I.V. Shostak, I.V. Gruzdo, M.A. Danova, I.I. Butenko

THEORY OF ALGORITHMS AND COMPUTING PROCESSES

2013

THE MINISTRY OF EDUCATION AND SCIENCE, YOUTH AND SPORTS OF UKRAINE
National Aerospace University of N.E. Zhukovsky
"Kharkov Aviation Institute"

I.V. Shostak, I.V. Gruzdo, M.A. Danova, I.I. Butenko

THEORY OF ALGORITHMS AND COMPUTING PROCESSES

The manual

Kharkov “KHAI” 2013

THE MINISTRY OF EDUCATION AND SCIENCE, YOUTH AND SPORTS OF UKRAINE
National Aerospace University of N.E. Zhukovsky
"Kharkov Aviation Institute"

I.V. Shostak, I.V. Gruzdo, M.A. Danova, I.I. Butenko

THEORY OF ALGORITHMS AND COMPUTING PROCESSES

The manual

Kharkov “KHAI” 2013

UDC 510.51
T33

Наведено основні засоби побудови алгоритмічних систем, що спрямовані на вирішення фундаментальних проблем теорії алгоритмів, а саме доведення обчислюваності та розв'язуваності. Особливу увагу приділено прикладним питанням математичної лінгвістики.

Для студентів спеціальності 6.020303 «Прикладна лінгвістика».

Reviewers: dr. techn. science, prof. E. I. Kucherenko,
dr. techn. science, prof. V. M. Levykin

T33 Theory of algorithms and computing processes [Text]: Tutorial for students studying in specialty 6.020303 - Applied Linguistics / I.V. Shostak, I.V. Gruzdo, M.A. Danova, I.I. Butenko. - Kh. : Nat. aerospace. univ. of N.E. Zhukovsky "Khark. aviat. Inst", 2013. – 80 p.

It is given basic means for algorithmic system development focused to solving fundamental problems of theory of algorithms namely proof of computability and resolvability. The applied problems of mathematical linguistics are under principal concern.

For students of speciality 6.020303 - Applied linguistics.

Il. 18. Table 8. Bibliogr.: 6 titles.

UDC 510.51

INTRODUCTION

Initially, the theory of algorithms has arisen in connection with the internal needs of theoretical mathematics, but as an independent science appeared in the 30-'40s XX-th century. Along with mathematical logic, it is the basis for the construction of the theory of computation. They form the theoretical basis for the design and use of computing devices to bad formalized objects. The first fundamental work on the theory of algorithms have been published independently in 1936 by Alan Turing, Alois Church and Emil Post. Their proposed the Turing machine, the machine Lent and Church's lambda calculus were equivalent formalisms algorithm, and formulated theses (Lent and the Church-Turing) postulated the equivalence of their proposed formal systems and the intuitive notion of an algorithm. An important development of this work was the formulation and proof of algorithmically unsolvable problems. In the 1950s, substantial contributions to the theory of algorithms have also works by Kolmogorov and Markov.

Thanks to the theory of algorithms the introduction of mathematical methods in economics, linguistics, psychology, pedagogy, and other humanities is happened. The tasks of the theory of algorithms include a formal proof of the algorithmic unsolvability of the problem, the asymptotic analysis of algorithms, classification algorithms in accordance with the complexity classes, criteria comparative quality assessment algorithms, etc. An example of one of the tasks in this area is the exact description of the algorithm implemented by the person in the process of forming and decision-making.

In modern conditions, the general computerization of all aspects of society has given a distinct theory of algorithms applied focus - is primarily algorithmic systems and algorithmic languages that are the foundation of the modern theory of programming for both individual computers and networks of different scales (local, corporate, global) and how to accurately describe the mappings implemented digital machines. Development of applied linguistics today also impossible without the use of computers, and consequently, the basis of the theory of algorithms and computational processes, especially the theory of formation languages and grammars. Typical problem for applied linguistics, solutions which are inextricably linked with the theory of algorithms and computational processes are machine perception and processing of natural language objects.

Thus, this manual is to form knowledge and skills of students speciality 6.020303 "Applied Linguistics".

CHAPTER 1. ALGORITHMIC SYSTEMS

1.1. INTUITIVE CONCEPT OF ALGORITHM. PROPERTIES OF ALGORITHMS

Intuitively the algorithm is viewed as process of consecutive output of the problem occurring in discrete time so that in each next moment of time algorithm system of objects is formed under the certain law from system of the objects available in the previous moment of time. Strictly speaking, intuitively such as the concept of algorithm is similar to concept of set it is impossible to define it mathematically strictly.

It is assumed that the word "algorithm" derives from the name of Central Asian (Uzbek) mathematician of the IX century Al Khwarizmi (Abu Abdulla Mohammed ebne Musa al Khwarizmi al Medgusi) — «Algorithmi» in the Latin transcription, who was the first to formulate calculating rules (procedure) for four arithmetic actions in a decimal notation.

As long as the calculations were simple, there was no need in algorithms. As a need in repeated step-by-step procedures appeared the theory of algorithms arose. However, when problem became even more complicated it showed that a part from them couldn’t be solved by algorithms. For example, these are the problems that a human solves based on thinking. The solution of such problems is based on neuromathematical methods. In this case, processes of training, tests and mistakes are realized.

Characteristics of algorithm are defined by its properties (characteristics). Basic properties of algorithm are as follows:

1. Massivness. It is supposed, that the algorithm can satisfy the solution of all problems of the given type. For example, the algorithm for the solution of system of the linear algebraic equations should be applicable to the system consisting of any number of the equations.

2. Productivity. This property means, that the algorithm should lead to result reception for finite number of steps.

3. Definiteness. The instructions entering into algorithm should be exact and clear. This characteristic provides unambiguity of result of computing process at the set initial data.

4. Discretness. This property means that process described by algorithm and algorithm can be broken into separate elementary stages which possibility of performance on the computer at the user cause no doubts.

It can seem that any problems is subjected to algorithms. It turns out that many problems cannot be solved algorithmically. Such problems are called algorithmically unsolvable.

To prove algorithmic solvability or unsolvability of problems mathematically strict and exact means are requeired. In the mid-thirties of the last century it was offered attempts to formalize concept of algorithm have been undertaken

and various models of algorithms: recursive functions, Turing and Post "machines", Markov normal algorithms.

Afterwards it has been ascertained these and other models to be equivalent in the sense that classes of problems solved by them coincide. This fact is called Church thesis. Now it is conventional. Formal definition for concept of algorithm has created prestates for developing the theory of algorithm even before working out of the first computers. Computer facilities progress stimulated further development of the theory of algorithms. Besides the defining the algorithmic resolvability of problems the theory of algorithms is engaged in an estimation of complexity of algorithms in sense of number of steps (time complexity) and demanded memory (spatial complexity), and also is engaged in working out effective algorithms in this sense.

For realization of some algorithms at any reasonable from the point of view of physics assumptions in performance speed of elementary steps it can demand more time, than, on modern views, Universe exists, or it is more memory cells, than the atoms making a planet the Earth.

Therefore, one more problem of the theory of algorithms is the output of a question of an exception of search of variants in combinatory algorithms. Estimation of complexity of algorithms and creation of effective algorithm is one of the major problems of the modern theory of algorithms.

Algotrithm is considered effective when its labour input (number of steps) is limited by a polynom from the characteristic size of a problem. See examples for effective and not effective algorithms.

Greedy algorithm. Let's consider the finite set X containing n elements, some family of its subsets and $X \subset 2^X$ weight function $w: X \to [0,+\infty)$.

Let

$$w(A) = \max_{B \in X} w(B),\ w(B) = \sum_{b \in B \subset X} w(b). \qquad (1)$$

The algorithm that in the specified family X chooses a subset A with the maximum value (weight) $w(A)$ is called as greedy.

The greedy algorithm is effective; the quantity of its steps is $O(n)$.

Full search. It is given finite set, $X = \{a_1,...,a_n\}$ containing elements n and predicate $X = \{a_1,...,a_n\}$ n $P(x_1,...,x_n)$. This predicate is not symmetric, i.e $P(...,x,...,y,...) \neq P(...,y,...,x,...)$. It is required to find a set of elements $(a_{i_1},...,a_{i_n})$ such that $P(a_{i_1},...,a_{i_n}) = \mathrm{T}$. The simplest decision of this problem is that all possible shifts $(a_{i_1},...,a_{i_n})$ from n elements with check of the validity of a predicate on them get over $(a_{i_1},...,a_{i_n})$. It is known, that number of shifts is equal to $n!$. Hence, labour input of such algorithm of full search is $O(n!)$. As $n!$ with growth of n grows faster than any polynom of n degree and faster, than the given 2^n, the algorithm is not effective.

1.2. FORMAL CONCEPTS OF STRICT DEFINITION OF ALGORITHMS

When working out an algorithm it is used one of three basic methods.

The first method is to bring a difficult problem to a sequence of more simple problems – such procedure is called a method of private purposes. Herewith it is assumed that simpler problems are easier to process than initial one. The output of the problem can be derived from the outputs off these simple problems. This method looks rather reasonably, but it is not always easy for transferring on a specific task. An intelligent choice of simpler problems is more likely arts or intuitions, than sciences. Moreover, there is no general set of rules for defining a class of problems that can be solved by means of such approach.

The second method of developing algorithms is known as a lifting method. In the beginning it is accepted the initial assumption or calculation for the initial output of a problem. Then as fast as possible move upward is made in the direction to better outputs. When the algorithm reaches such point from which it is impossible to move upward, the algorithm stops. Unfortunately, we cannot guarantee, that the definitive decision received by means of algorithm of lifting will be the best. Lifting methods remember some purpose and try to make everything, they can and where can to reach closer to the purpose. It makes theim a little shortsighted.

The third method is known as back workout, i.e. we begin with the purpose or the decision and move back in the direction to initial statement of a problem. Then, if these actions are reversible, we move from problem statement to the decision.

The heuristic algorithm usually finds comprehensible though not necessarily optimum decision, it is possible to realize it faster and easier than any known exact algorithm. Many of them are based on a method of the private purposes or on a lifting method. Often very good algorithms should be considered as heuristic: if we developed fast algorithm that worked on all known test problems, but we could not prove, that algorithm is correct. Such proof is not given yet, it is necessary to consider algorithm heuristic.

The instruction about sequence of algorithm actions of can be presented as a scheme — logic scheme of algorithm, matrix scheme of algorithm, algorithm flow graph.

The logic scheme of algorithm (LSA) was offered by Soviet mathematician A. Lyapunov (1911 — 1973) who was the professor of chair of mathematics in military artillery academy.

LSA is an expression consisting of symbols of operators, logic states following in a certain order, and also numbered arrows placed in a special way.

The matrix scheme of algorithm (MSA) is the square matrix which elements specify states of transfer of management from i operator of a line to j operator of a column. Lines of a matrix are numbered from the first operator to penultimate, columns — from the second to the last.

The algorithm flow graph (AFG) is a directed graph of a special kind. It

contains nodes of four types: 1) operational, designated by rectangles; 2) stateal, designated by rhombuses; 3) initial node and 4) finite node, designated by ovals. Nodes are connected with arches.

1.3. RECURSIVE FUNCTIONS

Recursion is one of the basic programming techniques. Recursive functions are the functions dependent on themselves. When we considered machines we spoke about transition functions that in the next moment of machine time depend on their values in the previous moment. This is the way automatic memory is realized.

In the theory of recursive functions that is considered historically the first formalization of concept of algorithm, it is applied numbering of words in any alphabet natural numbers (N), and any algorithm is to calculate some function at integer values of arguments.

