition 7.5. Automaton ***M*** recognizes input chain τ, if the relation $(q_s,\tau) \models^* (q_f,\varepsilon)$ holds true.

Examples 7.5. Let τ = *aaaab*. Then in KA $\boldsymbol{M}_2$ in the figure *7*.3 following relations (1, *aaaab*) $\models$(1, *ab*) and (1, *ab*) $\models$ (2, ε) hold true.

Definition 7.6. If the language ***L*** consists only of input chains recognized by automaton ***M***, then this language is recognized by automaton ***M*** and is denoted as ***L***(***M***), i.e.

$\boldsymbol{L}(\boldsymbol{M}) \rightleftharpoons \{\tau:\ \tau \in \boldsymbol{T}^*\ \ \&\ \ (q_s,\tau) \models^* (q_f,\varepsilon)\}$.

Lemma 7.1. If $(q_1, x) \models^* (q_2,\varepsilon)$ and $(q_2, y) \models^* (q_3,\varepsilon)$ is true, then $(q_1, xy) \models^* (q_3,\varepsilon)$ is true.

Proof. For this it is necessary to perform induction by a number of steps in the program of work KA, leading from configuration (q_1,x) to configuration (q_2,ε).

Examples 7.6. Let for $\boldsymbol{M}_6$=<{q_s,q_1,q_f},{0,1},q_s,{q_f},⊢,⊣,δ>

there exist the following transition relations:

<q_s,0,{q_1}>,<q_s,1,{q_s}>,<q_1,0,{q_f}>,<q_1,1,{q_s}>,<q_f,0,{q_f}>,<q_f,1,{q_f}>

KA $\boldsymbol{M}_6$ recognizes all chains of zeroes and ones in which there are two zeroes in a row. The conditions may be interpreted in the following way:

q_s–initial condition indicates that “two zeroes in a row have not been detected and the initial symbol is a zero”;

q_1–state indicates that “two zeroes in a row have not been detected and the initial symbol is a zero”

q_f– final condition shows that “two zeroes in a row have been detected”.

It may be noted that KA $\boldsymbol{M}_6$, once entering the state q_f, remains in that state.

For the initial chain 01001 the only possible chain of configurations starting from configuration $(q_0, 01001)$ will be $(q_s,01001) \models (q_1,1001) \models (q_s,001) \models (q_1,01) \models (q_f,1) \models (q_f, \varepsilon)$.

Thus, $01001 \in \boldsymbol{L}(\boldsymbol{M}_6)$.

The diagram of this automaton is shown in the figure 7.5.

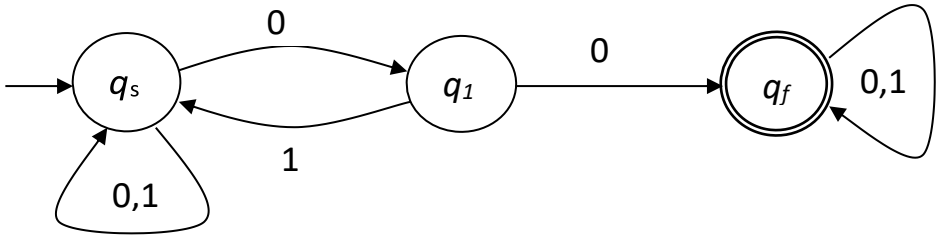


Figure 7.5. Diagram KA $\boldsymbol{M}_6$.

Definitions 7.7:

1. Path KA is a tuple $<q_0, r_1, q_1, r_2,\ldots, q_n>$, where $n \geq 0$ and $r_i = <q_{i-1}, \tau_i, q_i> \in \Delta$ for each i, $1 \leq i \leq n$. Here q_0 – *beginning of the path*, q_n – *end of the path*, $\tau_1 \ldots \tau_n$ – *mark of the path*, n – *length of the path*.

2. A path is called *successful* if its beginning belongs to $\boldsymbol{I}$ and its end belongs to $\boldsymbol{F}$.

Note 7.5. For any state $q \in \boldsymbol{Q}$ there exists a path$<q>$. Its mark ε, beginning and end coincide.

Examples 7.7. Let us consider KA $\boldsymbol{M_2}$in the figure 7.3 Let $\tau = baaab$. Then the path $<1,<1,b,2>,2,<2,\varepsilon,1>,1,<1,aaa,1>,1,<1,b,2>,2>$ is successful. Its mark is *baaab*, and its length is 4, i.e.: $q_0=1$, $q_1=2$, $q_2=1$, $q_3=1$, $q_4=2$;

$r_1=<1,b,2>$, $r_2=<2,\varepsilon,1>$, $r_3=<1,aaa,1>$, $r_4=<1,b,2>$;

$\tau_1=b$, $\tau_2=\varepsilon$, $\tau_3= aaa$, $\tau_4= b$.

Using the concept “path” it is possible to give alternative definitions to already introduced concepts of recognized chain and language.

Definitions 7.8:

1. Chain $\tau \in \boldsymbol{T}^*$ *is recognized* KA $\boldsymbol{M}$, if it is the mark of a successful path.

2. KA $\boldsymbol{M}$ *recognizes* a language $\boldsymbol{L}(\boldsymbol{M})$, if it consists only of marks of all successful paths.

Note 7.6. If $\boldsymbol{I} \cap \boldsymbol{F} \neq \varnothing$, then the language recognized by KA $\boldsymbol{M} = <\boldsymbol{Q}, \boldsymbol{T}, \vdash, \dashv, \boldsymbol{I}, \boldsymbol{F}, \Delta>$ contains an empty chain ε.

Examples 7.8. If КА $\boldsymbol{M_7}$= <$\boldsymbol{Q}$, $\boldsymbol{T}$,⊢,⊣,$\boldsymbol{I}$, $\boldsymbol{F}$,Δ> is given as $\boldsymbol{Q}$ = {q_1,q_2}, $\boldsymbol{T}$ = {a,b}, $\boldsymbol{I}$ = {q_1}, $\boldsymbol{F}$ = {q_1,q_2}, Δ = {<q_1,a,q_2>, <q_2,b,q_1>}, then it is determined and recognizes the following language:

$\boldsymbol{L}(\boldsymbol{M_7}) = \{(ab)^n: \ n \geq 0\} \cup \{(ab)^n a: \ n \geq 0\}$.

The diagram of this automaton is shown in the figure 7.6.

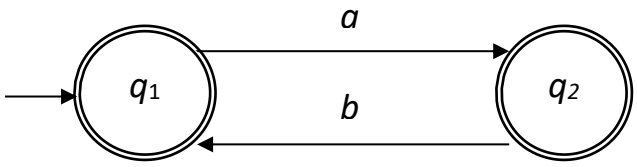


Figure 7.6. Diagram КА $\boldsymbol{M_7}$.

Definition 7.9. DFA $\boldsymbol{M}$ = <$\boldsymbol{Q}$, $\boldsymbol{T}$,⊢,⊣,$\boldsymbol{I}$, $\boldsymbol{F}$,Δ>, is called *full*, if for any state $q \in \boldsymbol{Q}$ and for any symbol $t \in \boldsymbol{T}$ there exists such state $p \in \boldsymbol{Q}$ that <q, t, p>∈Δ, i.e. $\delta(q, t) = p$.

Examples 7.9. The diagram of full automaton $\boldsymbol{M_8}$ with the following parameters Δ = {<1,a,2>, <1,b,3>, <2,a,3>, <2,b,1>, <3,a,3>, <3,b,3>}, $\boldsymbol{Q}$ = {1,2,3},$\boldsymbol{T}$ = {a,b}, q_s = {1}, $\boldsymbol{F}$ ={1,2} is shown in the figure 7.7.

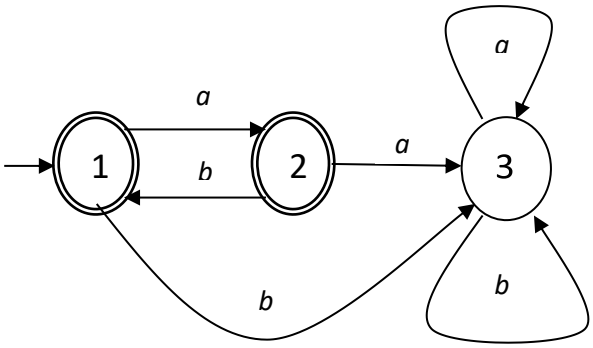
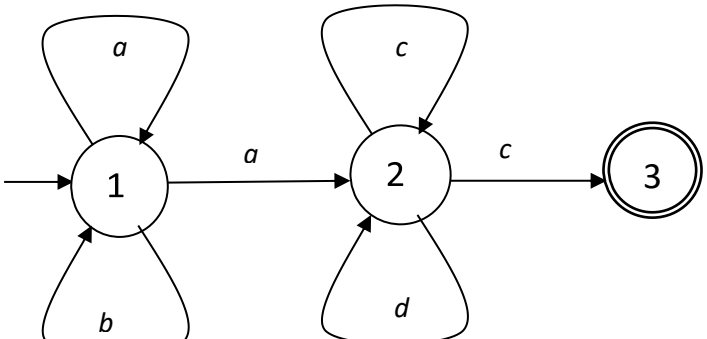


Рисунок 7.7. Диаграмма КА $\boldsymbol{M_8}$

1. Find a КА recognizing language $\{\alpha\beta: \ \alpha \in \{a,b\}^*, \beta \in \{a,b\}^*\}$.
2. Find a КА recognizing language $\{a,b\}^* \setminus (\{a^n: \ n \geq 0\} \cup \{b^n: \ n \geq 0\})$.
3. Find a КА recognizing language $\{a\xi b: \xi \in \{a,b\}^* \cup \{b\xi a: \ \xi \in \{a,b\}^*\}$.
4. Find a КА recognizing language $\{\tau \in \{a,b\}^*: \ |\tau|_a \geq 3\}$.
5. Find a КА recognizing language $\{a^m b^n a^m b^n: m,n \geq 1\}$.
6. List all configurations (q, τ), satisfying the condition (1, *abaacdcc*) $\models^*$ (q, τ), in КА $\boldsymbol{M_9}$ shown in the figure 7.8.



7. Find the step of the automaton if it is determined as

$\boldsymbol{M}$ = <{ q_0,q_1,q_2,q_f}, {a, b, c}, δ, q_0, {q_f}>,

where δ(q_0,a)={ q_1,q_2}, δ(q_1,a)={q_1}, δ(q_1,b)={q_f}, δ(q_2,c)={q_f},

$\boldsymbol{L}$($\boldsymbol{M}$) = {ac}∪{$a^n b$: n≥1}.

8. Find the full determined finite automaton for language (a∨b)*(aab∨$abaa$∨abb)(a∨b)*.

9. Find the full determined finite automaton for language (b∨c)((ab)*c∨(ba)*)*.

10. Find the full determined finite automaton for language (b∨c)*((a∨b)*c(b∨a)*)*.

Questions 7:

1. Is КА $\boldsymbol{M_{10}}$ shown in the figure рисунке 7.9. determined?

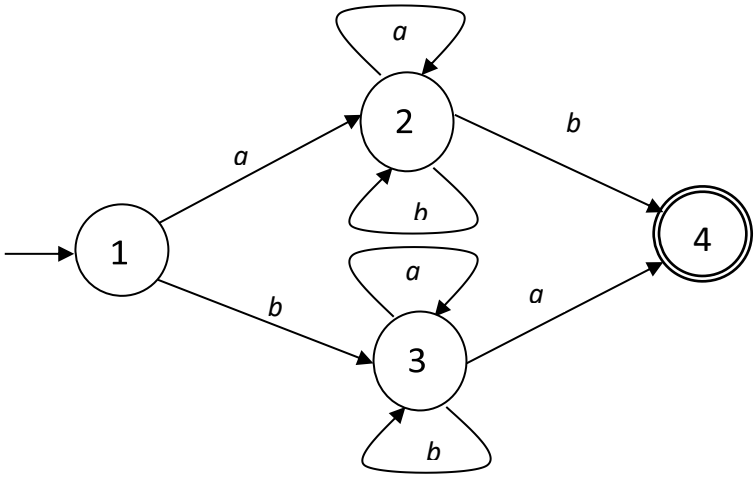


Figure 7.9. Diagram КА $\boldsymbol{M_{10}}$.

2. Do КА states q_1, q_2 and chains α,β,δ exist such that the relations (q_1, $\alpha\beta$) ⊨* (q_2, β) и ¬ (q_1, $\alpha\delta$) ⊨* (q_2, δ) hold true?

3. How are |$\boldsymbol{Q}$|, |$\boldsymbol{T}$|, |Δ|,|τ| and the number of configurations attainable from (q,τ) related in the sense of ⊨*?

4. What automaton can recognize the language generated by the regular expression (*abab*)∨(*aba*)*?

5. What contains the input tape?
6. What determines the direction of the shift of the head unit?
7. What does the automaton configuration consist of?
8. What types of configurations exist?
9. What does an automaton – recognized language consist of?