Function is computable if there is such an algorithm, i.e. step-by-step procedure «from simple to difficult» which calculates the value of a function from input variables or gives a message that the set of values does not belong to interval on which a function is defined.

Function is semicalculatable if at t the input set which does not belong to a range of definition of function, the algorithm does not stops (goes in cycles). The theory of computability was developed by A.Church. The idea was similar to the problem of functional completeness of switching functions: to find elementary computable functions (which « are computable intuitively»), i.e. the basis and offer techniques to get from these elementary computable functions from more difficult functions for finite number of steps (as a superposition principle in the theory of switching functions). So received functions are also computable.

These elementary computable functions are:

$S(x) = x + 1$ – successor function (specifies next natural number);

$O(x) = 0$ – zero-function;

$I_m^n(x_1, x_2, ..., x_n) = x_m$ – functions-projectors that result in samples of m th of n arguments $1 \le m \le n$.

To obtain from some semicomputable functions other functions for finite number of steps it was offered operators.

The first of them is *the operator of superposition*, i.e. substitution in function of functions instead of variables. Thus dimension of function increases.

Definition 1.1. (The operator of superposition). Let's consider that n-seater function φ is obtained from m-seater function f and n-seater functions $g_1, ..., g_m$ by means of the operator of superposition if for all $x_1, ..., x_n$ equality is fear:

$$\varphi(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n)). \quad (2)$$

The second operator is *the operator of primitive recursion*.

Definition 1.2. Let's consider that $(n+1)$-seater function φ is obtained

from n-seater function f and $(n+2)$-seater function g by means of the operator of primitive recursion if for any $x_1,...,x_n,y$ equalities are fear:

$$\begin{aligned}&\varphi(x_1,...,x_n,0)=f(x_1,...,x_n);\\&\varphi(x_1,...,x_n,y+1)=g(x_1,...,x_n,y,\varphi(x_1,...,x_n,y)).\end{aligned}\qquad(3)$$

The third operator is the operator of minimization – operator μ.

Definition 1.3. (The operator of minimization). Let's consider, that n-seater function φ is obtained from $(n+1)$-seater functions f_1 and f_2 by means of the operator of minimization, or the operator of the least number if for $x_1,...,x_n,y$ equality $\varphi(x_1,...,x_n)=y$ is executed if and only if values $f_i(x_1,...,x_n,0),...,f_i(x_1,...,x_n,y-1)$ $(i=1,2)$ are defined, pairwise unequal $f_1(x_1,...,x_n,0)\neq f_2(x_1,...,x_n,0),...,f_1(x_1,...,x_n,y-1)\neq f_2(x_1,...,x_n,y-1)$ and $f_1(x_1,...,x_n,y)=f_2(x_1,...,x_n,y)$.

Shortly speaking, the value $\varphi(x_1,...,x_n)$ is equal to the least value of argument y at which last equality is executed.

Let's consider an example of the task of Fibinacci numbers 1,1, 2, 3, 5, 8, 13, 21... using the operator of primitive recursion:

$$\begin{cases}f(0)=1;\\f(1)=1;\\f(n+2)=f(n)+f(n+1).\end{cases}\qquad(4)$$

Here are indicated two initial values of function f (0), f (1) and a principle of forming the subsequent value. Unlike machine transition functions it is indicated not machine time, but a step of calculations n, i.e. value of function on a step that is distinct from zero and first one, is equal to the sum of function values in two previous steps.

Then,

$f(0)=1$ $f(1)=1$ $f(2)=f(0)+f(1)=1+1=2$ $f(3)=f(1)+f(2)=1+2=3$,

$f(4)=f(2)+f(3)=2+3=5\ldots$

Are all functions primitive recursive? One can show, that a set of all single-seater integer functions like $N\mapsto N$, where N being a set of natural numbers is incalculatable, moreover it is true for functions like $N^n\mapsto N$. Each primitive-recursive function has the finite description, i.e. it is set by a finite word in some alphabet fixed for all functions. Set of all finite words is denumerable, that's why primitive-recursive functions form no more than a calculable subset of incalculable set of functions like $N^n\mapsto N$. However, it happens that not all computable functions can be described as primitive-recursive.

The third operator is the operator of minimization p that allows search in calculations to define the necessary value.

Definition 1.4. Function is called as primitively recursive if it can be obtained from the elementary functions O,S,I_m^n by means of finite number of operators of superposition and primitive recursion.

Example. Let's illustrate that function $s(x,y)=x+y$ can be obtained from the elementary by means of the operator of primitive recursion. The following identities are true for the function:

$$\begin{aligned} &x+0=x; \\ &x+(y+1)=(x+y)+1, \end{aligned} \tag{5}$$

that can be written as

$$\begin{aligned} &s(x,0)=x; \\ &s(x,y+1)=s(x,y)+1 \end{aligned} \tag{6}$$

or

$$\begin{aligned} &s(x,0)=I_1^1(x); \\ &s(x,y+1)=S(s(x,y)), \end{aligned} \tag{7}$$

and that is the scheme of primitive recursion, based on the elementary functions I_1^1 and S.

Definition 1.5. Function is partially recursive if it can be obtained from the elementary functions O,S,I_m^n by means of finite number of superposition, primitive recursion and operator μ. If function is defined everywhere and partially recursive it is called general recursive.

The theorem 1.1 (Kleene theorem) [4]. Any partially-recursive function $f(x_1,...,x_n)$ can be obtained with operator μ and the operator of superposition from two primitive-recursive functions [where one is fixed once and for all, and the other depends on function $f(x_1,...,x_n)$*]*. According to Kleene theorem there is such primitive-recursive function, $U(x)$ that for any partially-recursive function $f(x_1,...,x_n)$ there exists a primitive-recursive function $\theta(x_1,...,x_n,y)$ with the following property:

$$f(x_1,...,x_n)=U(\mu y[\theta(x_1,...,x_n,y)=0]). \tag{8}$$

Let's note that not always partially recursive function can be predetermined effectively to general recursive. It's clear, that any primitively recursive function is also partially recursive (and even general recursive as each primitively recursive function is defined everywhere) as for construction of partially recursive functions from the elementary it is used more means, than for construction of primitively recursive functions. At the same time, the class of partially recursive functions is wider than a class of primitively recursive functions as all primitively recursive functions are defined everywhere, and among partially recursive functions there are also functions that are not defined everywhere.

The concept of partially recursive function has appeared as exhaustive formalization of concept of computable function. At developing the axiomatic theory of statements initial formulas (axiom) and output rules got out so that the formulas received in the theory would settle all tautologies of algebra of statements. What do we aspire to in the theory of recursive functions? Why are

so chosen the elementary functions and operators for obtaining new functions? We aspire by recursive functions to settle all conceivable functions that are calculable by means of any certain procedure of mechanical character. Like Turing thesis, in the theory of recursive functions it is put forward the corresponding natural-scientific hypothesis carrying the name *of Church thesis* in the theory of recursive functions:

Numerical function if and only if is algorithmically (or machine) computable, when it is partially recursive.

Recursive functions are a basis of functional programming. An example of language of functional programming is LISP language developed in 1960 by D.Makkarti. It is one of the first languages of data processing in the symbolical form (LISP, from LISt Processing — processing of lists). One of the most essential properties of LISP language is that data, programs and even language are simply lists of symbols in brackets. The similar structure allows writing programs or the subroutines, capable to address to themselves.

Prefix form is used in LISP:

$$\begin{aligned} &(+xy) = x + y; \\ &(*x(+xy)) = x * (x + y); \\ &(+(*xx)(*yy)) = x^2 + y^2. \end{aligned} \tag{9}$$

There are also recursions in languages of structural programming.

1.4. TURING ALGORITHMIC CONCEPT. TURING COMPUTABILITY

Turing thesis (the basic hypothesis in the theory of algorithms) [3].

Let's return to intuitive representation about algorithms. Let’s remind that one of algorithm properties is that it represents the common way allowing for each problem from a certain infinite set of problems to find its output for finite number of step. One can look at concept of algorithm from a different point of view. From infinite set of problems, it is possible to express (code) each problem with some word of some alphabet, and the problem decision — any other word of the same alphabet. As a result, we obtain the function set on some subset of set of all words from the chosen alphabet and accepting values in set of all words of the same alphabet. To solve any problem it means to find value of this function on the word coding the given problem. To have an algorithm for solving all problems of the given class is to have the common way allowing in finite number of steps to "calculate" value of constructed function for any values of argument from its range of definition. Thus, an algorithmic problem is in essence a problem about calculation of values of the function set in some alphabet.

It is necessary to specify what it means to be able to calculate values of function. It means to calculate values of function by means of suitable Turing machine. For what functions is it possible thier Turing calculation? Numerous researches of scientists, extensive experience have shown that such class of functions is extremely wide. Each function, for which calculation of values there

is any algorithm, was computable by means of some Turing Machine. It allowed Turing to state the following hypothesis named the basic hypothesis of the theory of algorithms, or Turing thesis:

To find values of the function set in some alphabet, if and only if there is an algorithm when function is computable according to Turing, i.e. when it can be calculated by suitable Turing machine.

It means that strict mathematical concept of computable (according to Turing) function is on the substance an ideal model of the concept of algorithm taken from experience. The given thesis is neither more nor less an axiom, a postulate that is put forward on interrelations of our experience with that mathematical theory which we want to bring under this experience. Be finite the given thesis cannot be proved with mathematics methods because it has no intramathematical character (one part in the thesis — a concept of algorithm — is not an exact mathematical concept). It is put forward proceeding from experience, and experience confirms its solvency. In the same way, for example, it cannot be proved mathematical laws of mechanics; they were discovered by Newton and repeatedly confirmed by experience.

However, it is not excluded the basic possibility that Turing thesis will be denied. It is to be specified a function, which is computable by means of any algorithm, but incomputable with any Turing machine. Such possibility is improbable (it is among senses of the hypothesis): any discovered algorithm could be executed with Turing machine.

Additional indirect arguments confirming this hypothesis will be given in two subsequent paragraphs where it is considered other formalizations of intuitive concept of algorithm and is proved their equivalence with concept of Turing machine.

Turing Machine

Turing machine [1] is one of abstract models of algorithms, named after English mathematician Alan Turing (1912-1954). Turing machine includes:

1) the actuation device that can be in one of the states forming finite set $Y = \{y_1, y_2, ..., y_k\}$;

2) a tape broken into cells, in each of which can be written down one of symbols of the finite alphabet $X = \{x_1, x_2, ..., x_n\}$;

3) device of the reference to a tape is a reading out and writing down head which during each moment of time "surveys" a cell and depending on a symbol in a cell and actuation device states writes down in this cell a new symbol (it can coincide with former, read out) or an empty symbol (the former symbol is erased and the blank symbol registers in its place). Further, the reference device moves on a cell to the left or to the right or remains on a place. Thus, the actuation device passes in new inwardness or remains in an old (current) state. Among control means are initial states y_1 and finite states y_k (k — a mnemonic sign on the termination of work). Turing machine is in initial state of work, in finite — after the work is terminated.

Memory of Turing machine is a finite set of states (internal memory) and a tape (external memory). The tape is infinite in both sides, however during the initial moment of time only the finite number of cells of a tape is filled by symbols, the others are empty, i.e. contain an empty symbol — a blank symbol λ.

Turing machine data are words in the tape alphabet, on a tape register initial data and result.

Elementary steps are reading and record of symbols, head shift, and transition of a control means from one position to another one (Fig.1).

Under the influence of an input symbol X, for example 1, read out by a head, the control mean forms output symbol Z, operates head movement: to the left (L), to the right (R), on a place ().