10. Is the determined finite automaton M_{11} with alphabet $T = \{a, b, c\}$ shown in the figure 7.10 full?

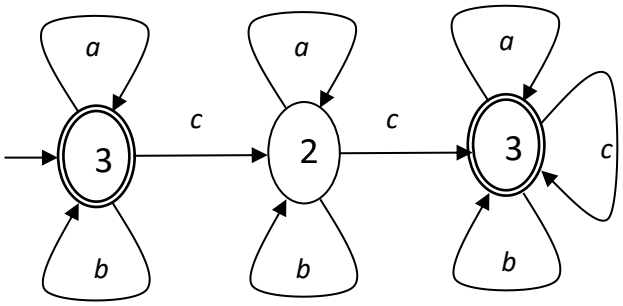


Figure 7.10. Diagram KA M_{11}.

11. Is the determined finite automaton M_{12} with alphabet $T = \{a, b\}$ shown in the figure 7.11. full?

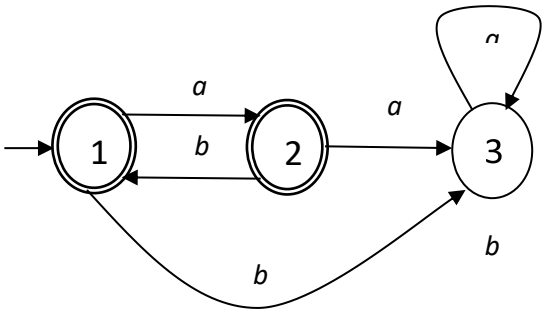


Figure 7.11Diagram KA M_{12}.

12. What does the graph of transition of finite automaton satisfying a given grammar look like?

$G = < \{+, -, *\}, \{A, B, C\}, P, C >$.

$P: A \to + \mid A+$

$B \to - \mid B- \mid A- \mid A+$

$C \to * \mid C* \mid B* \mid B- \mid A+$

Tests 7:

1. Finite automatons move to a state in accordance with:

A) transition table in the automaton's memory;

B) given task;

C) figures;

D) directions;

E) contents.

2. Which automaton is called determined?

A) if for any acceptable configuration of the identifier arising at one of the steps of its operation there exist two configurations in one of which the identifier will move in the following step;

B) if for any acceptable configuration of the identifier arising at one of the steps of its operation there exists a uniquely possible configuration in which the identifier will move in the following step;

C) if the identifier has an acceptable configuration for which there exists a finite set of configurations possible at the next step of operation;

D) if the identifier allows reading input symbols in one direction only ("from the left to the right");

E) if the identifier allows that the reading device move in both directions with respect to the chain of input symbols – both forwards from the beginning of the tape to its end and backwards going back to previously read symbols.

3.Finite automaton is a five – element set $\boldsymbol{M}$**=** $<Q, T, \delta, q_0, F>$, where Q is

A) a finite set of acceptable input symbols;

B) a finite set of states;

C) transition function;

D) initial state;

E) final state.

Table 2. Examples of real numbers with floating point.

№	Example	Mantissa	Order	Value
6.	$-12.*10^3$	–12	3	–12000

7.	$0.3*10^{+2}$	0.3	2	30
8.	$254*10^{-2}$	254	–2	2.54
9.	$1.5*10^{1}$	1.5	1	15
10.	$+2.17*10^{2}$	2.17	+2	217

One and the same real number with floating point can be represented in different ways. For example, the same number of 3.14 may be recorded:

$$314.*10^{-2} = 31.4*10^{-1} = 3.14*10^{0} = 0.314*10^{1} = 0.0314.*10^{2} = \ldots$$

To have a single entry for the submission of real number with floating-point we need to normalize it to the following condition:

$$q^{-1} \le |M| < 1,$$

where $|M|$ - the absolute value.

For example, real numberw with floating point in a normalized form are as follows:

$$0.1364*10^{4} \text{ and } 0.617*10^{-7}.$$

In order to simplify the arithmetic operations in the computer special codes to represent numbers are used. We consider direct code, inverse code and additional code of numbers.

Direct code of binary number is itself a binary number, and a sign of the binary number is written by dinary digit: "-" sign - the number 1, "+" sign - digit 0. For example, a negative binary number 1011_2 in direct code is written as 1.1011.

Representation of numbers in a computer, compared with forms well known since high school, has two important differences:

- numbers are recorded in the binary number system;
- for recording and processing of numbers a finite number of places are assigned (in the ordinary - non-computer arithmetic has no limit).

Addition and multiplication of binary numbers is done according to the table of addition and multiplication:

Addition of binary numbers	**Multification of binary numbers**
0 + 0= 0	0 · 0 = 0

0 +1 = 1 1 + 0 = 1 1 + 1 = 10	0 · 1 = 0 1 · 0 = 0 1 · 1 = 1

Arithmetic device in computer performs an action not with the binary numbers according to the rules of binary arithmetic, but with their binary codes according to the rules of arithmetic binary codes.

Differences between the rules of arithmetic of binary codes from ordinary arithmetic is in limit of discharge grid. In other words, for the record of number in the computer memory a fixed number of places is allocated. Computer memory has byte structure, however, the size of one addressed cell is typically several bytes: 2, 4, 8 bytes.

All the information on the computer is represented in binary code. From the whole set of codes, we consider the direct, inverse and additional codes.

To record integer binary number in the direct code binary numbers are complemented by sign pool, which is assumed to be equal to "0" for positive numbers and "1" - for negative. In manual recording of numbers with sign, the sign pool, for convenience, is separated from significant pools by point.

For example, the decimal number (+12) in direct binary code is written as (0.1100), and a decimal number - so (-12) - (1.1100).

Direct code is used for storage of numbers in the computer memory, as well as for operations of multiplication and division.

Other forms of presenting numbers with sign are the inverse and additional codes. These codes allow you to replace the subtraction of integers with their addition, based on the principle: a - b = a + (-b).

Positive numbers recorded in direct, reverse and additional codes are the same.

Thus, positive decimal number 12 in direct, inverse and additional binary codes can be written as follows: (0.1100).

To convert a negative number from direct code into reverse, one shoud be saved in sign pool and numbers of significant pools should be reversed, i. e. "1" is replaced by "0" and "0" to "1".

Additional code of negative number is obtained from the inverse code of number by adding "1" to the least significant digit of this number.

Rules of adding in additional code:

1. Addition is made accroding to the rules of addition of binary numbers, including the sign pool.
2. If as a result of adding the transfer occurs (overflow) from sign pool, the transfer is ignored (discarded).
3. If the sign of addition does not coincide with the signs of additives (this situation can arise only when the signs are the same), there is an overflow of digit grid of computer and the result should be declared invalid.

Addition in reverse binary code differs from adding in additional code on only one rule: if as the result of the addition there was the transfer from sign pool, i.e., overflow has occurred, it is necessary to add "1" to the least significant digit.

Example 2.1.

6. +5 - positive integer 5
7. 3.14 - positive real number with fixed-point, the integer part 3, and the fractional part 14.
8. 0.2 - positive real number with fixed-point, the integer part 0 and fractional part 2.
9. -1.001 - negative real number with fixed-point, the integer part 1 and the fractional part of 001.
10. 0.0 - positive real number, the integer part 0 and the fractional part 0.

Example 2.2. Write a decimal number (-12) in direct, inverse, and the additional binary codes in six-digit cell:

1.01100 - direct code;
1.10011 - reverse code;
1.10100 - additional code.

In this example, one place is assigned to the sign of number, five places to the number itself, to the point in the discharge grid no place stands out. The number itself is shifted to the right edge, and the excess discharge (in direct code) recorded as "0". Then direct code is inverted to transfer to reverse.

Transfer of numbers from reverse (additional) code into direct code performed on the same rules as to reverse (additional) code from direct.

Example 2.3. To perform this operation: 15 - 7 in direct, reverse, and additional code:

	Decimal number	**Direct code**	**Reverse code**	**Additional code**
Data	15 – 7	0.1111 – 1.0111	0.1111 + 1.1000	0.1111 + 1.1001
Intermediate result	8		10.0111 + 1	1 0.1000
Final result	8		0.1000	0.1000

Example 2.4. To perform this operation: 7 – 15 in direct, reverse, and additional code:

	Decimal number	**Direct code**	**Reverse code**	**Additional code**
Data	–15 +7	0.1111 1.0111	1.0000 + 0.0111	1.0001 + 0.0111
Intermediate result	–8		1.0111	1.1000
Final result	–8		1.1000	1.0111 + 1 1.1000

Exercise 2.5

1. Determine the real numbers with floating point:

6)40,23;
7)–5;
8)$3.3*10^{-2}$;

9)5.1+6i;

10) 0.14+7i.

2. Move a specified number from one number system to another:

1) 10000001 from binary to decimal system.

2) 129 from decimal to octal system.

3) 1952 from decimal to hexadecimal system.

3. Arrange the arithmetic operations so that it is true the following equation in the binary system: 1100 ? 11 ? 100=100000.

Questions 2.

1. What is a number system?
2. For what groups real numbers are divided?
3. Can the same numeric value be represented in the different number systems?
4. What are the types of numeric values?
5. For what groups real numbers are divided?

Test 2.

1. In what system data is coded in ANSI?

A) in binary system

B) in ternary system

C) in octal system

D) in decimal system

E) in hexadecimal system

2. In what system data is coded in in Unicode?

A) in hexadecimal system

B) in ternary system

C) in octal system

D) in decimal system

E) in binary system

3. How many bytes are used for encoding in Unicode?

A) 2

B) 1

C) 3

D) 5

E) 4

4. What is the number system?

A) A recording method using the numbers and a set of rules.

B) Possibility to record values of the numbers in a given range.

C) Each sequence of numbers identifies only one numerical value.

D) Easiness of performing of operations.

E) Values of numbers do not depend on their position in the record of number.

5. Which number system is the smallest?

A) binary.

B) octal.

C) hexadecimal.

D) Ternary.

E) Decimal.

3. Bases of mathematical logic

Statements and logic connectives. The logic form of the statement: the subject, a predicate, connectives, premises. Conclusions: deductive, inductive. Concepts of the proof. Logic connectives: disjunction, conjunction, negation,

implication, equivalence. Truth tables. Logic functions. Concepts of a tautology and the

Statements

The content of any science make statements (propositions) about the objects of her subject domain. Propositional logic is abstracted from the specific content of the statements and studies the structure of complex sentences and their logical connections.

Statement is the declarative proposition, wich can be true or false. Examples of statements: "Snow is white", "2> 3", "If there is rain, then I take an umbrella", etc.

Statements can be linked to each other by means of logical connections, "not", "and", "or", "implication", "equivalent."

Mathematical logic, we will study with the help of mathematical methods in a some meta-language, which is different from the subject language of the studied logic. Subject language of propositional logic consists of the alphabet and formulas:

Alphabet:

(1) P, Q, R, ... - variables for simple statements (propositional letters);

(2) $\neg$, &, $\vee$, $\rightarrow$, $\leftrightarrow$ - symbols on the statements of operations (logical ligament);

(3) (,) - auxiliary characters (braces).

The formulas or complex statements:

(1) P, Q, R, ... - propositional letters - elementary formula (atoms);

(2) if A, B - formula, $\neg$A, A&B, A$\vee$B, A$\rightarrow$B, A $\leftrightarrow$B - formula.

In the definition of the formulas used metaletters A, ie characters that do not belong subject language.

Examples of formulas: $\neg$P, (P&Q), (R$\rightarrow$ (P $\vee$ R)).

Subformulas - is part of the formula, is the formula itself.

Set Language, we have built a formal system. Now imagine it as meaningful propositional algebra, for this we give the meaning symbols of alphabet and formulas. Propositional letters, and logical operations are defined in the field of two elements {T, F}, T - True, F - False:

P	Q	P&Q	P∨Q	¬P	P→Q	P↔Q
T	T	T	T	F	T	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

The value of the formula E [P_1, ... , P_n] at this interpretation of its constituent propositional letters
γ : {P_1, ... , P_n} $\Rightarrow${T,F} we define by induction on the structure of the formula:

E = P : E[γ] = γ (P);
E = ¬A : E[γ] = ¬A[γ];
E = A&B : E[γ] = (A&B)[γ] = A[γ] & B[γ];
E = A∨B : E[γ] = (A∨B)[γ] = A[γ] ∨ B[γ];

If in the formula the operation ¬ is used only one, the formula is called the *formula with negation.*

Tautology (universally valid formula, logical law) - a formula, true for all interpretations of its constituent propositional letters, in other words, - the column of values, which contains only true values (denoted by the symbol ⊨)

Basic tautology.

1a. ⊨A→(B→A)

1б. ⊨(A→B)→((A→(B→C))→(A→C))

2. ⊨A→(B→A&B)

3a. ⊨A&B→A

3б. ⊨A&B→B

4a. ⊨A→A∨B

4б. ⊨B→A∨B

5. ⊨(A→C)→((B→C)→(A∨B→C))

6. ⊨(A→C)→((A→¬C)→ ¬A)

7. $\models \neg\neg A \rightarrow A$

8. $\models (A \rightarrow B) \rightarrow ((B \rightarrow A) \rightarrow (A \leftrightarrow B))$

9а. $\models (A \leftrightarrow B) \rightarrow (A \rightarrow B)$

9б. $\models (A \leftrightarrow B) \rightarrow (B \rightarrow A)$

10. $\models (A \rightarrow (\neg A \rightarrow C)$

Logic functions called n-place operation on the set {0,1}.

Alphabet:
(1) $x,y,...,x_1,x_2,...$ - individual variables;
(2) $f,g,...,f_1,f_2,...$ - functional symbols.
Term:
(1) $x,y,...,x_1,x_2,...$ - individual variables are terms;
(2) If $f^{(n)}$ - a functional symbol, $t_1,...,t_n$ - terms, then
$f^{(n)}(t_1,...,t_n)$ - term.
The value of the term:
(1) if t - object variable x, then Val $t = \gamma(x)$;
(2) if $t = f^{(n)}(t_1,...,t_n)$, then Val $t = f^{(n)}$(Val t_1,..., Val t_n).
Function:
$f^{(n)}(x_1,...,x_n)$ can be represented by the term $t(v_1, ..., v_m)$, if $\{v_1, ..., v_m\} \subseteq \{x_1,...,x_n\}$ and $t[\gamma] = f^{(n)}[\gamma]$ for all interpretations $\gamma : \{x_1,...,x_n\} \Rightarrow \{0,1\}$.

Examples 1.1:
1.The four-digit number of the 1952 decimal system is expressed thus:

$$1952_{(10)} = 1*10^3 + 9*10^2 + 5*10^1 + 2*10^0$$

2.The number of a decimal system with a three-digit integer part and a three-digit fractional part 596.174 (10) is expressed as follows:

$$596.174_{(10)} = 5*10^2 + 9*10^1 + 6*10^0 + 1*10^{-1} + 7*10^{-2} + 4*10^{-3}$$

3. The number of a binary system with a four-digit integer part and a three-digit fractional part 1010.101 (2) is expressed as:

$$1010.101_{(2)} = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$$

Examples II.
1. 3.14 - positive real number with fixed-point, the integer part 3, and the fractional part 14. 2. 5 - positive integer 5.
3. 0.2 - positive real number with fixed-point, the integer part 0 and fractional part 2.

4. -1.001 - negative real number with fixed-point, the integer part 1 and the fractional part of 001.

5. 0.0 - positive real number, the integer part 0 and the fractional part 0.

Tests III.

1. What will be important expression 2>5 ∨ 2<6?

A) 2
B) 1
C) 5
D) 6
E) 0

2. What order of operations an expression D∨¬F ∧ G?

A) first ¬F, then ¬F * G, and at the end D∨¬F ^ G.

B) first ¬F ^ G, and at the end D∨¬F ^ G.

C) first ¬F, and at the end D∨¬F ^ G.

D) first ¬F, then ¬F ^ G.

E), first ¬G, then ¬F ^ G, and at the end D∨¬F ^ G.

3. Which one is De Morgan's law?

A) $\neg(\neg p) \equiv p$
B) $p \equiv p$
C) $\neg(p \vee q) \equiv \neg p \wedge \neg q$
D) $p \wedge \neg p \equiv 0$
E) $p \vee \neg p \equiv 1$

4. Laws of logic

Lecture objective: explain the concept and definitions of the laws of logic and review their types.

Lecture plan: study the law of double negation, commutation law, distribution law, law of exclusion of constants, law of contradiction, law of excluded middle, the duality principle, logical corollary, rules of logical corollary, modus ponens rule.

Laws of logic **consist of the following tautologies:**

1) $\models A \vee \neg A$ (*law of excluded middle*)
2) $\models A \to A$ (*law of identity)*
3) $\models \neg(A \vee B) \sim \neg A \& \neg B$ (*first de Morgan's law*)
4) $\models \neg(A \& B) \sim \neg A \vee \neg B$ (*second de Morgan's law*)
5) $\models A \& A \sim A$, $\models A \vee A \sim A$
6) $\models A \to B \sim \neg A \vee B$
7) $\models (A \leftrightarrow B) \sim (A \to B) \& (B \to A)$
8) $\models (A \to B) \sim (\neg B \to \neg A)$ (contraposition law)
9) $\models A \& B \sim B \& A$ (*conjunction commutability*)
10) $\models A \vee B \sim B \vee A$ (*disjunction commutability*)
11) $\models A \& (B \& C) \sim (A \& B) \& C$ (*conjunction associativity*)
12) $\models A \vee (B \vee C) \sim (A \vee B) \vee C$ (*disjunction associativity*)
13) $\models A \& (B \vee C) \sim (A \& B) \vee (A \& C)$ (*first law of distributivity*)
14) $\models A \vee (B \& C) \sim (A \vee B) \& (A \vee C)$ (*second law of distributivity*)
15) $\models A \& (A \vee B) \sim A$, $\models A \vee (A \& B) \sim A$ (*absorption laws*)
16) $\models A \& И \sim A$, $\models A \& Л \sim Л$, $\models A \vee И \sim И$, $\models A \vee Л \sim A$.
17) $\models A \to (B \to C) \sim A \& B \to C$.

Let E be a formula with close negations which does not contain other operations except $\neg$,&, $\vee$. The E^X formula is the result of substituting all conjunctions in E with disjunctions and each proposition letter with its negation. Then $\models \neg E \sim E^X$.

The duality principle. Let E, F not contain other operations except $\neg$, &, $\vee$ and let them be formulas with close negations. The formulas E′, F′ obtained from E, F by simultaneous substitution of all & with $\vee$ and $\vee$ with & are called

dual with regard to the formulas E and F correspondingly. Then the following relations exist:

b) if $\models \neg E$, then $\models E'$. b) if $\models E$, then $\models \neg E'$.

c) if $\models E \sim F$, then $\models E' \sim F'$. d) if $\models E \rightarrow F$, then $\models F' \rightarrow E'$.

Logical corollary. Let there be formulas $A_1,A_2,...,A_m$ and B. If from the simultaneous truth of the formulas $A_1,A_2,...,A_m$ there follows the truth of the formula B, then the formula B is a logical corollary of the formulas $A_1,A_2,...,A_m$; this is indicated as $A_1,A_2,...,A_m \models B$, $(m \geq 1)$, where $A_1,A_2,...,A_m$ are premises and B is a corollary.

Logical corollary rules. For computation of relations one single rule called modus ponens is used which represents a procedure of transition from two formulas of the type $A, A \rightarrow B$ (premises) to the formula B (corollary):

$$\frac{A,\ A \rightarrow B}{B}$$ (modus ponens)

Corollary rules must satisfy the requirement that true premises lead to true corollaries.

Predicates are logical functions $J^{(n)}(x_1,...,x_n)$ *given in a non-empty space* D *and acquiring value in the set* {И,Л}.

The predicate $J^{(n)}(x_1,...,x_n)$ becomes an expression after its variables are attributed to the elements of the set D.

***Alphabet*:**

(1) $x,y,z,...,x_1,x_2,...$ – object variables;

(2) $P^{(n)}(x_1,...,x_n),...$ – predicate letters $(n=0,1,...)$;

(3) &, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$ – logical connectives and quantors;

(4) (,) – auxiliary symbols.

Formulas:

(1) $P^{(n)}(x_1,...,x_n)$, …– elementary formulas or atoms;

(2) if A, B are formulas, then $A\&B$, $A \vee B$, $\neg A$, $A \rightarrow B$, $A \leftrightarrow B$ – are formulas as well;

(3) if $A(x)$ is a formula with a free variable x, then $\forall x A(x)$, $\exists x A(x)$ are formulas.

Free and bound variables. All variables existing in the space of action of the quantor at such variables are called *bound variables, otherwise they are called free variables.*

Formula interpretation. The value of the formula $E[P_1,...,P_m; x_1, ...,x_n]$ for interpretation of the predicate letters τ: $P^{(n)} \Rightarrow J^{(n)}$ and attribution of γ: $\{x_1,...,x_n\} \Rightarrow D$ $(D \neq \varnothing)$ to object variables is denoted $E[\tau,\gamma]$. Let us define induction for construction of the formula E:

5) $E = P^{(n)}(x_1,...,x_n)$, then $E[\tau,\gamma] = J[\gamma]$;

6) $E = (A\&B)[P_1,...,P_m; x_1,...,x_n]$, then $E[\tau,\gamma] = A[\tau,\gamma] \& B[\tau,\gamma]$.

Analogously for other logical connectives.

7) $E=\forall x_1 A[P_1,...,P_m;x_1,...,x_n]$, then $E[\tau,\gamma] = \forall x_1 A[\tau,x_1,\gamma]=И$,

where γ: $\{x_2,...,x_n\} \Rightarrow D$, if $A[\tau,a,\gamma] = И$ for any $a \in D$.

8) $E=\exists x_1 A[P_1,...,P_m; x_1, ...,x_n]$, then $E[\tau,\gamma] = \exists x_1 A[\tau,x_1,\gamma] = И$,

where γ: $\{x_2,...,x_n\} \Rightarrow D$, if $A[\tau,a,\gamma] = И$ for some $a \in D$.

The formula $E[P_1,...,P_m; x_1,...,x_n]$ is called a universally valid formula or tautology if for any space $D \neq \varnothing$, for any interpretations τ of predicate letters and any attributes γ to object variables in interval D, $E[\tau,\gamma] = И$.

Logical foundations of computer consist of logic algebra which emerged in mid-19th century in the works of English mathematician John Boole. Its creation was due to an attempt to solve traditional logical problems by algebraic methods using logical operations such as $\neg$, &, $\vee$ denoting words and word combinations "*not*", "*and*", "*or*"". With help of these logical operations a logical expression of any complexity may be constructed.

Hardware implementation of the mentioned logical operations is realized by means of the following logical elements of computer shown in figure 4.

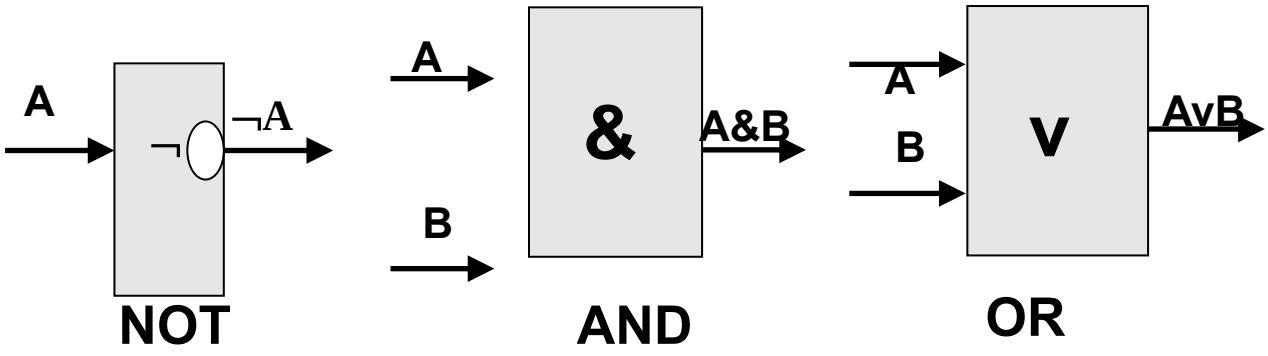


Figure 4. Logical elements of computer.

Examples 4.1.

Let us show that the formula $P(x,y) \rightarrow Q(x)$ is not 1-valid and, consequently, not universally valid.

Solution. D={1} is one-element set, I_1 and I_2 – interpretations of the letter P, and J_1 and J_2 – interpretations of the letter Q:

x	y	I_1	I_2	J_1	J_2
1	1	И	Л	И	Л

Truth-table of the formula $P(x,y)\to Q(x)$:

x	y	$P(x,y)$	$Q(x)$	$P(x,y)\to Q(x)$
1	1	И	И	И
1	1	И	Л	Л
1	1	Л	И	И
1	1	Л	Л	И

Examples 4.2.

Let us show that the formula $\forall x\exists y P(x,y)\to\exists y\forall x P(x,y)$ is not universally valid.

Solution. Let D={1,2}, then the interpretations of the predicate letter $P(x, y)$ may be given by means of the following table:

X	Y	J_1	J_2	J_3	J_4	…	J_7	…
1	1	И	И	И	И	…	И	…
1	2	И	И	И	И	…	Л	…
2	1	И	И	Л	Л	…	Л	…
2	2	И	Л	И	Л	…	И	…

In particular, for interpretation J_7 we obtain: for x=1: $\exists y J_7(1,y)\equiv$И; for x=2: $\exists y J_7(2,y)\equiv$И, then $\forall x\exists y J_7(x,y)=$И. For y=1: $\forall x J_7(x,1)=$Л, for y=2: $\forall x J_7(x,2)=$Л, then $\exists y\forall x J_7(x,y)=$Л. It follows that $\forall x\exists y J_7(x,y)\to\exists y\forall x J_7(x,y)$ = Л.