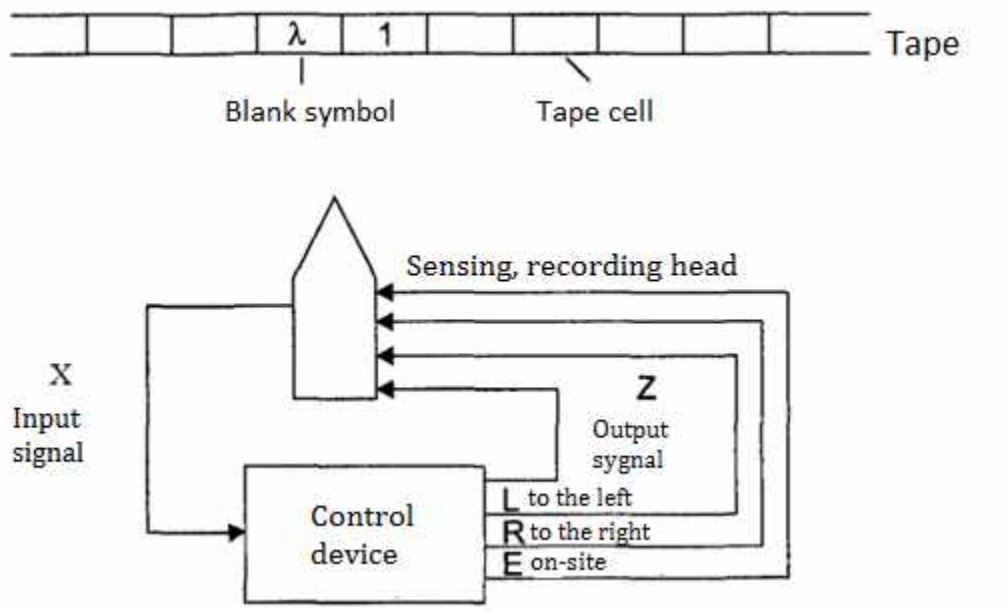


Fig. 1. Block diagram of Turing machine

The full state of the machine, or configuration (or a machine word) on which its further behavior is unequivocally defined, is described by inwardness, y_i, symbols to the left and to the right of a head, for example: the $...x_1 y_i x_2...$ System of commands of the machine contains kind records $y_1 x_1 \to \frac{y_2}{zR}$ where – x_1 a readable symbol in a state y_1; y_2 - new inwardness; z – a written down symbol; R – a sign of advancement of a head, – $\to$ a transition symbol in a new state.

Set of all commands is called a program. Each Turing machine is defined by the alphabet, states of internal memory and the program. Completely to define machine work, it is necessary to specify its configuration for the initial moment of time. Let's consider that in an initial configuration the head perceives the most left nonempty cell.

Turing machine is set by a three of data – M = <X, Y>, where X – the alphabet of symbols of a tape with the allocated empty symbol λ (blank); Y –

the alphabet of inwardness with the allocated symbols initial, y_1 and finite states y_k; P – the program, i.e. finite sequence ordered fives of symbols — commands.

If the machine, begun work with some word that has been written down on a tape, comes to a finite state it is called applicable to this word. The result of its work considers the word that has been written down on a tape in a finite state. If the machine during any moment of time does not come, to a finite state, it is called inapplicable to the given word and the result of its work is not defined.

Computability in Turing functions

Function is called computable according to Turing if there is Turing machine calculating it, i.e. such Turing machine which calculates its values for those sets of values of arguments for which function is defined, and working eternally if function for the given set of values of arguments is not defined [1,3].

Partial numerical n-seater function $f(x_1,...,x_n)$ is called computable according to Turing if there is the machine calculating it in following sense.

1. If the set of values of arguments belongs to a function range of definition f the machine, begun work in some configuration setting value of arguments, stops, finishing work in a configuration corresponding to value of function.

2. If set of values of arguments does not belong to a range of definition of functions the machine, begun work in some configuration, works infinitely, not coming in a finite state.

1.5. MARKOV NORMAL ALGORITHM. MARKOV COMPUTABILITY

The theory of normal algorithms has been developed by Soviet mathematician A.A.Markov (1903–1979) in the late 1940 - beginning of 1950th of XX century. These algorithms represent some rules on processing of words in any alphabet so the initial given and required results for algorithms are words in some alphabet.

Markov substitutions. Alphabet (as before) is called any nonempty set. Its elements are called letters, and any sequences of letters – words in the given alphabet. For convenience of reasoning mere words (they do not incorporate any letter) are supposed. Let's designate an empty word $\wedge$. If A and B are two alphabets, and A $\subseteq$ B the alphabet B is called expansion of the alphabet A.

Let's designate words with Latin letters: P, Q, R (or the same letters with indexes). One word can be a component of another word. Then the first is called subword of the second word or occurrence in the second one. For example, if A is the alphabet of Russian letters let's consider such words: P1 = paragraph, P2 = graph, P3 = ra. Word P2 is subword of word P1, and P3 is a subword of P1 and P2, and word P1 it enters twice. The first occurrence is of special interest.

Definition 1.6. Markov substitution [3] is called operation over the words, set by means of the ordered pair of words (P, Q), consisting in the following. In

set word R they find the first occurrence of word P (if that is available) and, not changing other parts of word R, replace in it this occurrence by word Q. The received word is called as result of application of Markov substitutions (P, Q) to word R. If first occurrence P in word R is not present (and, hence, in general it is not present any occurrence P in R) it is considered, that Markov substitution (P, Q) is inapplicable to word R.

To designate Markov substitutions (P, Q) it is used the following record

P-> Q. It is called the substitution formula (P, Q). Some substitutions (P, Q) let's name finite (the sense of the name becomes clear hardly later). For a designation of such substitutions, let's use record P->. Q, naming it a formula of finite substitution. Word P is called the left part, and Q — the right part in the substitution formula.

Normal algorithms

The arranged finite list of formulas of substitutions in the alphabet A is called the scheme (or record) of normal algorithm in A. (A point in brackets means, that it can stand in this place or not.) The given scheme defines (determines) algorithm of transformation of the words, named Markov normal algorithm. Let's give its exact definition.

Definition 1.7. Normal algorithm (Markov) in the alphabet A is called the following rule of construction of sequence Vi of words in the alphabet A, proceeding from the given word V in this alphabet. As initial word of sequence, V0 the word V is undertaken. Let for some i> 0 word Vi is constructed and process of construction of considered sequence has not finished yet. If in the scheme of normal algorithm there are no the formulas whose left parts would be in Vi then Vi+1 is equal to Vi and process of construction of sequence is considered come to the end. If in the scheme there are formulas with the left parts in Vi then instead of Vi+1 it is taken the result of Markov substitutions in the right part for the first of such formulas instead of the first occurrence of its left part in word Vi; process of construction of sequence is considered come to the end if in the given step the formula of finite substitution has been applied, and proceeding — otherwise. If process of construction of the mentioned sequence breaks, they say, that the considered normal algorithm is applicable to a word V. Last member of Inconsistency is called as result of application of normal algorithm to a word V. It is said that the normal algorithm processes V in W.

Let's write down sequence VI as follows:

$$V_0 \Rightarrow V_1 \Rightarrow V_2 \Rightarrow \ldots \Rightarrow V_{m-1} \Rightarrow V_m, \qquad (10)$$

where $V_0 = V$ and $V_m = W$.

We have defined concept of normal algorithm of alphabet A. If algorithm is set in some expansion of the alphabet A, they say, that it is a normal algorithm over A.

Normally computable functions and Markov principle of normalization

As well as Turing machine, normal algorithms do not make actually calculations: they only make transformations of words, replacing in them one letters others by the rules ordered to it. In turn, we order them such rules, which results of application we can interpret as calculation.

Definition 1.8. Function f, set on some set of words of the alphabet i, is called normally computable if there will be such expansion of the given alphabet (And $\subseteq$) and such normal algorithm in In, that each word V (in the alphabet) from a function range of definition f this algorithm processes in a word f (V [3].

The founder of the theory of normal algorithms the Soviet mathematician A. Markov has put forward the hypothesis which has received the name «Markov *principle of normalization* ». According to this principle, for a finding of values of the function set in some alphabet, in only case when there is any algorithm, when function normally вычислима.

1.6. METHODS OF ALGORITHM ESTIMATION

In the theory of algorithms the concept "algorithm" is usually specified by means of the description of "mathematical model» computer. Here two approaches depending on are possible, whether complexity of algorithm (the car, the program) or ability of the computing process proceeding according to algorithm is estimated.

The theoretical and empirical analysis of efficiency is necessary for proved use of time and memory of the computer.

In practice, it is necessary to be content with the approached decision of a problem and to be reconciled with uncertainty elements in the decision. As a rule, problems are precisely solved with finite number of the input and output data supposing finite representation only. It is possible to allocate two reasons on which are limited to the approached decision: or the problem cannot be solved precisely, or the exact decision is not necessary. The impossibility receives the exact decision can speak that:

- the information is incomplete, i.e. some elements of a problem in coincident among themselves cannot be distinguished, having only this information;
- the information approached which can appear as a result of many reasons, including errors of the computer, errors at the data transmission, the limited accuracy of representation and processing of numbers, restrictions on accuracy of measurements;
- the class of admissible algorithms is limited.

The majority of real problems should be solved, having only incomplete or approached information. Numerical procedures use arithmetic of finite accuracy and they are based on the approximation theory. It is desirable to describe numerical algorithms with the same severity, as algebraic. The concept of

calculation can concern not only numbers. The first symbolical manipulations have been connected with use of code numbers and secret writing. Since 1963, program systems are spread formula transformations. Among other examples of symbolical calculations are: work with texts, games in draughts, chess, and guo. Programs for reception of mathematical proofs concern the same group of programs.

Algebraic algorithms are realized in the program systems supposing input and output of the information in symbolical designations. It differs in simple formal descriptions, existence of proofs of correctness and asymptotical borders of performance time. Besides, it is possible to present algebraic objects precisely to computer memories; therefore, algebraic transformations can be executed without accuracy and importance loss. Accuracy at use of algebraic algorithm is often paid by big time of performance and a necessary memory size, than for its numerical analogue.

There are a number of important reasons for the analysis of algorithms. One of them is necessity of reception of estimations (or borders) for a memory size or an operating time that is required to algorithm for successful processing of concrete data. Machine time and memory are rather deficit and expensive resources. Another one is a desire to have a certain quantitative criterion for comparisons of two algorithms applying for the decision of the same problem. One more reason is a desire to have the mechanism for revealing of the most effective algorithms. Sometimes it is impossible to make accurate opinion on relative efficiency of two algorithms. One can work better on the average, for example, on casual input data; another works better on some special data. It is important to establish absolute criterion also. When we consider the problem decision optimum i.e. when our algorithm is so good, that it is impossible (irrespective from our mental faculties) to be improved considerably.

Usually it is required efficiency from algorithm. It means, that all operations which are necessary for making in algorithm, should be simple enough that they could be executed precisely and for a short time interval by means of a pencil and a paper.

Finiteness restriction is not rigid enough for practical purposes: the used algorithm should have not simply finite, but extremely finite, reasonable number of steps. Whether in real calculations the main question concerning some function consists in that "is the given function computable", and whether more likely in that "is it practically computable". I.e. whether there is a program calculating function in time that we have? It is possible to measure time demanded for calculation each value of function under the concrete program, in the assumption, that each step is made for a time unit. As a measure of computing complexity, it is to be taken calculation time.

It is not enough to prove correctness of algorithm. All of us can make errors at the proof and while translating correct algorithm in the program. Everyone can forget some special case of a problem. The smallest features of operational system can cause such action of a part of your algorithm about

which you did not suspect for some input data. The program should be checked up for a wide spectrum of admissible input data. This process can be tiresome and difficult. The analytical and experimental analyses supplement each other. The analytical analysis can be inexact if too strong simplifying assumptions are made. Only rough estimates in this case can be received. Experimental results, especially when generated data are used casually, can appear too unilateral. To receive authentic results, it is necessary to spend both analytical and experimental research there where it is possible.

As a measure of complexity of algorithms $\grave{A}$ it is considerd a functional correlating to each algorithm $\grave{A}$ some number $\mu(\grave{A})$ characterizing its bulkiness, for example: number of commands $\grave{A}$, length of record $\grave{A}$ or any other numerical parameter characterizing volume of the information, containing in $\grave{A}$. Similar functionals have already been applied for a long time in theoretic-cybernetic researches of the schemes realizing functions of logic algebra and also in calculus mathematics where capacity of the scheme, on which the multinomial is calculated, is measured by number of the arithmetic operations appearing in the scheme. Distinction consists only that in the specified works special narrow classes of functions and special ways of the description of algorithms are considered. We are interested in working out and application of similar concepts of more general situation (any computable functions, the general concepts of algorithm). The first publications, in which such measures of complexity have found applications, belong to A.A.Markov and A.N.Kolmogorov.

As a measure of complexity of calculations it is considered functional, correlating to each pair $(\grave{A},\grave{a})$ where $\grave{A}$ is algorithm, $\grave{a}$ is an individual problem from this a class of problems that the algorithm $\grave{A}$ solves, some number $\delta(\grave{A},\grave{a})$. This number characterizes complexity of work of algorithm with reference to $\grave{A}$ initial data and before delivery of corresponding result. For example, as $\delta(\grave{A},\grave{a})$ it is possible to take a number of elementary steps of which this work consists of (in other words, duration of process of calculation) or a memory size which can be necessary for realization of all calculations in the course of the given process, etc. As for each algorithm $\grave{A}$ it is defined the class of problems Ω it can solve (for example, that functions f whose value it calculates) than it is possible to consider, that in a given situation each algorithm $\grave{A}$ is characterized by function of variable $\alpha\delta_A(a) \underset{df}{=} \delta(A,a)$. In other words, a measure of complexity of work of algorithm (calculation) is the operator comparing with each algorithm $\grave{A}$ corresponding function $\delta_A(a)$.

Such approach to an estimation of complexity of calculations contained in the work of Soviet mathematician G.S.Tsejtin made it in 1959 in which complexity of work of normal algorithm was measured by the function specifying dependence of number of steps of algorithm from a word to which it is applied. Approximately at the same time and irrespective from G.S.Tsejtin

B.A.Trahtenbrot gave the similar functions measuring a memory size, necessary for recursive calculations, and named their signaling functions.

Defining some measure of complexity for algorithms (or calculations), we thereby hope to receive the convenient tool for comparison of algorithms, and for estimation of the objective difficulty inherent in various computable functions. In this connection, it is possible to note distinction in realization of such plan in dependence from, whether the measure of complexity of algorithm or a measure of complexity of its work is considered. As complexity of algorithm is measured by a real number, any two algorithms are comparable on complexity.

Usually they consider that the measure of complexity of algorithm accepts only natural value, therefore for each computable function there is an algorithm calculating it with the minimum complexity. This minimum complexity is natural for considering as complexity of the function, thereby and the set of all dependent functions is ordered on degree of complexity of these functions.

If the initial measure is the measure of complexity of calculations -, another picture turns out. Two signaling functions can appear incomparable even if to consider, as usually it is accepted, that signaling less δ_A signaling, δ_B if for all α but for, perhaps, their finite number $\delta_A(\alpha) < \delta_B(\alpha)$. Therefore, as far as some function f it is not clear a priori whether there is its better calculation. Here it is necessary to be limited in essence to weaker characteristic of complexity of the function f, namely it is obtained namely functions φ_1 (the bottom estimation) and φ_2 (the top estimation), such as:

1) There is an algorithm A_1 calculating f with signaling, not surpassing φ_2;

2) Whichever is the algorithm A calculating f its signaling it is not less φ_1.

Certainly, the closer to each other the top and bottom estimations φ_1 and φ_2 the complexity of the function f is more precisely characterized.

So, unlike the hierarchies based on complexity of algorithms, the hierarchies based on a measure of complexity of calculations, are partially ordered. It complicates their studying, but at the same time allows, apparently, is thinner to catch essence of that we intuitively understand as complexity of function evaluations.

The complexity theory depends on the concepts of algorithm put in its basis (recursive functions, Turing machine, etc.), and from the chosen measure of complexity.

Therefore a priori it is possible to assume, that at transition from one concept of algorithm to another it is necessary to build the complexity theory anew. However, the idea of modeling of one algorithm for others relieves us from it.

Let's consider estimations of complexity of algorithm with reference to Turing machines.

With each configuration k which Turing machine is applicable to, it is possible to associate the number characterizing complexity of process $M(k)$ in this or that sense.

Varying k we obtain function from it, defined on set of all configurations to which the given machine is applicable. Such type functions are called ***signaling functions.***

Machine work M is characterized with the following signaling functions:

Signaling time $S_M(k)$ is equal to to duration of process ($M(k)$ (i.e. to number of its configurations) if M it is applicable to k and $t_M(k)$ it is not defined, if M is not applicable to k**.**

Active zone of process is called the minimum part of a tape, containing active zones of all its configurations. Working (active) zone of the given configuration is called the minimum coherent part of a tape containing the surveyed cell, and all cells in which meaning letters are written down.

Signaling **capacity** $S_M(k)$ is equal to length of an active zone of process $M(k)$ if M is applicable to, k and it is not defined otherwise.

Signaling fluctuations $\omega_M(k)$ is equal to number of fluctuations (change of directions) heads in time if $t_M(k)$ M is applicable to, k and it is not defined otherwise.

Signaling a **mode** $r_M(k)$ it is equal to the maximum number of passages of a head over cell edge in time $t_M(k)$ if M is applicable to, k and it is not defined otherwise.

In particular, when as configurations k it is taken initial configurations for a word p the processes will be characterized by signaling functions. $t_M(k)\ S_M(k)\ \omega_M(k)\ r_M(k)$ Along with them type functions are also considered.

$$t_M^{(n)} = \max_{|p|=n} t_M(p), \quad \omega_M^{(n)} = \max_{|p|=n} \omega_M(p);$$

$$S_M^{(n)} = \max_{|p|=n} S_M(p), \quad r_M^{(n)} = \max_{|p|=n} r_M(p),$$

where $|p|$ is length of a word p and is also function

$$t_M^*(n) = \max_{\nu \le n} t_M(\nu), \quad \omega_M^*(n) = \max_{\nu \le n} \omega_M(\nu);$$

$$S_M^*(n) = \max_{\nu \le n} S_M(\nu), \quad r_M^*(n) = \max_{\nu \le n} r_M(\nu),$$

where ν is the longest of words p_i**.**

Machine construction gives only top estimations of the signaling. The finding of the bottom estimations is more difficult and it requires special theory.

There is a theorem showing, that in the presence of the top estimation, for any one of signaling, $t\ s\ \omega$ and r the set parameters and m (n m—number of symbols of the external alphabet, — n number of symbols of the internal alphabet) it is possible to receive the top estimations for the others.

The theorem. For any machine M and for any word p to which it is

applicable the following inequalities are fair:

$$\omega(p) \le t(p);$$
$$r(p) \le \omega(p) + 1;$$
$$s(p) \le |p| + 2n^{r(p)+1};$$
$$t(p) \le m^{s(p)} \cdot s(p) \cdot n.$$

Let's consider in more details estimations for **complexities of algorithm P and NP-classes complexities** in details.

Complexity is a way of comparison of algorithms [1]. They are compared in quantity necessary for performance of algorithm of steps (time complexity) and on memory size, necessary for algorithm work (capacitor complexity).

Complexity of algorithm reflects the expenses demanded for its work. We consider time complexity. It is function which puts to each input length the minimum time spent by algorithm on the decision of all same individual problems of this length in conformity.

From a course of the mathematical analysis it is known, that function is, $f(n)$ $O(g(n))$ if there is a constant *with* such, that $|f(n)| \le c(g(n))$ for all $n \ge 0$.

$O(n)$ – Complexity of order n, n is parameter of initial data of algorithm. – $O(n^2)$ is complexity of order n^2.

Complexity can be minimum, average and maximum.

Classes of problems P and NP

Complexity of a problem is estimated, as a rule, from the point of view of expense of time necessary for the Turing-Post machine to calculate function by means of which there is a decision of a considered problem [1,2].

Class of problems P

Let's consider a mass problem under which actually mean class P of the same problems consisting of infinite number of individual specific outputs and described by set of parameters. Each specific task $Z \in \Pi$ has its set of parameters. Concerning each problem the attention to the question is brought, the answer on which looks like "YES" or "NO". For example, we are interested, whether partially recursive functions have property X.

It is necessary, that the decision of problem P is reduced to calculation by the Turing-Post machine of some function.

We assume, that the coding system α is connected with class P that to each problem Z puts in conformity a word $\alpha(Z)$ in some alphabet. The size of problem Z is a length of word $|\alpha(Z)|$.

Let Turing machine solve T problems of class P and

$$t_{\mathrm{T}}(n) = \max_{Z, |\alpha(Z)|=n} t_{\mathrm{T}}(\alpha(Z)). \quad (11)$$

It is corresponding time complexity (in the worst case).

It is said that Turing machine T solves problem P for polynomial time, if

$$t_{\mathrm{T}}(n) = O(p(n)). \quad (12)$$

It is for some polynom $p(n)$. Otherwise, they say, that the problem dares for exponential time.

The class of problems P is called polynimial solvable if there is Turing machine T solving problems $z \in \Pi$ in polynomial time.

Set of classes of problems, solvable for polynimial time, is called a class of problems P.

Class of problems NP

Mass problem P belongs to class NP if in case of the answer "YES" for a problem $z \in \Pi$ there is a word $c(Z)$ with length $O(p(|\alpha(Z)|))$ such, that the problem $(Z, c(Z))$ belongs to P. Word $c(Z)$ is called true or guess for problem Z. It allows checking if problem Z belongs to class P.

For example, feasibility of the formula $\mathrm{A}(X_1,...,X_n)$ is checked if but for the formula it is given the specific set of variables – $X_1^0,...,X_1^0$ a guess promoting feasibility of the formula A.

1.7. ALGORITHMICALLY SOLVEBLE AND UNSOLUBLE PROBLEMS

Object of studying in the theory of algorithms is, first, algorithmic resolvability of some mass problem. Solvable problem is that for which there is an abstract model, for finite number of steps checking for any input data, whether is available the decision of the given problem.

Examples of algorithmically solvable problems are [1]:

- finding of the sum of two numbers;
- resolvability of propositional formulas, i.e. a finding the algorithm allowing for any PF for finite number of actions to solve, whether it is a tautology or not;
- equivalence of two propositional formulas, i.e. a problem of algorithm finding the equivalence or non-equivalence for any two PF, etc.

Problems are called algorithmically unsolvable if there is no single algorithm solving all individual semiproblems of these problems. It does not mean that such algorithm does not exist for some subclass of all class of problems of algorithmically unsolvable problem. For example, the problem of the decision of predicate formulas, a problem of equivalence of two predicate formulas, problems of applicability for Turing machine and Markov normal algorithms, a problem of equivalence of words in associative calculations and others are algorithmic unsolvable.