Examples 4.3.

Let us show that the formula $\forall x(\exists x P(x)\to P(x))$ is not 2-valid.

Solution. D={1,2}, J_1, J_2, J_3, J_4 – interpretations of the letter P :

x	J_1	J_2	J_3	J_4
1	И	И	Л	Л
2	И	Л	И	Л

Truth-table of the formula $x\ (\exists x P(x) \to P(x))$:

x	$P(x)$	$\exists x P(x)$	$\exists x P(x) \to P(x)$	$\forall x(\exists x P(x) \to P(x))$
1	J_1	И	И	И
2	J_1		И	
1	J_2	И	И	Л
2	J_2		Л	
1	J_3	И	Л	Л
2	J_3		И	
1	J_4	Л	И	И
2	J_4		И	

Examples 4.4.

Let P be a false statement 1 = 5, Q is a false statement as well 3 = 7 and R is a true statement 4 = 4. Demonstrate that conditional statements: «if P, then Q» and «if P, then R» are both true.

Solution. If 1 = 5, then adding 2 to both parts of the equality we obtain 3 = 7. Therefore, the statement «if P, then Q» is true. Now let us subtract 3 from both parts of the equality 1 = 5 obtaining –2 = 2. Therefore, $(-2)^2 = 2^2$, i.e. 4 = 4. Therefore, «if P, then R» is true as well.

Problems 4.

1. Translate each of the following arguments into logical symbols and analyze the correctness of the result:

1) I would pay for television repair only if it functioned. It does not. For this reason, I will not pay.

2) If he had told her nothing, she would never have found it out. And if she had not asked him, he would not have told her. But she found it out. Therefore, she asked him.

3) He said he would come if it did not rain. But it is raining. Therefore, he will not come.

2. Check the correctness of argument: Ivanov will not do this work if Petrov does it. Petrov and Sidorov will do this work if and only if Ivanov does it.

Sidorov will do this work, and Ivanov will not. Therefore, Petrov will not do this work.

3. Which formulas yield the following formula sequences: A⊃(B⊃C), A, B⊃C, B, C.

Questions 4.

7. Are the following expressions equivalent?
   6) A∧B and A and B?
   7) A∧B and not only A, but also B?
   8) A∧B and B, even though A?
   9) A∧B and B, in spite of A?
   10) A∧B and both A, and B?
8. Are the following expressions equivalent?
   6) A∨B and A or B?
   7) A∨B and A or B?
   8) A∨B and A, if not B?
   9) A∨B and A and B?
   10) A∨B and A or B?
9. Are the following expressions equivalent?
   6) A∼B and A, if and only if B?
   7) A∼B and if A, then B, and vice versa?
   8) A∼B and A, if B, and B, if A?
   9) A∼B and A equivalent to B?
   10) A∼B and A if and only if B?

10. For which of the statements *X*: *X=1, X=6, X=5, X=3, X=4* are the relations (*X*>3) & (*X*<5) true?

11. For which of the words “Informatics”, “Psychology”, “Economics” will the statement “The first letter is consonant, and the second letter is a vowel” be true?

12. Which of the following statements are true, and which are false?

(a) The sum of interior angles of any triangle is 180°.

(b) All cats have a tail.

(c) There is an integer x satisfying the equation $x^2 = 2$.

(d) There is an even prime number.

(e) Snow is white.

(f) The Earth revolves around the Moon.

(g) Paris is the capital of France.

(h) To govern is to know.

Tests 4.

7. What characterizes the law of excluded middle?

1) Implication of two statements is equivalent to the inverse implication of their negations.

2) Any statement is either false or true, no third possibility exists.

3) Any statement is the logical corollary of itself.

4) To negate a negation of a statement is equivalent to its assertion.

8. Interpretation is:

9) Concepts whose application to logical calculation expressions depends in great measure on the choice of interpretation.

10) Juxtaposition of every elementary expression p with a certain true value.

11) Concepts whose application to logical computation depends in great measure on the choice of interpretation.

12) Relation between objects which means that the state or properties of any of them change if the state or properties of others are changed.

9. Is the logical connective «or»:

6) connective?

7) exclusive?

8) divisive?

9) auxiliary?

10) negating?

10. What characterizes the law of double negation:

5) Any statement is either false or true, no third possibility exists.

6) Any statement is the logical corollary of itself.

7) To negate a negation of a statement is equivalent to its assertion.

8) Any statement is the logical corollary of itself.

11. Graphs.

The purpose of the lecture: to consider the concept of the graph, the types of graphs and their properties.

Outline of the lecture: to explore formal definitions and ways to represent graphs, to analyze different types of graphs and types of applications of graphs for various tasks.

Definitions 6.1:

The graph is a dynamic networking connected structure of data represented by of a plurality of pairs called vertices and edges. Each vertex can be connected with several other vertices or with itself by means of edges and vertices, which do not form a hierarchy. Formally, a graph is defined as a set of pairs of G = (X, A), where X - the set of vertices, A - the set of edges, actually is a relation on a set X, i.e. $A \subseteq X \times X$. If $x_i \in X$ and $x_j \in X$ – vertices , then (x_i, x_j) – edges.

There are several types of graph. If from each vertex of the graph originates equal number of edges and if equal number of edges goes in each vertex, such a graph is a regular graph. If for each edge of the graph direction is defined, the graph is called a directed graph. If each edge of the graph has a weight, a graph is called weigthed graph, i.e., you can define a function w : E, where R - the set of real numbers, w -weight of graph and w≥0.

Matrix of adjacency is one of the ways to represent a graph in the form of a matrix.

Matrix of adjacency of a graph G with a finite number of n vertices (numbered from 1 to n) is a square matrix A of size n, wherein the value of element a_{ij} equals to number of edges from the i-th vertex in the j-th vertex. Sometimes, especially in the case of an undirected graph, the loop (the edge of the i-th vertex in itself) counts as two edges, i.e., the value of the diagonal element a_{ij} in this case equals to double number of loops around the i-th vertex.

Matrix of adjacency of a simple graph (not containing loops and multiple edges) is a binary matrix which contains zeros on the main diagonal.

In graph theory are used following:

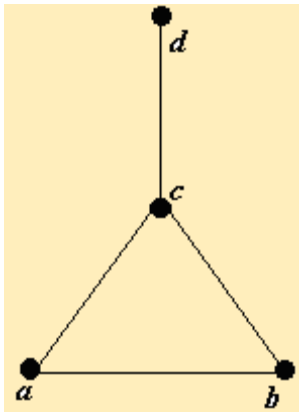
– Incidence matrix. This matrix A with n rows corresponding to the vertices and m columns corresponding to the edges. For a directed graph column corresponding to the arc (x, y) contains - 1 in the row corresponding to vertex x, and 1 in the row corresponding to the vertex y. In all others 0. Loop, i.e. arc (x, x) may be represented by a different value in the row x, e.g., 2. If an undirected graph, the column corresponding to the edge (x, y) contain 1, the corresponding x and y and zeros in all other rows.

– The matrix of adjacency. This is a matrix n × n where n - the number of vertices, where a_{ij} =1, if there is an edge going from vertex x to vertex y and a_{ij}=0 otherwise, i.e.:

a_{ij} - the number of edges connecting vertices v_i and v_j, and in A) in some applications of each loop (an edge $\{v_i, v_i\}$ for some i)) is counted twice;

B) adjacency matrix of empty graph, does not contain any edges, consists of zeroes.

Below are examples of incidence matrix of and adjacency matrix for continuous graph shown in Figure 6.1



	u	v	w	x
a	1	1	0	0
b	1	0	1	0
c	0	1	1	1
d	0	0	0	1

	a	b	c	d
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

Figure 6.1 Incidence matrix Adjacency matrix

Given a graph $G=(X, A)$, where $X=\{x_i\}$, i = 1, 2, ..., n – the set of vertices, $A=\{a_j\}$, j = 1, 2, ..., m – the set of arcs.

Subgraph $G'=(X', A')$ of the original graph G is a graph G ', for which $X' \subseteq X$ и $A' \subseteq A$. Examples of subgraphs are shown in Fig. 6.2, b, and original graph - Fig. 6.2 a.

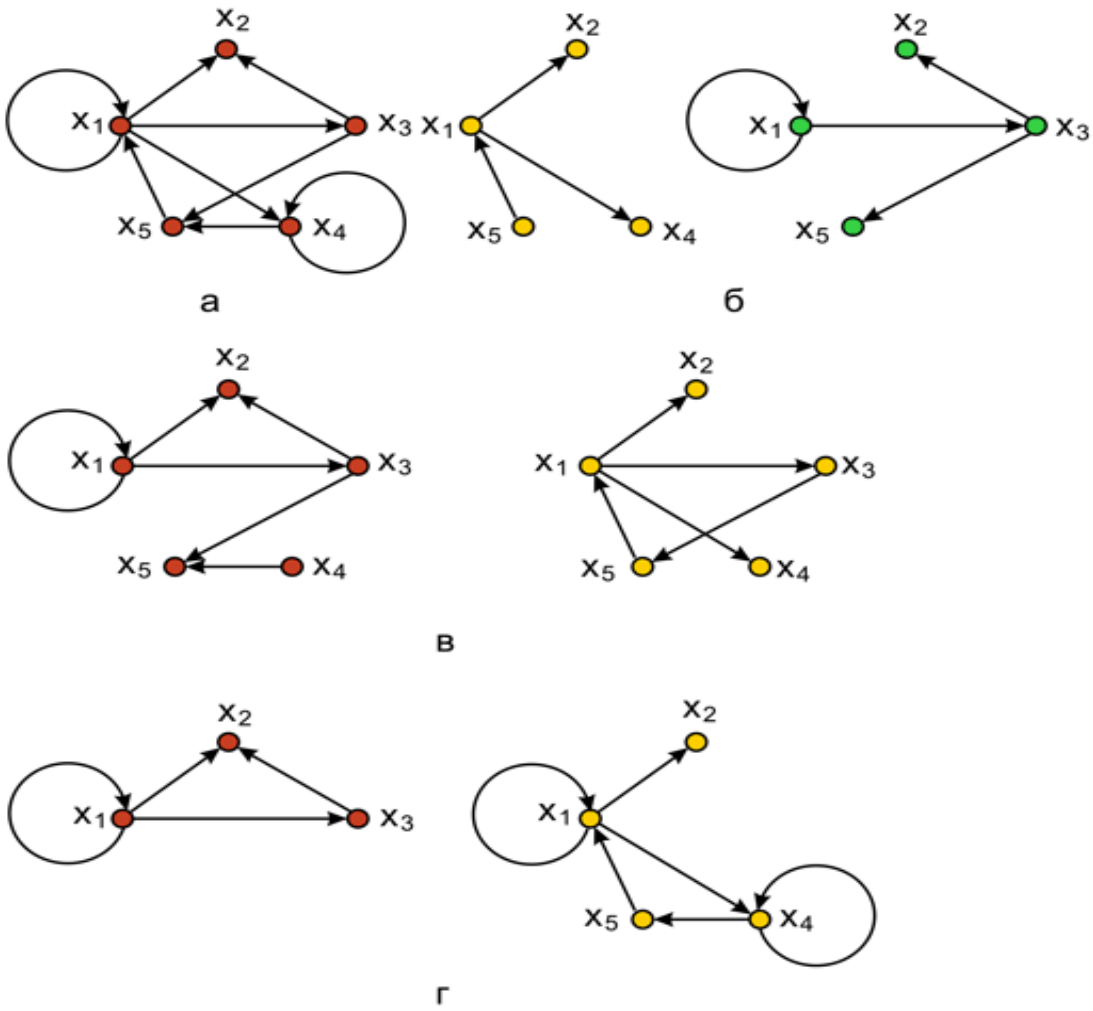


Figure 6.2. Types of subgraphs: а - the original graph; б - subgraphs; в - spanning subgraph; г - induced subgraphs

If A - adjacency matrix of the graph G, the matrix A^n has the following property: item at the i-th row, j-th column is equal to the number of paths from the i-th vertex to the j-th consisting of exactly n edges.

The path in a graph is a sequence of edges leading from one vertex to another vertex, such that every two neighboring edges have a common vertex and no edge occurs more than once, that is, formal path in a graph is a sequence of vertices $(x_1, x_2, x_3, ..., x_{m-1}, x_m)$, that pairs $\{(x_1, x_2),(x_2, x_3),...,(x_{m-1}, x_m)\}$ will be edges. Two vertices $x_i \in X$ and $x_j \in X$ in the graph is called connected (disconnected), if it exists (do not exist) the path leading from x_i to x_j. This path can be in both directions. If every two vertices in the graph are connected, then this graph is a connected graph. If the graph contains at least one pair of

disconnected vertices, the graph is disconnected. If all pairs of vertices connected in both directions, so the graph is strongly connected graph.