Let's remind, that the machine is called applicable to an initial word if it, having started to work with this word, comes in a finite state.

Example of the inapplicable machine — Turing machine that in the first part of commands does not have a finite state y_k. The machine M_1 applicable to word $n(M_1)$ i.e. to a code of own number, is called self-applicable. It is supposed, that Turing machine is universal, it reads a code of number (program) from a tape, deciphers it and according to it carries out necessary

actions depending on the initial configuration (data), also written down on a tape.

The machine, inapplicable to word $n(M)$ is called as not self-applicable.

Self-applicability problem (for the first time this problem is considered by domestic mathematician O.B.Lupanov is in that on set program P for the abstract machine to learn, whether it is applicable to own record P ((P)), where (P) is program or subroutine record $n(P)$.

For example, the program of the machine replacing symbols 1 for 0, and 0 for 1, is applicable to any word, in particular and to the record if we code records of programs in a binary code that is quite possible, therefore it self applicable, and the program B

B: 1) $?\{\lambda,2\}$;

2) HLT,

is applicable only to an empty word, i.e. unselfapplicable.

In the machine B if the head during the initial moment surveys a cell in which the blank λ is written down not, there will be an ineffectual stop.

The problem is in search of such algorithm that would define its self-applicability for any program.

The theorem 1.2. The self-applicability problem is algorithmically unsolvable.

Unsolvabilities become a mathematics life, and with their existence, it is necessary to be considered. From the theoretical point of view, unsolvability is not failure, but the scientific fact. The knowledge of the cores unsovlability theories of algorithms should be for the expert in the discrete mathematics the same element of scientific culture, as for the physicist — knowledge of impossibility of a perpetuum engine. If it is important to deal with a solvable problem (and this aspiration is natural to applied sciences) it is necessary to imagine two circumstances accurately. First (about it it was already spoken at discussion of a problem of a stop), absence of the general algorithm solving the given problem, does not mean, that it is impossible to become successful in each special case of this problem. Therefore, if the problem is unsolvable, it is necessary to search for it solvable special cases. Secondly, insolvability occurrence is, as a rule, result of an excessive generality of a problem (or language on which objects of a problem are described). The problem in more general statement has more chances to appear unsoluble. Except concepts of resolvability and insolvability the concept of complexity of algorithms is entered.

EXAMPLES AND PRACTICAL TASKS

1.1. Effective resolvability

Example 1

To show MNP functions $f(x,y)=x+y$.

The output

Let's make the MNR-PROGRAM of calculation $f(x,y)$ leaning on algorithm of addition 1 to x equal to y times and using the register R_3 as the counter of additions 1 to at $R_1 = x$ an initial configuration

	R_1	R_2	R_3	R_4	
	x	y	0	0	. . .

The program stops process of calculations when $R_2 = R_3 = y$ and thus in the register R_1 the number, R_1 $x + y$ collects as it is required.

The MNR-PROGRAM:

$$I_1 = J(3,2,5),\ I_2 = S(1),\ I_3 = S(3),\ I_4 = J(1,1,1).$$

Obviously, it is carried out by a command of conditional transfer I_1 to command I_5 that is absent in the program. Hence, $f(x, y) = (x + y)$ is computable.

Example 2

Prove MnR computability of function

$$g(x, y) = \begin{cases} 0, \text{ if } x \le y, \\ 1, \text{ if } x > y. \end{cases}$$

The solution

g (x, y) can be calculated on the algorithm given by the following block diagram (Fig. 2) $X' = X; Y' = Y;$

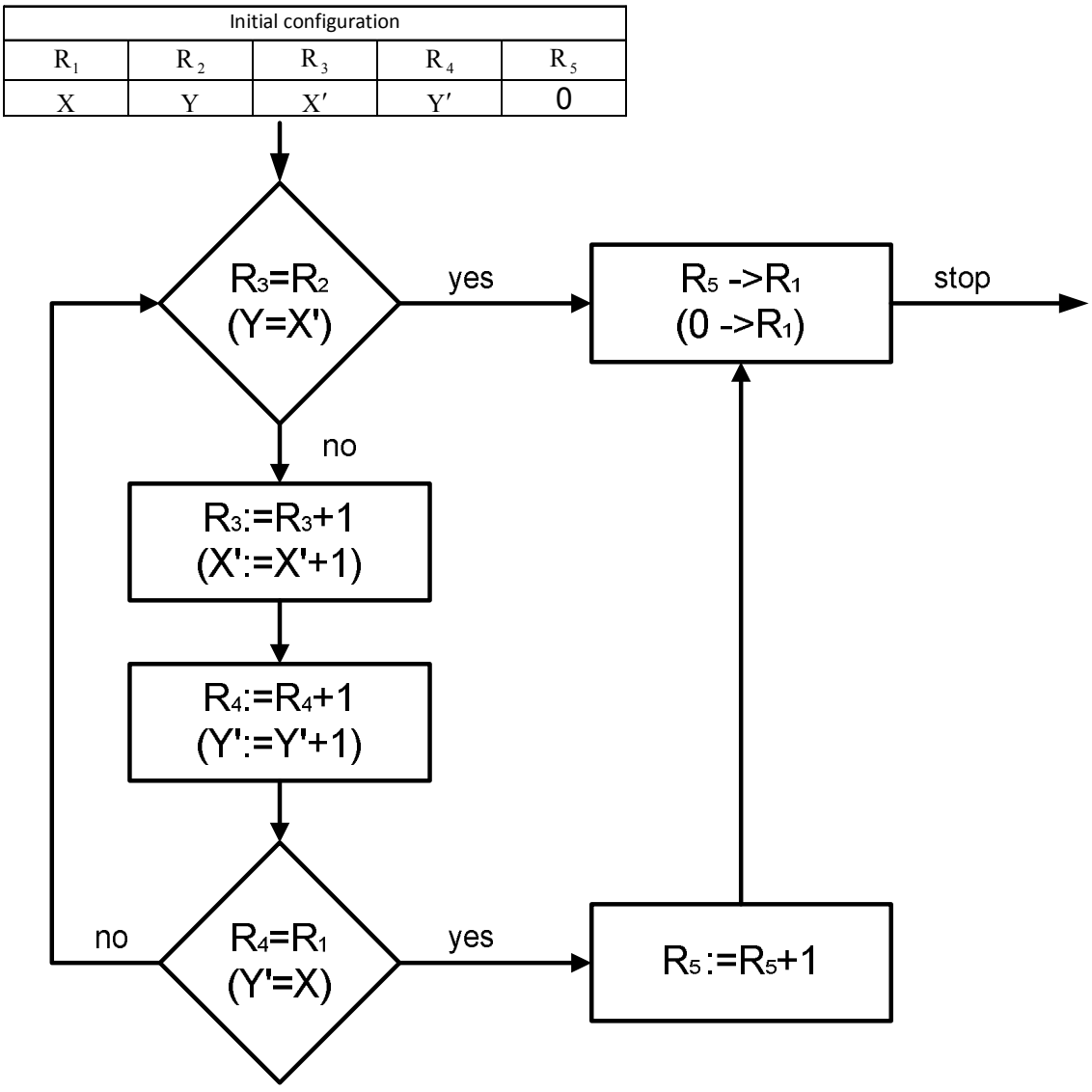


Fig. 2. Block diagram for calculation g (x, y)

Exercises

1. Show MHP – computability for the following functions:

$f(x)=10$ $\mathrm{f(x)}=\begin{cases}0,\text{if x}=0,\\1,\text{if x}\neq 0;\end{cases}$

$\mathrm{f(x,y)}=\begin{cases}0,\text{if x}=\text{y},\\1,\text{if x}\neq\text{y};\end{cases}$ $\varphi(\mathrm{x})=\begin{cases}\frac{1}{3}\mathrm{x},\text{if x multiple 3},\\\text{is\_not\_defined\_otherwise};\end{cases}$

$\mathrm{g(x,y)}=\begin{cases}0,\text{if x}\leq\text{y},\\1,\text{if x}>\text{y}.\end{cases}$

2. Prove that for each command of readdressing $T(m,n)$ at any configuration MHP there is a program, which does not contain $T(m,n)$.

3. Prove resolvability of following predicates:

$R(x,y)=(x\neq y)$; $Q(x)=(x\neq 0)$;

$R_1(x,y)=(x<y)$ ч $Q_1(x)=(x$ is even).

4. Prove that function $f(x)=x-1$ in Z is computable.

5. Prove that a predicate $Q(x)=(x\geq 0)$ in Z is solvable.

1.2. Recursive functions

1. Prove that any primitively recursive function is defined everywhere.

2. Prove that from O^1 and I_m^n using superpositions and schemes of primitive recursion it is impossible to receive functions and $x+1$ $2x$.

3. Using the operator of primitive recursion on variables x_2 and x_3 to functions $g(x_1)$ and $h(x_1,x_2,x_3)$ f obtain function $f(x_1,x_2)$. Write function $f(x_1,x_2)$ in the analytical form:

$g(x_1)=x_1$ $\quad h(x_1,x_2,x_3)=x_1+x_3$;

$g(x_1)=x_1$ $\quad h(x_1,x_2,x_3)=x_1+x_2$.

To (accept $g(x_1)=2^{x_1}$ $h(x_1,x_2,x_3)=x_3^{x_1}$ $0^0=1$);

$g(x_1)=1$ $h(x_1,x_2,x_3)=x_3(1+sg|x_1+2-2x_3|)$.

4. Apply the operator of minimization to function f in variable x_i. Resultant function is to be given in «the analytical form»:

$f(x_1)=3$ $i=1$;

$f(x_1)=[x_1/2]$ $i=1$;

$f(x_1,x_2)=x_1+x_2$ $i=2$;

$f(x_1,x_2)=I_1^{(2)}(x_1,x_2)$ $i=2$;

$f(x_1,x_2)=x_1-x_2$ $i=1,2$;

$f(x_1,x_2)=2^{x_1}(2x_2+1)$ $i=1,2$.

5. Having applied minimization operation to corresponding primitively recursive function, prove that function f is partially recursive:

$f(x_1) = x_1 / 2$;
$f(x_1) = 2 - x_1$;
$f(x_1, x_2) = x_1 - 2x_2$.

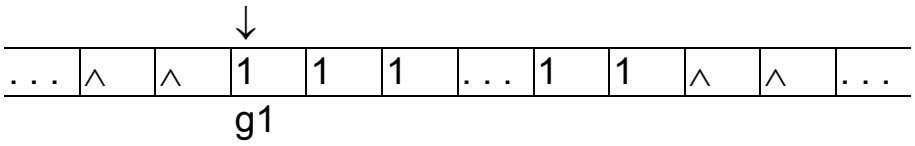
1.3. Turing machine

Example 1

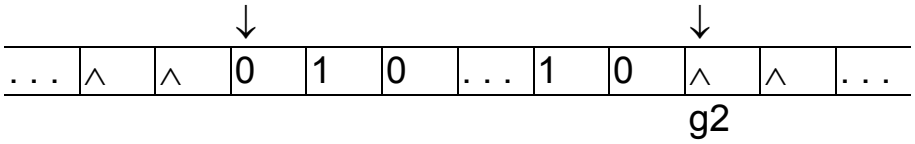
Develop *the T-machine* with the alphabet *A* = {∧0, 1} which would transform any finite word into a word of the same length, but with 0 instead of 1 standing on odd places.

The solution

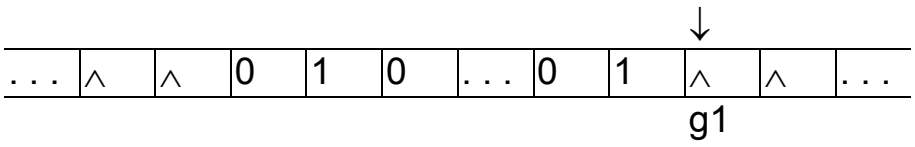
Action of the required *T-machine*, obviously, consists in movement on a tape from left to right and replacement through one symbol 1 into 0; to stop *the T-machine* should at the first survey of an empty cell, i.e. a symbol∧. These actions are provided with Q_T-program, $g_1 1 0 g_2\ g_2 0 \Pi g_2\ g_2 1 \Pi g_1$ with an initial configuration on a tape



The finite configuration on a tape looks like,



If the initial word contains even number "1", or a configuration



at odd number "1" in an initial word. *The T-machine* stops, as, surveying an empty cell in, Q_T does not find a command defining the further actions *of the T-machine*.

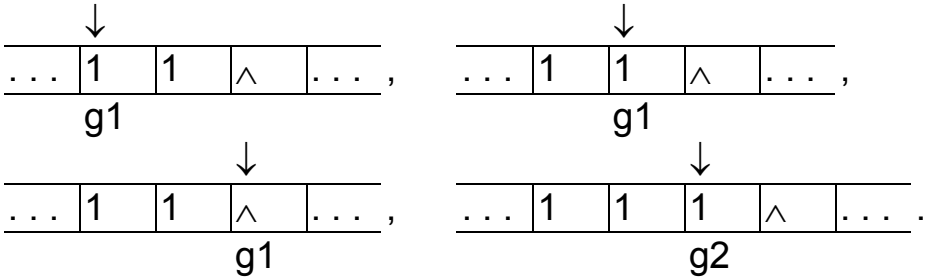
When calculating numerical functions (i.e. defined on *N* and accepting values from *N*) by Turing machines they use special coding of numbers. For example, natural number m we set a set from *m*+1 units and to designate through 1^{m+1}. Then the zero is coded 1, unit - 11, the two - 111 etc.

Example 2

Develop T_1 machine with the external alphabet *A* = {∧1}, calculating function $S(x) = x + 1$.

The solution

It is obvious two states of T_1-machine - g_1 and g_2, and program Q_T: $g_1 1 П g_1$, $g_1 Л 1 g_2$. Work T_1-machine at calculation, for example, S (1) consists of configurations:



Exercises

1. Develop Turing machine that transforms word α for word β in alphabet $\{0,1\}$:

$\alpha = 1^n \ \beta = 1^n 0 1^n$;

$\alpha = 0^n 1^n \ \beta = (01)^n$;

$\alpha = 1^n \ \beta = 1^{2n-1}$;

$\alpha = 1^n 0 1^m \ \beta = 1^m 0 1^n$.

2. Develop Turing machine that on any input chain of a kind $0^n 1^m$ defines whether equality $n = m$ is true. Whether it is possible thereof to draw a conclusion, what Turing machine can more in comparison with the finite state machine? Prove the answer.

3. Develop Turing machine that each word $x_1 x_2 ... x_{n-1} x_n$ in alphabet $\{a,b\}$ transforms to word $x_n x_{n-1} ... x_2 x_1$.

4. Calculates a symmetry predicate $S(\alpha)$: for any word $\alpha = x_1 x_2 ... x_{n-1} x_n$ in the alphabet, $\{a,b\}$ $S(\alpha) = 1$ if $x_i = x_{n-i+1}$ for all $i = 1,2,...,n$ and $S(\alpha) = 0$ otherwise.

5. Develop Turing machine that is applicable to words of a kind $1^{3n} (n \geq 1)$ and is not applicable to words of a kind $1^{3n+m} (n \geq 1; m = 1,2)$.

6. Develop Turing machine that calculates numerical function

$$I_3^{(4)}(x_1, x_2, x_3, x_4) = x_3.$$

7. Develop Turing machine that calculates numerical function

$$x - y = \begin{cases} x - y, \textbf{ if } x > y; \\ 0, \textbf{ in other case.} \end{cases}$$

8. Develop Turing machine that calculates numerical function

$$sg(x) = \begin{cases} 0, \textbf{ if } x = 0; \\ 1, \textbf{ in other case.} \end{cases}$$

9. Develop Turing machine that calculates numerical function

$$\overline{sg}(x) = \begin{cases} 1, \textbf{ if } x = 0; \\ 0, \textbf{ in other case.} \end{cases}$$

10. Develop Turing machine that calculates numerical function

$$f(x,y)=\min\{x,y\}.$$

11. Develop Turing machine that calculates numerical function

$$f(x)=2x+1.$$

12. Develop Turing machine that calculates numerical function if $f(x,y)=[x/2]=m$ if $x=2m$ or $x=2m+1, m\geq 0$.

13. Are Turing machines T_1 and T_2 which are given with commands applicable?

$$T_1: q_1 1 \to 1\ddot{I}q_1,\ q_1 0 \to 0Hq_0,\ q_1\Lambda \to \Lambda\ddot{I}q_1;$$
$$T_2: q_1 1 \to \Lambda\ddot{I}q_1,\ q_1 0 \to 1\ddot{E}q_2,\ q_1\Lambda \to \Lambda\acute{I}q_0;$$
$$q_1 1 \to \Lambda\ddot{I}q_1,\ q_2 0 \to 0\ddot{E}q_2,\ q_2\Lambda \to 0\ddot{I}q_1.$$

To words a-d?

a) 1111111;

b) 0110111;

c) 111101;

d) 001101.

1.4. Markov computability

1) Develop the graph for realization of algorithm in the alphabet, $A=\{+,\perp,?,Q\}$ set by substitutions $?\perp'\to'\perp',\ '+\perp'\to'?',\ '?Q'\to'+'$:

a) define to what kind of normal algorithms it concerns;

b) consider on it examples of deductive chains, setting an initial word not less than three symbols length.

2) Set the Markov algorithm realizing subtraction where $A-B$ values and A are B the natural numbers given in the lines, consisting of symbols 1 (for example, for word $A=4, B=3, A-B=1$ ‘ 1111—111 ’ should be processed algorithm in a word ‘).

Check up work of algorithm for cases:

a) $A=6, B=2$;

b) $A=3, B=5$.

Markov normal algorithm realizing operation of multiplication, is set by the alphabet $A=\{1,*,T,\hat{O}\}$ and sequence of supports:

$$*11\to T*1;\ *1\to T;\ 1T\to T1\hat{O};\ \hat{O}\grave{O}\to\grave{O}\hat{O};$$
$$\hat{O}1\to 1\hat{O};\ \grave{O}1\to\grave{O};\ \grave{O}\hat{O}\to\hat{O};\ \hat{O}\to 1;\ 1\to 1.$$

Develop a deductive chain from a word ‘111*1111’ to a word ‘11111111111’.

Set Markov normal algorithm realizing operation of multiplication of numbers, units given in the form of sequence (the algorithm should be distinct from algorithm of item 3). Develop a deductive chain for one of input words.

Set the normal algorithm realizing operation of comparison a sign part of

two numbers, set in a kind:

$$\vee\vee\vee 111 * \vee\vee\vee 111 \ \text{èëè} \ \vee\vee 11 * \vee\vee\vee\vee 1111;$$

$$\vee\vee\vee * 111 - \vee\vee * 111 \ \text{èëè} \ \vee\vee - 111 * \vee\vee\vee - 11;$$

$$.111 - 11 * 11 - 1111.$$

Develop corresponding deductive chains on each algorithm.

CONTROL QUESTIONS AND TASKS FOR SELF-CHECKING

1) When did the theory of algorithms arise?
2) What problems have led to occurrence of the theory of algorithms?
3) What is the subject of studying of the theory of algorithms?
4) Whose name is the concept "algorithm" related to?
5) What is a greedy algorithm?
6) What is it necessary to formalize concept of algorithm for?
7) What are the models of specification of concept "algorithm" exist?
8) What are the names of the authors of the basic types of algorithmic models?
9) What is an alphabet, word, a composition of words, a subword of a word, length of a word?
10) What methods are used when an algorithim is being developed? What words are called to be equal?
11) Why is it possible to be limited only to numerical functions, at studying of computable functions?
12) What is a function, a range of definition, and area of values?
13) What functions are partial, everywhere defined?
14) What is the functional alphabet?
15) What is a term?
16) What functions are called computable?
17) What sets are called solvable? What are their properties?
18) What are the properties of the schedule of computable function?
19) What are the properties of type and prototype of graphable sets at computable function?
20) What are concepts of Fibonacci numbers?
21) What is the idea for construction of a class of partially recursive functions?
22) What is the essence of Kleene theorem?

23) What elementary functions are included into a base set at construction of a class of partially recursive functions?

24) Turing thesis?

25) What is the essence of Turing machine?

26) What are normal algorithms intended for?

THEMES FOR INDEPENDENT WORK

1) Algebra solvable sets.
2) Algebra of countable sets.
3) Programming for RAM.
4) Computable functions on RAM.
5) Examples of algorithmically unsoluble problems.
6) The models of calculations distinct from RAM.
7) The proof of equivalence of any two various models of calculations.
8) Examples of the problems belonging to classes P and NP.
9) Examples of NP-full problems.
10) Calculations with the oracle.
11) Countable sets and computable functions.
12) Communication between countable and solvable sets.
13) Derivation trees.

CHAPTER 2. BASES OF THE FORMAL GRAMMARS THEORY

2.1. CONCEPT OF FORMAL GRAMMAR. HOMSKY HIERARCHY

In the theory of languages, principles and features of construction of various languages are considered. Prior to the beginning of the XX-th century there were only natural (spoken languages). Thus, language was understood as dialogue means between people. With linguistics development it has been established, that dialogue means are inherent not only to the person. Now language is understood as any means of dialogue.

Language includes following components:

- Sign system (set of admissible sequences of signs);
- Set of senses of this system;
- Conformity between sequences of signs and senses.

As language signs can act:

- Symbols (letters) of some alphabet (the written form of language);
- Sounds (the oral form of language);
- Gestures, colour, smells etc.

The most developed are sign systems based on symbols. The symbol is the elementary element of sign system. Designs that are more difficult are under construction of symbols. At the analysis of spoken languages the hierarchy of designs of language looks as follows:

Letter Þ word Þ sentence Þ the text.

(A sign, a symbol) (phrase)

In the theory of formal languages the formalistic approach at which the set of designs of language looks as follows is used:

Symbol Þ a line Þ the text.

(A sign, the letter) (a chain, the sentence)

In any language, it is possible to allocate correct (admissible) and wrong designs. Rules of construction of correct texts make ***syntax*** of language. The conformity description between senses and texts make ***semantics*** of language.

Semantics of language depends on an origin and character of language, i.e. from character of the objects described by language. Syntax of language depends on character of language less. Therefore, at syntax studying it is possible to use a formalistic approach.

The formalistic approach essence consists that language is considered as set of the formal objects constructed by certain rules. As formal objects sequences of symbols act. At construction of such sequences, their sense is not considered. Occurrence and formalistic approach development is connected with necessity of the decision of problems of following type:

- Machine translation from one natural language on another;
- Working out of compilers;

– Recognition of images.

Depending on an origin and degree of universality languages can be divided into the types given on Fig. 3.