The path with no repeated edges is called a chain and the chain without repeated vertices called simple.

Chain in which the end vertices coincide is called *a cycle*, and the cycle in which no recurring peaks other than the end, called *simple*, i.e. the path way back to the same vertex, then that path is called the *closure (cycle)*, i.e. in the closure of the initial and final vertices are the same. If the closure does not pass through one of the vertices of the graph more than once, it is called a *simple closure*. If the closure originates from a single vertex and directly enters into the top back, it is called *a loop*, i.e, the loop has a unique vertex.

The length of the path is the number of edges of this path. If the weights of the edges are their length, then the path length is calculated as follows:

$$w(x_1, x_2, x_3, ..., x_{m-1}, x_m) = \sum_{i=1}^{m-1} w(x_i, x_{i+1}) .$$

In the graphs you can perform the following tasks: *a comparison of the two graphs, finding the shortest path from one vertex to another, finding the number of closed paths* and etc.

A tree is a graph in which all vetices are connected, and the paths are not closed, i.e., connected graph is without cycles and without loops.

The tree vertices are divided into the following types:

1) *the root* – a vertex, from which originates one or more edges, but enter no edge, i.e., a vertex, which does not have a single ancestor, but it can have many descendants;

2) *branch* - the vertex, to which enters a single edge, but many egdes can originate from it, i.e., the veretx which has a single ancestor and can have many descendants;

3) *sheet* - the vertex, to which enters only one edge, but originate no edge, i.e. the vertex which has a single ancestor, but does not have any descendants.

In the tree the direction of path passes through the branches from the root to the leaves. Inside the tree can be a few trees, which will be called *subtrees.* You can now give the following recursive definition (referring to itself):

1. *A recursive basis*: the set {v}, consisting of only one vertex v is a tree where its unique vertex is both the root and leaf.

2. *Recursive step:* if v - vertex and $A_1, A_2, ..., A_n$ - the trees, then it is possible to build a new tree in which the root is the vertex v, and edges – originates from this vertes and enters the roots of $A_1, A_2, ..., A_n$ trees.

3. *Recursive conclusion:* Trees obtained only by rules 1 and 2. This definition of a tree can be represented in Figure 6.3 as follows:

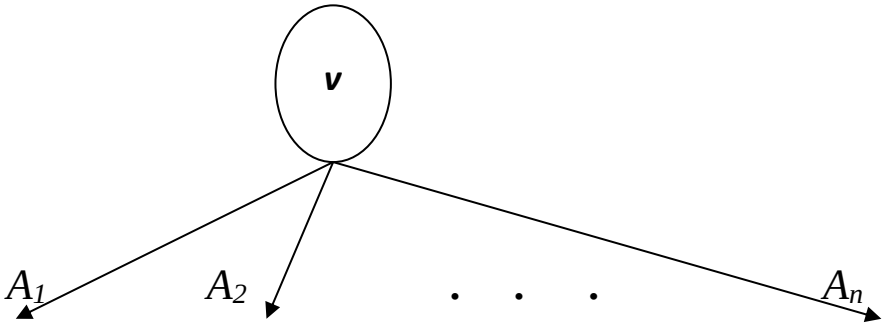


Figure 6.3. Determination of tree

From this definition it is clearly evident that the tree is a hierarchical connected dynamic structure of data represented by single root vertex and its descendants. The maximum number of descendants of each vertex and determines the size of a tree.

Among the trees stands out, the so-called *binary trees*. It can be defined as follows:

Binary Tree - a tree in which each node has at most two descendants. This node is called the *parent node* and the descendants are called *left heir* and *right heir*. We give a recursive definition of a binary tree. A binary tree is the following set of vertices:

– either contains nothing (the empty set);

– or consists of a root, which is connected with two binary trees, called left-hand subtree and right-hand subtree.

Thus, the binary tree is either empty or consists of data and two subtrees, each of which may be empty. If in some vertex two subtrees are empty, then it is a leaf. Formally, a binary tree is defined as follows: <binary tree> :: = nil | (<data> <binary tree> <binary tree>) where nil - empty.

The following tasks are solved in trees: *tree traversal, search for tree, adding a new node to the tree, destroying the tree tops, comparisons of trees* and others.

Binary trees are used in the search algorithms: each vertex of binary search tree corresponds to an element of a sorted set, all his left descendants the left to fewer elements, and all his right descendants to a great element. Each node in the tree is uniquely identified by a sequence of non-recurring vertices from the root and until it – by path. The path length is a level of node in the hierarchy tree. For practical purposes, generally two subspecies of binary trees are used: binary search tree - binary search tree (BST) and binary heap.

– Binary search tree has the following properties:
  – the left subtree and the right subtree are binary search trees;
  – all the vertices of the left subtree of v arbitrary vertex has value of key of data that is less than the value of key of data of the vertex v itself;
  – all the vertices of the right subtree of the same vertex v has value of key of data that is greater than the value of key of data of vertex v.

Clearly, data from each node should have keys on which the comparison operation is determined.

Binary heap or sorting tree has the following properties:

– value at any vertex is not less than the values at the vertices of its descendants;

– leaf depth (distance until the root) does not differ by more than one layer;

– the last layer is filled from left to right.

Such heap is called *max-heap*. There are also heaps, where the value in each vertex, conversely, no more than the values of its descendants. Such heaps are called *min-heap*.

Examples 6.2:

1. A binary relation over finite objects can be represented as a directed graph as shown in Figure 6.4. The following shows the relationship divisibility of integers from 1 till 12: 2 and 3 divided by 1; 4 and 6 is divided into two; 6 is divisible by 2 and 3; 12 divided by 4 and 6.

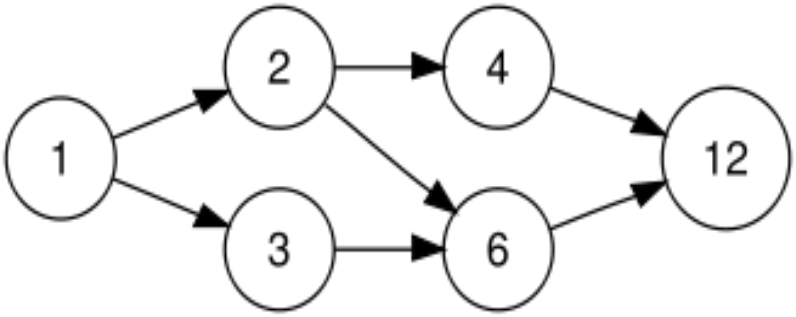


Figure 6.4. Representation of binary relation

2. Presentation of a binary tree shown in Figure 6.5.

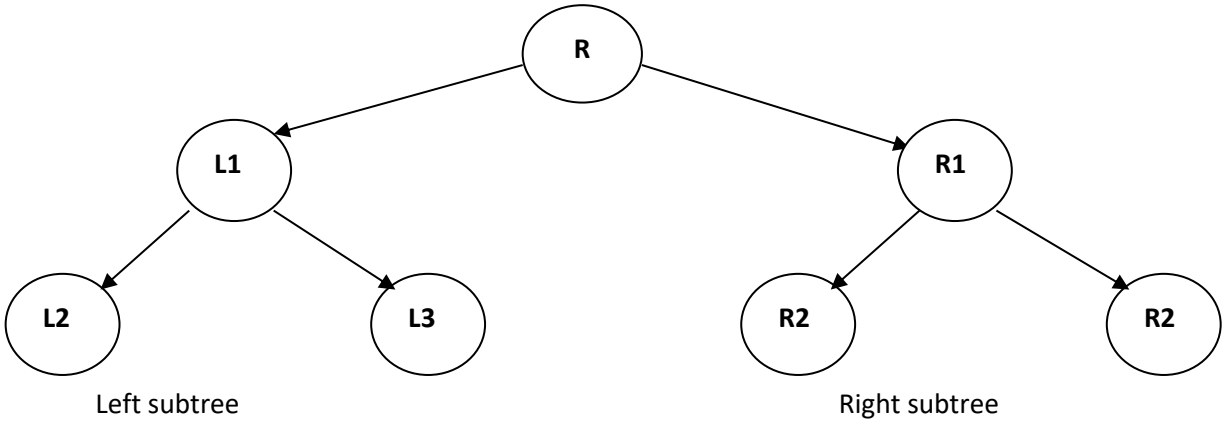


Figure 6.5. A binary tree.

2. Bypass of binary tree of arithmetic expression

((3 + 1) * 3 / (9-5) 2 + (3 * (7-4) 6)

from the top to the bottom and from the left to the right is shown in Figure 6.6.

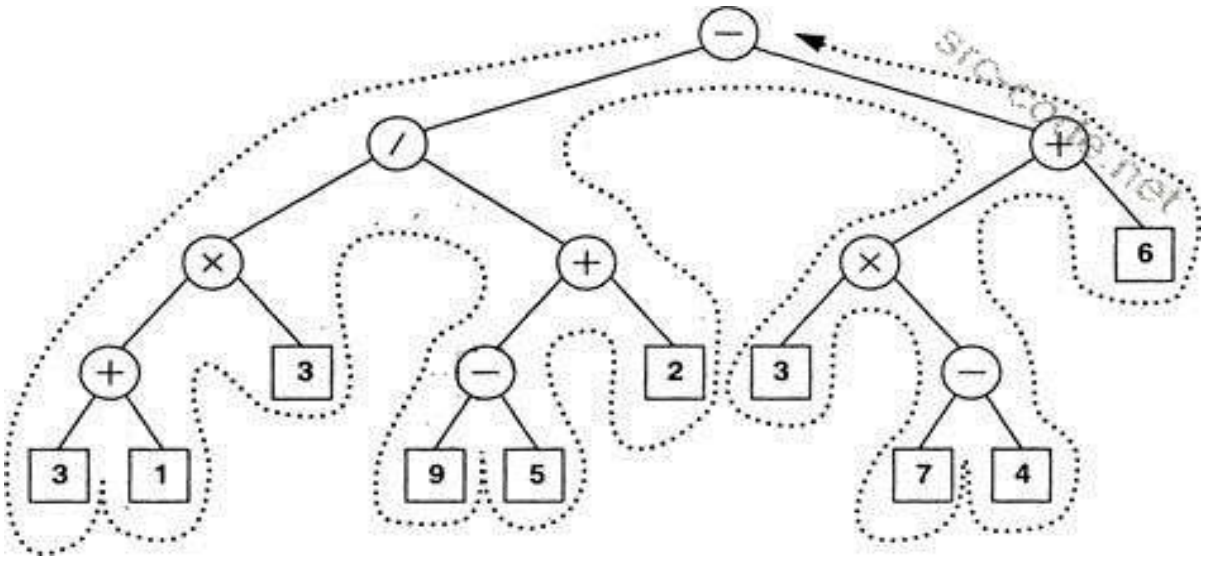


Figure 6.6. Bypass of tree

Exercises 6.1:

1. Build a directed weighted graph for describing the structure of identifier.
2. Build the tree for the expression ((a / (b + c)) + (x * (y - z))).
3. Determine the adjacency matrix A of an undirected graph that contains a loop around the vertex one, which depending on the application element a_{11} may be considered equal to one (as shown below), or to two.

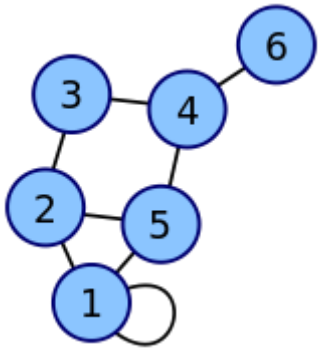


Figure 6.1. Undirected graph

Help:

1. Without loss of generality, to facilitate the construction of the desired graph we will consider not letters, but only one letter not numbers, only one number, which will serve as weight for required weighted graph.
2. In the corresponding binary tree, leaves are operands, and other vertices are operations.
3. The adjacency matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Exercises 6.2:

On a finite set N = {1, 2, 3, 4, 5} is given binary relation.
R = {(1,2), (1,4), (1,5), (2,3), (3,2), (3,4), (4,4), (4,5) , (5,3), (5,4)}.
Record domain and the range of values for this relation. Draw a graph of this relation. Make up adjacency and incidence matrix for it.

Question 6:

1. How path is formed in the graph?
2. What edges are called multiple?
3. What vertex is called an isolated?
4. What is the level of the isolated vertices?
5. What means the level of vertex?
6. What graph is called a cyclic?
7. What is the incidence matrix?

Test 6:

1. What are the types of graphs?

A) directed graph, undirected graph;
B) directed graph, defined graph;
C) specified graph, undirected graph;
D) specified graph, unsepcified graph;
E) unspecified graph, undirected graph.

2. What is a tree?

A) graph without loops and cycles;
B) graph without weights;
C) graph without networks and cycles;
D) weighted graph;
E) directed graph.

3. What is a binary tree?

A) tree in which each vertex has at most two descendant;
B) tree, which has two vertices;
C) tree, which has no cycle;
D) tree, which has no loop;
E) tree, in which one vertex has no direct descendants.

7. Finite automatons

Lecture objective: explain the concept of universal automaton and finite automaton.

Lecture plan: study the composition and structure of abstract automaton; give a formal definition of indeterminate and determinate finite automaton and of languages recognized by such automatons.

Usually under the term “automaton” we understand a device which, once turned on, can perform a number of given operations on its own. However, we deal with an abstract automaton used as a mathematical model of any digital (discrete) devices in which all signals are quantized in level, and all actions are quantized in time.

An abstract automaton (hereinafter – automaton) can distinguish a set or transform a set into another set; it consists of a tape, a head unit and a controller device; it may also have working memory.

Tape – a linear sequence of cells, each of which can store only one symbol from a certain finite input (output) alphabet.

The tape is infinite, but at each given moment only a finite number of cells is occupied. Special markers denoting the beginning and end of the tape may occupy the boundary regions to the left and right of the occupied cell area. The marker may be just at one end of the tape or be absent altogether.

Input (output) head unit – a device which can view only one tape cell at any given moment of time. The head unit can shift one cell to the left or to the right, or remain immobile. It is generally assumed that the head unit is read-only, i.e. during the work of the automaton the symbols on the tape do not change. But it is also possible to consider automatons whose head unit both reads and writes. Thus, the head unit may perform both reading and writing operations.

Working memory – an auxiliary storage for reading and writing data. Working memory may be organized as a dynamic data structure (queue or stack).

Controlling unit – a device which governs the automaton’s behavior and has a finite internal memory for storing a finite number of states. It governs the automaton’s behavior by means of a function (relation) which describes how the states change depending on the current state and current input symbol read by the head unit, and the current information extracted from the working memory if available. The controlling unit also

determines the direction of the shift of the head unit and the information to be entered in the working memory.

The automaton is determined by the input of a finite set of states of the controlling unit, finite set of accepted input symbols, the source state and the set of final states, as well as the state transition function which, by the current state and current input symbol being its arguments, indicates all possible next states or values of this function. The work of the automaton may be conveniently described by means of its configuration. The automaton's configuration includes:

- controlling unit's state;
- contents of the input tape and the position of the input head unit;
- contents of the working memory and the position of the working head unit if available;
- contents of the output tape if available.

The automaton's configuration can be initial, current and final.

In its *initial configuration* the internal memory contains a previously entered symbol denoting the initial state of the controlling unit; the controlling unit is in the initial state; the head unit reads the leftmost input symbol on the tape; if working memory is available, it contains preconfigured initial contents.

In its *current configuration* the internal memory contains previously entered symbols of current states of the controlling unit; the controlling unit is in one of its current states; the head unit reads neither the leftmost nor the rightmost current input symbol; if working memory is available it has preconfigured current contents.

In its *final configuration* the internal memory contains previously entered symbols denoting the final states of the controlling unit; the controlling unit is in one of its final states; the head unit views the right end marker or, if the marker is not available, it leaves the input tape; if working memory is available then it satisfies certain conditions.

Prior to its inception the automaton is its initial configuration, i.e. the symbol denoting the initial state of the controlling unit is entered in the internal memory, the input chain is entered in the input tape; if working memory is available, corresponding data is entered in the memory.

The automaton uses a program consisting of a finite sequence of *steps*. Each step consists of the current (initial) and next (final) configuration.

At the step's beginning the memory reads the symbol of the current state of the controlling unit, the input tape reads the current input symbol; the information in the working memory, if available, is also read. Then, depending on the current state and read information the automaton's actions are determined:

(6) Input head unit moves to the right, left or remains in place;

(7) A new symbol is entered in the current cell of the input tape or the previous symbol is not changed;

(8) Some information, if available, is entered in the working memory;

(9) A symbol is entered in the output tape, if the tape is available.

(10) The controlling unit moves into another state and the number (symbol) of this state is entered in the internal memory.

As a result, during one step of the automaton the input head unit can move one cell to the left, right or remain in its place. As the automaton functions, the contents of the input tape cells do not change, but the contents of the output tape cells and the working tape cells can.

If the automaton views the input chain and executes a sequence of steps starting from the initial configuration and finishing in a final configuration, then it recognizes the chain.

A *language* recognized by the automaton is a set of chains that the automaton recognizes.

Examples 8.1:

4. A public pay telephone may serve as an example of automaton: it recognizes the input of a coin and enters the dial number state.

5. An ATM is an automaton: it recognizes an inserted card and enters the pin-code input state.

6. A subway ticket gate is an automaton: it recognizes a token and enters the open gate state.

Finite automatons recognize regular languages. First, formal definitions of indeterminate and determinate finite automatons are given,

then the languages they recognize are described, followed by the proof of their equivalency.

Finite automatons are among the simplest and most widespread recognizing machines. A finite automaton contains *output tape, internal memory, external memory, head unit and controlling unit.*

Finite automaton may be indeterminate or determinate, but its head unit must be one-way only and move only to the right. Their formal definitions are as follows:

Definition 8.1. *Indeterminate finite automaton (IFA)* is determined by the seven element set $M = <Q,T,I,F,\vdash,\dashv,\Delta>$ where:

Q – finite set of states of the controlling unit;

T– finite set of input symbols, $Q \cap T = \varnothing$;

I – set of initial states of the controlling unit, $I \subseteq Q$;

F– set of final states of the controlling unit indicating that the input chain is recognized, $F \subseteq Q$;

$\vdash,\dashv$– tape start and end markers $\vdash,\dashv \notin T$;

Δ–set of relations of transition $\Delta \subseteq Q \times T^* \times \vartheta(Q)$, $\vartheta(Q)$ – set of all subsets of the set Q.

The determined finite automaton (DFA) is a special case of IFA.

Definition 8.2. Finite automaton $M = <Q,T,I,F,\vdash,\dashv,\Delta>$ is called *determined,* if:

(5) The set of initial states I contains exactly one element;

(6) For each transition $<q, \tau, p> \in \Delta$ $|\tau|=1$ holds true;

(7) For each state $q \in Q$ and for each symbol $t \in T$ there exists no more than one state $p \in Q$ with an attribute $<q, t, p> \in \Delta$;

(8) Other symbols are identical to IFA.

Notes 8.1:

4. Sometimes instead of the set of relations of transition Δ taking logical values “true” or “false”, the function of transition δ is used which takes value as a symbol of the set Q, where δ: $Q \times T^* \rightarrow \vartheta(Q)$ – in the case of IFA and δ: $Q \times T^* \rightarrow Q$– in the case of DFA. From the function δ it is easy to arrive at the relation Δ by assuming

$\Delta = \{<q, \tau, \delta(q, \tau)>: \ q \in Q, \tau \in T^*\}$

5. Henceforth we shall use both relations of transition and functions of transition depending on the context without making particular mention. For any $q \in \boldsymbol{Q}, p \in \boldsymbol{Q}$ и $\tau \in \boldsymbol{T}^*$ we may use:

3)For relations of transition: $<q,\tau,\{p\}>$–for IFA, $<q,\tau,p>$–for DFA;

4)For function of transition: $\delta(q,\tau)=\{p\}$–for IFA,$\delta(q,\tau)=p$ – for DFA.

6. If we want to use the function of transition instead of the relation of transition, then in the formal definition KA it is necessary to substitute the symbol Δ with δ, and leave other symbols unchanged at their previous values, i.e. we obtain $\boldsymbol{M} = <\boldsymbol{Q}, \boldsymbol{T}, \boldsymbol{I}, \boldsymbol{F}, \vdash, \dashv, \delta>$.

The KA transition may be illustrated as a *diagram*, in which each state is denoted with a circle and transition with an arrow. An arrow from the state $q \in \boldsymbol{Q}$ to the state $p \in \boldsymbol{Q}$ denoted with a chain $\tau \in \boldsymbol{T}^*$ indicates that $<q, \tau, p>$ (or $\delta(q, \tau) = p$) is a transition within the given IFA. Each initial state may be recognized by a short arrow leading to it. Each final state is indicated with a double circle.

14. Are the following grammars equivalent?

$$S \to ab,\ S \to aKSb,\ K \to bSb,\ KS \to b,\ K \to \varepsilon$$

and

$$S \to aAb,\ A \to \varepsilon,\ A \to b,\ A \to S,\ A \to bSbS$$

15. Are the following grammars equivalent?

$$S \to aD,\ D \to bba,\ D \to baDa,\ D \to aDaDa$$

and

$$S \to aaE,\ S \to abD,\ E \to bDD,\ D \to aaEa, D \to abDa, D \to ba?$$

16. What class does the following grammar belong to?

$$S \to abba,\ S \to baa?$$

17. What class does the following grammar belong to?

$$S \to AD,\ A \to aA,\ A \to \varepsilon,\ D \to bDc,\ D \to \varepsilon$$

18. Is the grammar with the rules

$$S \to AB,\ A \to a|Aa, A \to a|Aa$$

equivalent to the grammar with the rules

$$S \to AS|SB|AB,\ A \to a,\ B \to b?$$

19. Is the grammar with the rules

$$S \to cE,\ E \to ddc, E \to dcEc, E \to cEcEc$$

equivalent to the grammar with the rules

$S \rightarrow ccA$, $S \rightarrow cdB$,$A \rightarrow dBB$,$B \rightarrow ccAc$,$B \rightarrow cdBc$,$B \rightarrow dc$?

How should one describe in unambiguous grammar a language generated by the ambiguous grammar $E \rightarrow E+E|E*E|(E)|i$?

Examples 8.2:

3. For KA $\boldsymbol{M}_1$ with one transition and parameters: $\boldsymbol{Q}=\{q,p\}$; $\boldsymbol{T}^*=\{\tau\}$, $\boldsymbol{I}=\{q\}$, $\boldsymbol{F}=\{p\}$, $\delta(q,\tau)=p$ the diagram is shown in the figure 8.1.

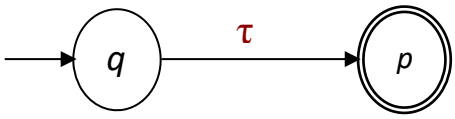


Figure 8.1. Diagram KA $\boldsymbol{M}_1$ with one transition.

4. Let KA $\boldsymbol{M}_2$ have the following parameters: $\boldsymbol{Q}=\{1,2\}$, $\mathbf{T}=\{a,b\}$, $\boldsymbol{I}=\{1\}$, $\boldsymbol{F}=\{2\}$, $\Delta=\{<1, aaa, 1>, <1, ab, 2>, <1, b, 2>, <2, \varepsilon, 1>\}$. As we can see, figure 8.2 shows a diagram of transitions of IFA $\boldsymbol{M}_2$, in which regular expressions *aaa*, *ab*, *b*,ε are used as arc markings. Such conception makes construction of the diagram easier and renders it compact and intuitive.

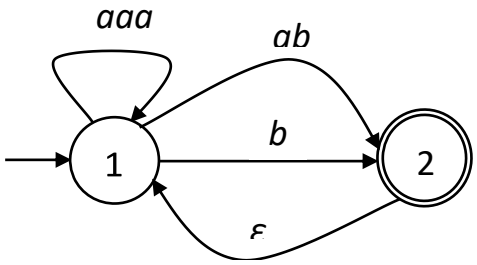
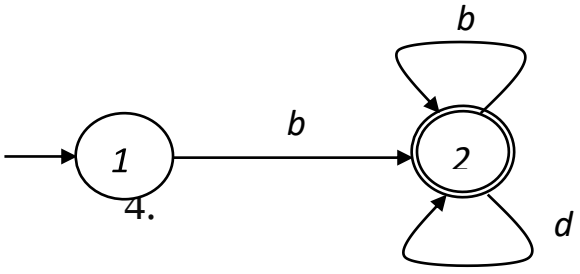


Figure 8.2. Diagram KA $\boldsymbol{M}_2$ with regular expressions



4.

Figure 8.3. Diagram KA $\boldsymbol{M}_3$ for identifier.

KA $\boldsymbol{M}_3$ for recognition of identifiers consisting only of letters and numbers and starting with a letter will have the following parameters: $\boldsymbol{Q}=\{1,2\}$, $\boldsymbol{T}=\{b,d\}$, $\boldsymbol{I}=\{1\}$, $\boldsymbol{F}=\{2\}$, $\delta(1,b)=2$,$\delta(2,b)=2$,$\delta(2,d)=2$, where *b* – letter, *d* – number. The diagram KA $\boldsymbol{M}_3$ is shown in the figure 8.3.

Note 8.3.If a diagram contains several transitions with the same starting and ending point, they are called *parallel transitions*. Usually parallel transitions are indicated in a diagram with a single arrow. The markings of transitions are separated with commas. In figure 8.4 a diagram КА $\boldsymbol{M}_4$ is shown with parallel transitions for chains *ab*, *b*.