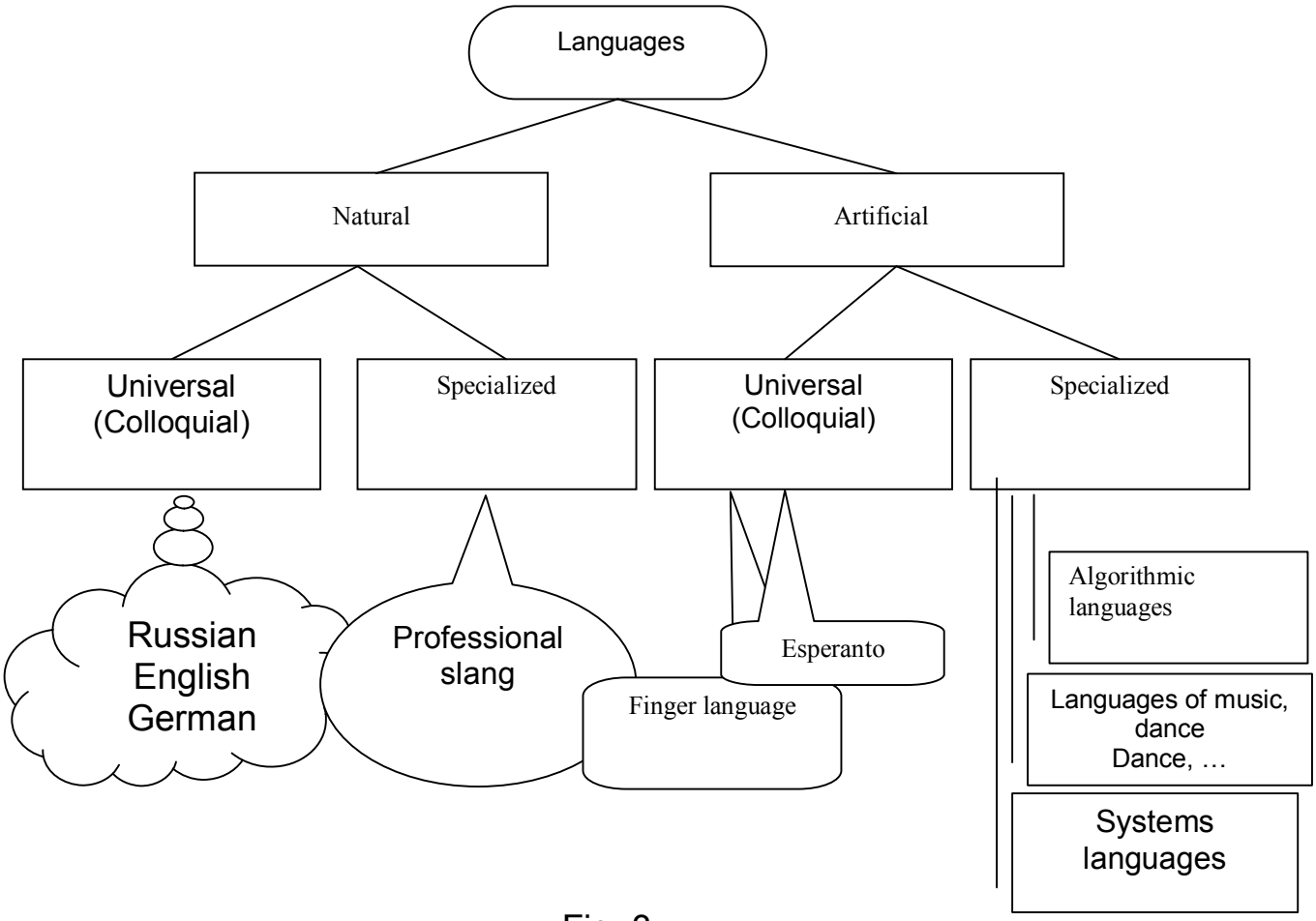


Fig. 3

Natural languages arise and develop gradually with society development for a long time.

Artificial languages are developed specially for a certain scope for rather short period.

Universal languages are used for dialogue of people in a daily life.

Specialized languages are means of dialogue enough a narrow circle of people at information interchange in some special field of knowledge. A various professional slang (language of users of the computer), algebra language, language of algebra of logic etc. can be examples of specialized languages

Formal systems are systems of operations over the objects understood as sequence of symbols. It is supposed, that between symbols there are no communications and relations except what are obviously described by means of the most formal system.

Problem. Order objects 53, 109, 3?

Most likely they will be arranged as 3, 53, 109, i.e. usual arithmetic interpretation will be given this problem: the sequence of figures is considered as the image of numbers in decimal system; streamlining of these sequences is

an arrangement of numbers represented by them on increase, and rules of comparison of such images of numbers are known so well, that anybody and does not reflect on them.

Actually, such interpretation of a problem does not follow from its text. He can be understood as a problem of lexicographic streamlining (and then the result will be 109,3,53), as a problem of distribution of runners with specified numbers on paths (which decision is connected with procedure of distribution and obviously is not connected with numerical interpretation of objects) etc.

Possibility of ambiguous extraction of problems from the specified text means, that this text does not contain formal definition of a problem. For such definition it is necessary to describe accurately a class of objects for which the problem dares and to enter obviously for them concept of streamlining, having described it as system of local operations over symbols of which these objects consist.

Historically the concept of formal system has arisen within the limits of the mathematics bases at research of a structure of axiomatic theories and proof methods in such theories.

Any exact theory is defined, first, by language, т.е some set of the statements which are making sense from the point of view of this theory, and, secondly, set of theorems – the subset of language consisting of statements, true in the given theory.

One of fundamental ideas on the mathematics bases is the idea of formalization of theories, i.e. consecutive carrying out of an axiomatic method of construction of theories. Thus, it is not supposed to use any assumptions of objects of the theory, except what are obviously expressed in the form of axioms; axioms are considered as formal sequences of symbols, and methods of proofs – as methods of reception of one the expressions from others by means of operations over symbols.

Such approach guarantees clearness of initial statements and unambiguity of outputs, the impression however is made, that intelligence and the validity in the formalized theory do not play a role. However, actually, and axioms and output rules aspire to choose so that the formal theory constructed with their help could give substantial sense.

More particularly, the formal theory is under construction as follows:

1. The set of formulas or correctly constructed expressions forming language of the theory is defined. This set is set by constructive means (as a rule, inductive definition) and, hence, it is enumerable and usually it is solvable.

2. The subset of the formulas named axioms of the theory is allocated. This subset can be and infinite, but anyway it should be solvable.

3. Rules of output of the theory are set. The output rule is a computable relation on set of formulas. Formulas are called as rule parcels, and its consequence or the output.

Obtaining formula B from formulas A1, A2... An is called such a sequence of formulas F1, F2... Fm such, that Fm=B, and any Fi is either an axiom, or one

of initial formulas A1, A2... An, or it is directly deduced from F1... Fi-1 according to one of obtaining rules.

B it is deduced from A1, A2... An if there is output B from A1, A2... An. This fact is designated A1, A2... An. A1, A2... An are called as hypotheses or output parcels.

The proof for formula B in theory T is called the output B from empty set of formulas, i.e. output in which as initial formulas axioms are used only. The formula B, for which there is a proof, is called as the formula, demonstrable in theory T or the theorem of theory T.

The fact of demonstrability of the formula B is designated by $\mapsto$ B. It is obvious that joining of formulas to hypotheses does not break deductibility. Therefore, if lB (B is demonstrable), A B (that is B it is demonstrable and with some formula A).

At studying of formal theories, there are two types of statements:

1) with statements of the theory, i.e. theorems that are considered as purely formal objects defined earlier;

2) with statements about the theory (about properties of theorems, proofs, etc.) which are formulated in language, external in relation to the theory, and are called as metatheorems.

Terminal symbols are symbols of the alphabet of nonterminal symbols form set of symbols N that is not entering in T and used on intermediate steps of generating process.

As initial symbol is called the non-terminal symbol from, which are deduced all the line long language.

Formal grammar or simply grammar in the theory of formal languages — a way of the description of formal language, that is allocation of some subset from set of all words of some finite alphabet. Distinguish generating and distinguishing (or analytical) grammar — the first set rules with which help it is possible to construct any word of language, and the second allow by given word to define, it enters into language or not [6].

The terminal (terminal symbol) — the object that is directly present at words of language, corresponding to grammar, and having concrete, unchangeable value (generalization of concept of "letter"). In the formal languages used on the computer, as terminals usually take all or a part of standard symbols ASCII — Latin letters, figures and special symbols.

Non-terminal (a non-terminal symbol) — the object designating any essence of language (for example: the formula, arithmetic expression, a command) and not having concrete symbolical value.

Generating process itself consists in application continually one of rules of transformations or production. This process transforms the set line in a new line; process comes to an end or when any of productions cannot be applied, or when the line consists of one thermal symbols.

Formal grammar G is four G = (N, T, E, P), where N – set of non-terminal symbols, T – set of terminal symbols, E – an initial symbol, P – set of productions, and $N \cap T \neq 0 \quad \alpha \to \beta, \text{‹ŠŒ } \alpha, \beta \in (N \cup T), \alpha \neq \lambda$.

Each new line in the course of a conclusion should turn out from already deduced line production application.

The sentence is the line consisting only from terminal symbols, deduced of an initial symbol.

Language L defined by grammar G, is a set of the sentences deduced in G from L: $L(G) = \left\{ \varpi \in T^* / E \overset{*}{\Rightarrow} \varpi \right\}$.

Chomsky hierarchy grammars.

Chomsky has offered to divide generating grammars in four types depending on their rules.

Type 0. Unrestricted grammars. By sight of their rules it is not imposed any restrictions. Rules look like:

$$\alpha \to \beta, \tag{13}$$

where α and β are chains of terminals and nontermanals. The chain α cannot be empty.

Type 1. Context-sensitive grammars. Rules in such grammars look like:

$\alpha A \beta \to \alpha \gamma \beta$, where α, β, γ are chains of terminals and nonterminals; A is a non-terminal symbol. Such type of rules means, that nonterminal A can be replaced by a chain γ in a context formed by chains α and β.

Type 2. Context-free grammar. Their rules look like:

$A \to \gamma$, where A is nonterminal; γ is a chain of terminals and nontermanals. Prominent feature – in the left part one corrected always nonterminal.

Type 3. Regular grammar. All rules in regular grammars have one of three forms:

$$A \to aB,\ A \to a,\ A \to \varepsilon, \tag{14}$$

where A, B – nonterminals; a – terminal; ε - empty chain.

Apparently, from the definitions, each subsequent grammar is a special case of previous one.

The languages generated by grammars of type 0-3, are correspondently called unrestricted, context-sensitive, context-free and regular languages. It is considered to be, that, for example, language for which exists context-free, but not regular grammar is context-free. As define both context-sensitive, and languages without restrictions.

Let's give examples for grammars of various types. Let’s consider the grammar generating G_6 language $L_6 = \{a^n b^n c^n \mid n \geq 0\}$:

$$G_6: \quad S \to aSBc \quad \text{(type 2)};$$
$$S \to abc \quad \text{(type 2)};$$
$$cB \to Bc \quad \text{(type 0)};$$
$$bB \to bb \quad \text{(type 1)};$$
$$S \to \varepsilon \quad \text{(type 3)}.$$

Grammar type is consided minimum of types of its rules. Hence, the grammar G_6 concerns type 0.

As example of context-free can be the grammar of arithmetic expressions. With their help, syntax of programming languages is also set. For example let's consider grammar

$$G_7: \quad S \to a \quad (1);$$
$$S \to Sa \quad (2);$$
$$S \to Sb \quad (3).$$

It is grammar of type 2 (a rule 1 – type 3, rules 2 and 3 – type 2). Let's consider language L_7: the chain a belongs to rule (1). If to correct sentence S to write a or b it will the correct sentence again. Chains of language L_7 beginning with a, further follow *a* and *b* in any order. If under *a* is meant a letter, and under *b* figure then G_7 can be considered a grammar of identifiers.