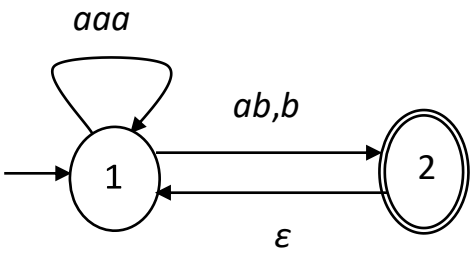


Figure 8.4. Diagram КА $\boldsymbol{M}_4$. with parallel transitions

The КА transitions may be represented as functions by means of a table or commands.

Convention 8.1.Among all KA states the initial state q_s and final state q_f stand out; here s and f are understood not as numeral variables but as mnemonic marks of start (*start*) and end (*final*).

Examples 8.3. In the table *8*.1 the function of transition δ КА $\boldsymbol{M}_5$ is shown determined by the sets $\boldsymbol{Q} = \{q_s, q_1, q_2, q_3\}$ and $\boldsymbol{T} = \{t_1, t_2, t_3\}$.

Table *8*.1. Values of the function of transition δ КА $\boldsymbol{M}_5$.

δ		Input		
		t_1	t_2	t_3
State	q_s	q_2	q_2	q_2
	q_1	q_3	q_s	q_s
	q_2	q_2	q_2	q_2
	q_3	q_3	q_2	q_s

The function of transition in the table *8*.1 may be represented as commands in the following way:

$$\delta(q_s, t_1) = q_2,\ \delta(q_s, t_2) = q_2, \delta(q_s, t_3) = q_2,$$
$$\delta(q_1, t_1) = q_3,\ \delta(q_1, t_2) = q_s, \delta(q_1, t_3) = q_s,$$
$$\delta(q_2, t_1) = q_2,\ \delta(q_2, t_2) = q_2, \delta(q_2, t_3) = q_2,$$
$$\delta(q_3, t_1) = q_3,\ \delta(q_3, t_2) = q_2, \delta(q_3, t_3) = q_s.$$

Let KA $\boldsymbol{M}$ be given with initial state $q_s \in \boldsymbol{Q}$,current state $q \in \boldsymbol{Q}$, final state $q_f \in \boldsymbol{Q}$ and unused current input chain $\tau \in \boldsymbol{T}^*$. Then the following description may be given.

Definitions 8.3:

4. If the head unit views the leftmost symbol of the input chain, then the pair $(q_s,\tau) \in \boldsymbol{Q} \times \boldsymbol{T}^*$ is called *initial configuration* KA;

5. If the head unit views the current symbol of the input chain τ, then the pair $(q,\tau) \in \boldsymbol{Q} \times \boldsymbol{T}^*$ is called *current configuration* KA;

6. If the input chain τ has been read completely, then the pair $(q_f, \varepsilon) \in \boldsymbol{Q} \times \boldsymbol{T}^*$ is called *final configuration* KA;

Note 8.4. By its contents the configuration is an "instantaneous description" of KA. Assuming that the initial chain whose belonging to the language under discussion is to be verified is in the tape, then in the configuration (q,τ) the chain τ is the part of the initial chain which remains in the tape.

The step of KA is determined by the state of the controlling unit and the input symbol being viewed at that moment. The step itself consists in the change of state of the controlling unit and the shift of the head unit one cell to the right.

The Step KA $\boldsymbol{M}$ is yielded by the binary relation $\models_{\boldsymbol{M}}$, determined over its configurations in the set $\boldsymbol{Q} \times \boldsymbol{T}^*$. If the automaton is known, then the letter $\boldsymbol{M}$ in the relation $\models_{\boldsymbol{M}}$ may be omitted.

Let $t \in \boldsymbol{T}$ be the leftmost symbol of the input chain still not read and both for $q \in \boldsymbol{Q}$ and $p \in \boldsymbol{Q}$ $<q, t, p> \in \Delta$ holds true; then for the chains $\tau \in \boldsymbol{T}^*$ the relation $(q, t\tau) \models (p, \tau)$ is true which determines the step of the automaton; this means that the automaton is in the state q and the state unit is viewing the symbol t in the input tape*;* then KA $\boldsymbol{M}$ moves into the state p and the head unit moves one cell to the right. If $\tau = \varepsilon$, then the input chain is considered to have been *read completely*.

Examples 8.4. Let $\tau = abba$. Then in the diagram KA $\boldsymbol{M_2}$ in the figure *8*.3 there is a step determined as relation $(1, abba) \models (2, ba)$.

Definition 8.4. $\models^k$ is the k–th degree of relation $\models$, if a chain of $k+1$ configurations exist

$$(q_0,\tau_0), (q_1,\tau_1), (q_2,\tau_2),\ldots, (q_{k-1},\tau_{k-1}), (q_k,\tau_k)$$

so that for any i $(1 \le i \le k)$ the relation is true

$$(q_{i-1},\tau_{i-1}) \models (q_i,\tau_i), \text{ where } q_0=q_s,\ \tau_0=\tau,\ q_k= q_f,\tau_k=\varepsilon.$$

If for any $i \geq 1$ or $i \geq 0$ $(q_0,\tau) \models^i (q_i,\varepsilon)$ holds true, then we may write $(q_0,\tau) \models^+ (q_i,\varepsilon)$ or $(q_0,\tau) \models^* (q_i,\varepsilon)$ correspondingly. Here by $\models^+$ is denoted the transitive closure of relation $\models$, and by $\models^*$ – the reflexive and transitive closure of relation.

Definition 8.5. Automaton ***M*** recognizes input chain τ, if the relation $(q_s,\tau) \models^* (q_f,\varepsilon)$ holds true.

Examples 8.5. Let $\tau = aaaab$. Then in KA $\boldsymbol{M}_2$ in the figure *8*.3 following relations $(1, aaaab) \models (1, ab)$ and $(1, ab) \models (2, \varepsilon)$ hold true.

Definition 8.6. If the language ***L*** consists only of input chains recognized by automaton ***M***, then this language is recognized by automaton ***M*** and is denoted as ***L***(***M***), i.e.

$\boldsymbol{L}(\boldsymbol{M}) \rightleftharpoons \{\tau:\ \tau \in \boldsymbol{T}^*\ \ \&\ \ (q_s,\tau) \models^* (q_f,\varepsilon)\}$.

Lemma 8.1. If $(q_1, x) \models^* (q_2,\varepsilon)$ and $(q_2, y) \models^* (q_3,\varepsilon)$ is true, then $(q_1, xy) \models^* (q_3,\varepsilon)$ is true.

Proof. For this it is necessary to perform induction by a number of steps in the program of work KA, leading from configuration (q_1,x) to configuration (q_2,ε).

Examples 8.6. Let for $\boldsymbol{M}_6 = <\{q_s,q_1,q_f\},\{0,1\},q_s,\{q_f\},\vdash,\dashv,\delta>$
there exist the following transition relations:
$<q_s,0,\{q_1\}>,<q_s,1,\{q_s\}>,<q_1,0,\{q_f\}>,<q_1,1,\{q_s\}>,<q_f,0,\{q_f\}>,<q_f,1,\{q_f\}>$

KA $\boldsymbol{M}_6$ recognizes all chains of zeroes and ones in which there are two zeroes in a row. The conditions may be interpreted in the following way:

q_s–initial condition indicates that “two zeroes in a row have not been detected and the initial symbol is a zero”;

q_1–state indicates that “two zeroes in a row have not been detected and the initial symbol is a zero”

q_f– final condition shows that “two zeroes in a row have been detected”.

It may be noted that KA $\boldsymbol{M}_6$, once entering the state q_f, remains in that state.

For the initial chain 01001 the only possible chain of configurations starting from configuration $(q_0,\ 01001)$ will be $(q_s,01001) \models (q_1,1001) \models (q_s,001) \models (q_1,01) \models (q_f,1) \models (q_f,\ \varepsilon)$.

Thus, $01001 \in \boldsymbol{L}(\boldsymbol{M}_6)$.

The diagram of this automaton is shown in the figure *8*.5.

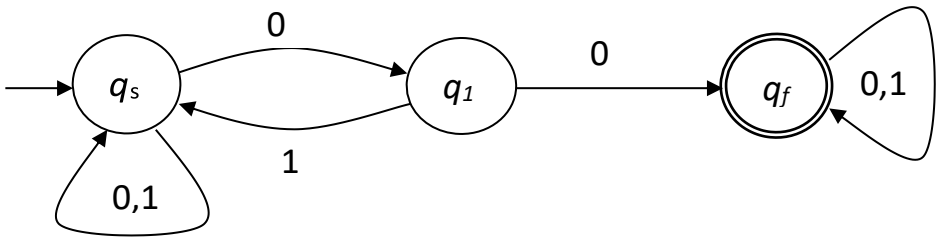


Figure II.2.1.5. Diagram КА $\boldsymbol{M}_6$.

Definitions 8.7:

3. Path КА is a tuple $<q_0, r_1, q_1, r_2,\ldots, q_n>$, where $n\geq 0$ and $r_i = <q_{i-1}, \tau_i, q_i>\in\Delta$ for each i, $1\leq i\leq n$. Here q_0 – *beginning of the path*,q_n – *end of the path*, $\tau_1...\tau_n$ – *mark of the path*, n – *length of the path*.

4. A path is called *successful* if its beginning belongs to $\boldsymbol{I}$ and its end belongs to $\boldsymbol{F}$.

Note 8.5. For any state $q\in\boldsymbol{Q}$ there exists a path$<q>$. Its mark ε, beginning and end coincide.

Examples 8.7. Let us consider КА $\boldsymbol{M_2}$in the figure *8*.3 Let τ = *baaab*. Then the path <1,<1,*b*,2>,2,<2,ε,1>,1,<1,*aaa*,1>,1,<1,*b*,2>,2> is successful. Its mark is *baaab*, and its length is 4, i.e.: $q_0=1$, $q_1=2$, $q_2=1$, $q_3=1$, $q_4=2$;
r_1=<1,*b*,2>, r_2=<2,ε,1>, r_3=<1,*aaa*,1>, r_4=<1,*b*,2>;
$\tau_1=b$, $\tau_2=\varepsilon$, $\tau_3= aaa$, $\tau_4= b$.

Using the concept “path” it is possible to give alternative definitions to already introduced concepts of recognized chain and language.

Definitions 8.8:

3. Chain $\tau\in\boldsymbol{T}^*$ *is recognized* КА $\boldsymbol{M}$, if it is the mark of a successful path.

4. КА $\boldsymbol{M}$ *recognizes* a language $\boldsymbol{L}(\boldsymbol{M})$, if it consists only of marks of all successful paths.

Note 8.6. If $\boldsymbol{I}\cap\boldsymbol{F}\neq\varnothing$, then the language recognized by КА $\boldsymbol{M}$ = <$\boldsymbol{Q}$, $\boldsymbol{T}$,⊢,⊣,$\boldsymbol{I}$, $\boldsymbol{F}$,Δ> contains an empty chain ε.

Examples 8.8. If КА $\boldsymbol{M_7}$= <$\boldsymbol{Q}$, $\boldsymbol{T}$,⊢,⊣,$\boldsymbol{I}$, $\boldsymbol{F}$,Δ> is given as $\boldsymbol{Q}$ = $\{q_1,q_2\}$, $\boldsymbol{T}$ = $\{a,b\}$, $\boldsymbol{I}$ = $\{q_1\}$, $\boldsymbol{F}$ = $\{q_1,q_2\}$, Δ = $\{<q_1,a,q_2>, <q_2,b,q_1>\}$, then it is determined and recognizes the following language:
$\boldsymbol{L}(\boldsymbol{M_7}) = \{(ab)^n:\ n\geq 0\}\cup\{(ab)^n a:\ n\geq 0\}$.

The diagram of this automaton is shown in the figure *8*.6.

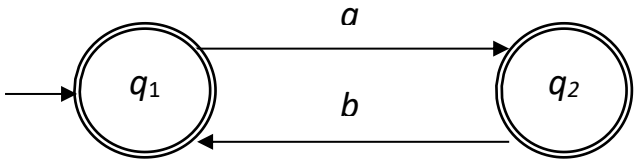


Figure 8.6. Diagram КА $\boldsymbol{M_7}$.

Definition 8.9. DFA $\boldsymbol{M} = <\boldsymbol{Q}, \boldsymbol{T}, \vdash, \dashv, \boldsymbol{I}, \boldsymbol{F}, \Delta>$, is called *full*, if for any state $q \in \boldsymbol{Q}$ and for any symbol $t \in \boldsymbol{T}$ there exists such state $p \in \boldsymbol{Q}$ that $<q, t, p> \in \Delta$, i.e. $\delta(q, t) = p$.

Examples 8.9. The diagram of full automaton $\boldsymbol{M_8}$ with the following parameters $\Delta = \{<1,a,2>, <1,b,3>, <2,a,3>, <2,b,1>, <3,a,3>, <3,b,3>\}$, $\boldsymbol{Q} = \{1,2,3\}$, $\boldsymbol{T} = \{a,b\}$, $q_s = \{1\}$, $\boldsymbol{F} = \{1,2\}$ is shown in the figure *8*.7.

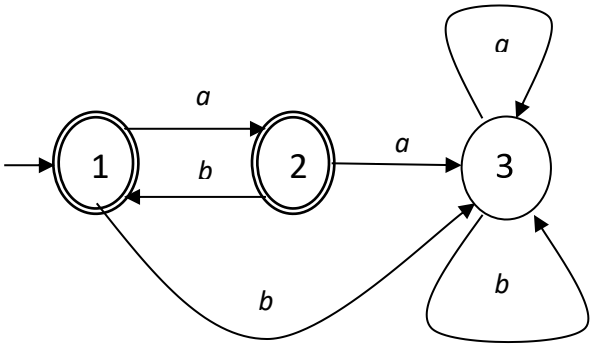


Рисунок II.2.1.7. Диаграмма КА $\boldsymbol{M_8}$

10. Find a КА recognizing language $\{\alpha\beta: \alpha \in \{a,b\}^*, \beta \in \{a,b\}^*\}$.
11. Find a КА recognizing language $\{a,b\}^* \setminus (\{a^n: n \geq 0\} \cup \{b^n: n \geq 0\})$.
12. Find a КА recognizing language $\{a\xi b: \xi \in \{a,b\}^* \cup \{b\xi a: \xi \in \{a,b\}^*\}$.
13. Find a КА recognizing language $\{\tau \in \{a,b\}^*: |\tau|_a \geq 3\}$.
14. Find a КА recognizing language $\{a^m b^n a^m b^n: m,n \geq 1\}$.
15. List all configurations (q, τ), satisfying the condition $(1, abaacdcc) \models^* (q, \tau)$, in КА$\boldsymbol{M_9}$ shown in the figure *8*.8.

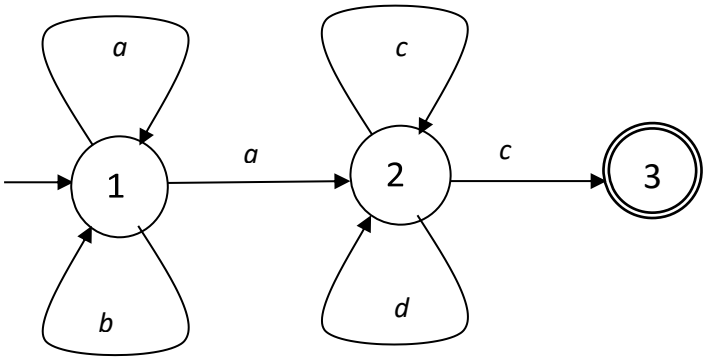


Figure 8.8. Diagram КА $\boldsymbol{M}_9$.

16. Find the step of the automaton if it is determined as

$\boldsymbol{M} = <\{ q_0, q_1, q_2, q_f\}, \{a, b, c\}, \delta, q_0, \{q_f\}>$,

where $\delta(q_0,a)=\{ q_1,q_2\}$, $\delta(q_1,a)=\{q_1\}$, $\delta(q_1,b)=\{q_f\}$, $\delta(q_2,c)=\{q_f\}$,

$\boldsymbol{L}(\boldsymbol{M}) = \{ac\} \cup \{a^n b: n \geq 1\}$.

17. Find the full determined finite automaton for language $(a \vee b)^*(aab \vee abaa \vee abb)(a \vee b)^*$.

18. Find the full determined finite automaton for language $(b \vee c)((ab)^*c \vee (ba)^*)^*$.

10. Find the full determined finite automaton for language $(b \vee c)^*((a \vee b)^*c(b \vee a)^*)^*$.

Questions 8:

13. Is КА$\boldsymbol{M_{10}}$ shown in the figure рисунке 8.9. determined?

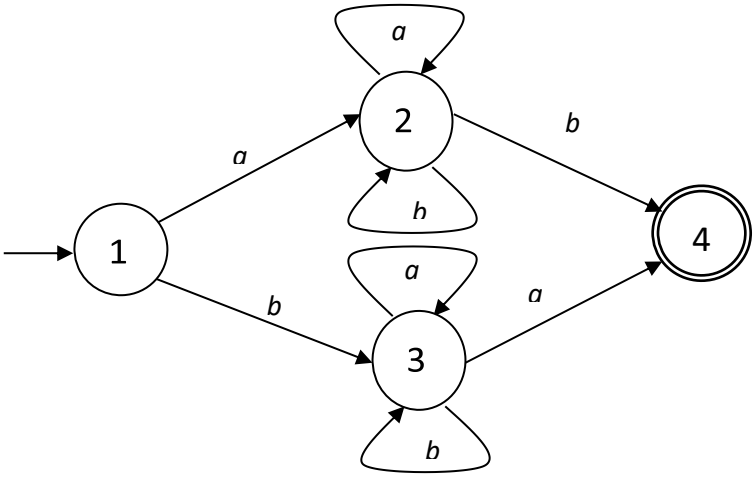


Figure 8.9. Diagram КА $\boldsymbol{M_{10}}$.

14. Do КА states q_1, q_2 and chains α,β,δ exist such that the relations $(q_1, \alpha\beta) \models^* (q_2, \beta)$ и $\neg (q_1, \alpha\delta) \models^* (q_2, \delta)$ hold true?

15. How are $|\boldsymbol{Q}|$, $|\boldsymbol{T}|$, $|\Delta|$,$|\tau|$ and the number of configurations attainable from (q,τ) related in the sense of $\models^*$?

16. What automaton can recognize the language generated by the regular expression $(abab) \vee (aba)^*$?

17. What contains the input tape?

18. What determines the direction of the shift of the head unit?

19. What does the automaton configuration consist of?

20. What types of configurations exist?

21. What does an automaton – recognized language consist of?

22. Is the determined finite automaton $\boldsymbol{M_{11}}$ with alphabet $\boldsymbol{T} = \{a, b, c\}$ shown in the figure 8.10 full?

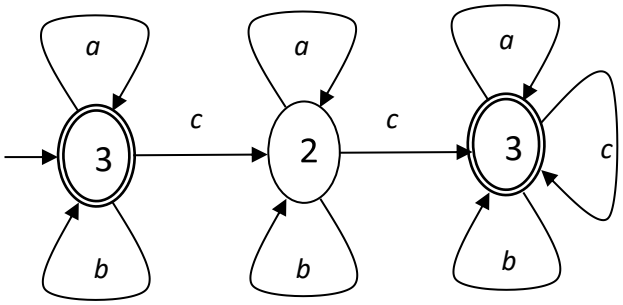


Figure 8.10. Diagram KA M_{11}.

23. Is the determined finite automaton M_{12} with alphabet $T = \{a, b\}$ shown in the figure 8.11. full?

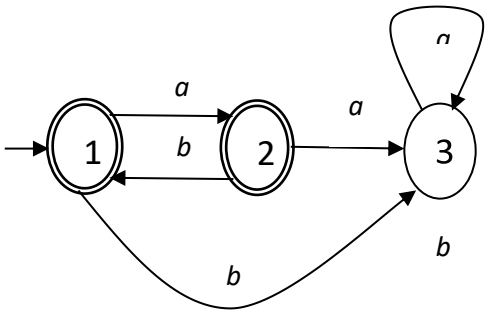


Figure 8.11Diagram KA M_{12}.

24. What does the graph of transition of finite automaton satisfying a given grammar look like?

$$G = < \{+, -, *\}, \{A, B, C\}, P, C > .$$

$$P: A \to + \mid A+$$

$$B \to - \mid B- \mid A- \mid A+$$

$$C \to * \mid C* \mid B* \mid B- \mid A+$$

Tests 8:

1. Finite automatons move to a state in accordance with:

A) transition table in the automaton's memory;

B) given task;

C) figures;

D) directions;

E) contents.

2. Which automaton is called determined?

A) if for any acceptable configuration of the identifier arising at one of the steps of its operation there exist two configurations in one of which the identifier will move in the following step;

B) if for any acceptable configuration of the identifier arising at one of the steps of its operation there exists a uniquely possible configuration in which the identifier will move in the following step;

C) if the identifier has an acceptable configuration for which there exists a finite set of configurations possible at the next step of operation;

D) if the identifier allows reading input symbols in one direction only ("from the left to the right");

E) if the identifier allows that the reading device move in both directions with respect to the chain of input symbols – both forwards from the beginning of the tape to its end and backwards going back to previously read symbols.

3.Finite automaton is a five – element set $\boldsymbol{M}$= <Q, T, δ, q_0, F>, where Q is

A) a finite set of acceptable input symbols;

B) a finite set of states;

C) transition function;

D) initial state;

E) final state.

LITERATURE

1. Turetsky V.J. Mathematics and computer science.- Moscow: Infra-M, 2000.

2. Urginovich N. The Practical work on information technologies. - Moscow: BINOM, 2004.

3. Kuk D. The computer mathematics. - Moscow: Nauka, 1990.

4. Kozlov V.N. Mathematics and computer science.- Piter, 2004.

5. Chechkin A.V. Mathematical computer science. Moscow: the Science, 1991.

6. Akimov O.E. The discrete mathematics: logic, groups, graphs. - Moscow: The Laboratory of base knowledge, 2001.

7. Novikov F.A. The discrete mathematics for programmers: the Textbook for high schools. –St-Petersburg: Piter, 2008.

8. Meyer B., Baudoin C. Programming methods. In two parts: Part 2. Translate from French Y.A. Pervina - Moscow: Mir, 1982.

9. Borubaev A.A., Pankov P.S. discrete mathematics. Kyrgyz-Russain Slavic University, Bishkek, 2010.

10.http://www.regentsprep.org/regents/math/algebra/AP1/Interval

11. http://en.wikipedia.org/wiki/Interval_(mathematics)

12. Sharipbay A.A. Informatics, Almaty, Evero, 2015, -313 p.

13. Sharipbay A.A. Theory of languages and automata, Astana, L.N. L.N. Gumilev, 2015, -207 p.

Co-funded by the
Tempus Programme
of the European Union

ШАРИПБАЙ А.А.

МАТЕМАТИКА ДЛЯ КОМПЬЮТЕРНЫХ НАУК

(Учебное пособие)

Подписано к печати 29.05.2017 г. Бумага офсетная №1.
Формат 60X84 1/16 Объем 9,5 усл.печ.л.
Тираж 50 экз. Заказ0168
Отпечатано в типографии ТОО «Мастер ПО»
010008, г. Астана, ул. Пушкина, 15-76
(ул. Кенесары, 93-76) тел.: 8 (7172) 223-418
e-mail.: master08@mail.ru

SHARIPBAY Altynbek Aminuly, Doctor of technical sciences, professor, Academician of the International Academy of Informatization, Academician of the Academy of Pedagogical Sciences of the Republic of Kazakhstan, Laureate of the State Prize of the Republic of Kazakhstan in the field of science, technology and education, Director of the Research Institute "Artificial Intelligence" and Professor of the Department of Computer Science & Information Security LN Gumilyov Eurasian National University (www.e-zerde.kz).

His scientific interests are theoretical and applied problems of informatics and information technologies: theory and technology of programming, automated systems, artificial intelligence, computer linguistics and e-learning. In this area, he developed methods for formalizing the semantics of rewriting and logical programming languages, methods for automated verification of software and hardware. Based on the results of these studies, he defended his doctoral thesis on "Verification of software and hardware of computers and systems" on the specialty "05.13.13 (05.13.11) - Computing machines, systems and networks (mathematical, software and hardware). Some of his scientific results were introduced into major scientific centers: "The translator from the language of simulation of space systems" in the Flight Test Institute, Zhukovsk; "System of verification of digital circuits" in the Research and Production Center, Zelenograd; "Parallel Programming System for Multiprocessor Computing Complex" in the Scientific Research Center of Electronic Computer Engineering, Moscow. In the field of information technology, he created various automated information systems. In particular, the "Automated Depository System of Securities" for the Department of State Property and Joint Stock Companies of Almaty; "Automated accounting system for utility payments" for the Department of Public Utilities and KSK in Almaty; "Automated system of elections"; "Database of Information Control" for the Accounts Committee of the Republic of Kazakhstan; "Automation system for the creation of electronic textbooks" for schools and universities of Kazakhstan.

He published more than 400 scientific papers, 10 textbooks and more than 10 teaching aids, 4 monographs and 5 terminology and explanatory dictionaries on computer science and computer technology, received more than 30 certificates of state registration of intellectual property, participated in the development of many states standards: 10 - in information technology, 9 - in the field of education.

Under the scientific guidance of A.Sharipbay 5 doctors and 8 candidates of sciences, 6 doctors of PhD on group of specialties "Computer science, computer facilities and management" are prepared. His scientific school created a mathematical theory of the Kazakh language, developed methods for automated analysis and synthesis of oral and written words and suggestions of the Kazakh language, proposed a technology for creating electronic educational publications, etc. These scientific results are used to create many electronic text books, a system of distance learning of the Kazakh language, a system for recognizing and synthesizing Kazakh speech and other automated systems, including accounting and expert systems in the public and private agencies and organizations of Kazakhstan.