Let's design the regular grammar generating language of identifiers.

The formula (15) it is possible to leave G_7.

$$G_8: \quad S \to a. \tag{15}$$

Let's designate B a part of the identifier that can follow the first letter. Then it is possible to write down a rule (2):

$$S \to aB. \tag{16}$$

Let's write down the formula for B "Tail" can be the letter or figure:

$$B \to a, \tag{17}$$
$$B \to b. \tag{18}$$

Having written down a tail behind the letter or figure, again we will receive a correct tail:

$$B \to aB, \tag{19}$$
$$B \to bB. \tag{20}$$

The grammar G_8 is equivalent G_7.

2.2. CLASSES OF FORMAL GRAMMARS

Formal grammar differs from each other first of all in type of rules of output. Classification of formal grammars according to output rules was given by American linguist Noam Chomsky. According to Chomsky formal grammar are divided into 4 types.

The formal grammar of type 0 (unlimited grammar or grammar of any type) uses kind substitutions: $\alpha \to \beta$ where and α - β chains of any kind.

The formal grammar of type 1 or contextual grammar (context-sensitive grammar) uses kind substitutions

$$\alpha\ A\beta \to \alpha\omega\beta, \tag{21}$$

where - $\grave{A}$ a non-terminal symbol; α,ω,β - chains of any kind, thus the chain ω is not empty or $\omega \in (NUT)^+$; α,β - rule context.

The formal grammar of type 2 or context-free grammar uses substitutions, $A \to \alpha$ where α - any nonempty line, i.e $\alpha \in (NUT)^+$.

The formal grammar of type 3 or regular grammar uses substitutions

$$A \to \alpha B \ \ or \ \ A \to a,,$$

where $\grave{A}$ and $\hat{A}$ non-terminal symbols, α - terminal symbol.

In the theory of formal languages it is proved, that all regular grammars are context-free, all context-free – context-sensitive, all contextual – unrestricted.

Rules, $\grave{A} \to \varepsilon$ are called ε as-rules. It is replacement of nonterminal for an empty word. Otherwise, application ε-corrected it deletion of a corresponding non-terminal symbol is simple.

From definitions of Chomsky hierarchy it is obvious, that:

- Any grammar of a class 3 is grammar of a class 2;
- Any grammar of a class 2 without ε-corrected is grammar of a class 1;
- Any grammar of a class 1 is grammar of a class 0.

Further, we will see, that ε-rule in context-free grammar not too strongly influences set of deduced words. It is more precisely if the empty chain is not deduced from initial nonterminal of the grammar is easy for altering in equivalent without ε-rule. If the empty chain nevertheless is deduced, it is possible to alter in equivalent with the unique ε-rule $S \to \varepsilon$.

Therefore in many sources, not especially going into detail, write, that any grammar of a class 2 is grammar of a class 1. Moreover, the resulted classes of grammars form increasingly narrowed hierarchy.

It is accepted a grammar class to consider the minimum class to which it gets. For example, the grammar is considered G context-free if it is grammar of a class 2, but is not grammar of a class 3. It does not privet G to be equivalent to some regular grammar.

Let's notice, that not looking on relative stability of the resulted classification of Chomsky grammars, in some not too essential details at different authors it is possible to meet some different interpretations. We will result the short review of variants of definition of these classes:

1. In definition of context-dependent grammar (a class 1) they sometimes do not demand $\gamma \in (NUT)^+$ i.e. any chain γ replacing nontermanal A was nonempty. Any such grammar is either equivalent to any context-sensitive in former sense, or can be altered in equivalent with the same states, that in initial definition, but with the unique ε-rule $S \to \varepsilon$.

2. Definition of context-sensitive grammar is given as follows: it is the grammar in which all rules look like, $a \to \gamma$ where - a,γ any words from terminals and nonterminals, but in a it is obligatory to be nontermanals, and the length, γ is not less than length a. It is possible to prove that all such grammars are equivalent to grammars of class 1 and on the contrary.

3. Grammar of class 3 is called right regular. Together with them, they define similar concept of left-regular grammar – when all rules look like

$$A \to Ba,$$

$$A \to a,$$

$$A \to \varepsilon.$$

It is possible to prove, that any right-regular grammar is equivalent to left-regular and on the contrary. Therefore, these grammars are called simply regular.

4. It is known the concept of so-called linear grammar. Rules of output of linear grammar have one kind

$$A \to aB,$$

$$A \to Ba,$$

$$A \to a,$$

$$A \to \varepsilon.$$

Unfortunately, there exists a linear grammar not equivalent to any regular. Such grammar, for example, is G

$$S \to aB,$$

$$B \to Sb,$$

$$B \to b.$$

Generating language

$$L = \{ a^n b^n \mid n > 0 \}.$$

It is valid, easy to see, that this language is set by the grammar $H : S \to aSb, S \to ab$ which rules are deduced from data продукций. "Having designated" *B* is everything, $a^{n-1}b^n (n > 0)$ from rules we will easily receive *H* rules *G*. From here *H* and *G* are equivalent.

2.3. CONTEXT-FREE GRAMMARS

Let's pay attention to that in NC-grammar rules one symbol is replaced only, the left part corrected not necessarily consists only of this symbol: $A \to \omega$. At rules can be present and other symbols — a context: $\varphi A \psi \to \varphi \omega \psi$. Such rules mean the permission to replace a symbol on *A* only ω in a context φ and ψ. The context at this replacement corresponds without change.

The rules using a context, we call contextually connected, and the rules which are not using a context, — context-free.

The NC-grammar, context-free rules of a kind containing only, $A \to \omega$ are called *as context-free* (Ks-grammar) or context-free grammars.

The nanosecond-grammar, the containing contextually connected rules, are called as contextually connected grammars.

The languages generated Ks-grammars, are called as Ks-languages.

Let's notice, that connected by a context, or free, rules, instead of elements in a terminal chain are only.

Ks-grammar represents the important special case of NC-grammars. Their value is caused by following two circumstances:

First, refusal of a context, i.e. the requirement that in the left part of a rule one symbol was equal, does structure of grammars even more simple, that facilitates its studying;

Secondly, though in natural languages replacement of one of units with others is often admissible only in certain contexts, it is expedient to investigate possibility to describe languages, distracting from the specified fact. In natural languages situations when the phenomena which are represented essentially dependent on a context, can be described and as independent of a context are possible, i.e. In Ks-grammar terms. Thus, certainly, the description can become complicated in other relations. For example, new categories, rules or that and another can be demanded many.

In the most general lines such replacement becomes so: let there is a class of elements X in the neighborhood with elements of some class elements Y behave X differently, than in the neighborhood with elements of a class *2* so rules take place

$$YX \to YAB,$$
$$ZX \to ZCD$$

(Rules use a context).

Let's enter two new symbols X_1 and X_2. An element X in a position after Y let's desiginate through and X_1 and in a position after Z through X_2.

Then we come to the rules that do not use a context

$$X_1 \to AB,$$
$$X_2 \to CD.$$

It is not necessary to think, however, that any contextually connected NC-grammar can be replaced by Ks-grammar equivalent to it. It is known, that there are the NC-languages that are not Ks-languages, for example, language, consisting of every possible chains of a kind or $a^n b^n a^n (aba, aabbaa, ...)$ of kind chains $a^n b^n c^n$. It is impossible to refuse a context, if the rule provide shift of symbols, as shift on the being is multidimensional operation. Hence, the Ks-grammar cannot generate the language containing chains that cannot be constructed without application of shifts.

Almost all available examples of the NC-languages that are not Ks-languages have abstract character and have no interpretations in natural languages.

Till now we were engaged in introduction all new and new restrictions on classes considered grammars. At first we have demanded, that the number of symbols in the right part of rules was not less, than in left, and have received not shortening grammar. Then have demanded, that one symbol was exposed to replacement only, and have received NC-grammar. At last, we have demanded, that in the left part of a rule in general there was only one symbol, and have received Ks-grammar.

Clearly, that no further restrictions on the left parts of rules can be imposed already. Therefore, if we wish to allocate narrower classes of grammars, it is necessary to impose restrictions on the right parts.

Let's begin with number of symbols in the right part. Depending on number of symbols in the right part of rules of Ks-grammar it is possible to divide in *binary* and *nonbinary.*

Ks-grammar we will name *binary* if the right part of any rule contains no more than two symbols.

For example, rules of a kind, $A \to BC$.

Or, $A \to bB \quad A \to B$ where $A, B, C \in V_H, b \in V_T$.

Ks-grammar we will name *nonbinary* if the right part of any rule contains more than two symbols. For example, grammar with kind rules, $A \to Aab\ B \to ABC\ A \to \omega$ where, $A, B, C \in V_H, a, b \in V_T \omega$ not an empty chain in this grammar, containing more than two symbols.

Binary Ks-grammar possesses that feature that in trees of structure of components corresponding to them — S-markers — from each top proceeds no more than two branches. It means that any difficult component always consists exactly of two components directly enclosed in it.

Linear grammar — such Ks-grammar, the right which parts of rules contain no more than on one occurrence of a non-terminal symbol. Thus, for binary Ks-grammars it is kind rules

$$A \to aB,$$

where $B, A \in V_H, a \in V_T$.

For not binary Ks-grammars kind rules

$$A \to aBab, A \to acB;$$

$$A, B \in V_H, a, b, c \in V_T.$$

The language generated linear grammars, is called as linear language. Ks-grammar, the right which parts of rules contain more than one non-terminal symbol, we will call *nonlinear K&-grammars.*

The ks-grammar is called as metalinear if the right parts of its rules do not contain the purpose of grammar and all rules that left parts are distinct from the purpose, have the same appearance, as a rule linear grammar.

As example of metalinear grammar, the following grammar can serve $G=(\{a,b,c\},\{S,T\},\{S\to TT,T\to aTa,T\to bTb,T\to c\},S)$.

Language is called as metalinear if there is a metalinear grammar generating it.

Imposing restrictions on structure of symbols of the right part of rules Ks-grammars, Floyd has allocated following subclasses not binary, nonlinear grammars.

Operational grammar — the right which parts of rules cannot contain two number of standing non-terminal symbols. As examples of rules of such kind can serve

$$A\to BbC,$$
$$A\to BabC.$$

where $A,B,C\in V_H, a,b\in V_T$.

Grammar of precedencies — the right parts of rules that can contain two nearby of a standing terminal symbol. Thus there is a possibility to specify what of these terminal symbols arises in *word-formation by the* first, having large priority.

As examples of rules of such grammar the following can serve:

$$A\to BaaB,$$

where $A,B\in V_H, a\in V_T$. There are also other subclasses.

Subclass linear Ks-grammatik are unilateral linear grammar.

Unilateral linear grammar — such grammar, the right which parts of rules contain terminal symbols, only on the one hand from a non-terminal symbol.

Unilateral linear grammar are subdivided on link sided - with rules of a kind and $A\to xB$ and $A\to x$ right-hand - with rules of a kind and $A\to Bx$ $A\to x$. In both cases — A,B non-terminal symbols, and — x a nonempty chain of terminal symbols.

Unilateral *linear grammar* at which in each rule the chain consists x only of one symbol, are called as automatic grammars, or *A-grammars*, and the languages generated by these grammars, are called as automatic languages, or A-languages.

From the given definitions clearly, that each following class of grammars contains in the previous.

It is possible to present interrelation of the considered classes in a kind of the graph represented on Fig. 4. Thus, KS-grammar represent the most important subclass of NC - grammars. It speaks following four principal causes:

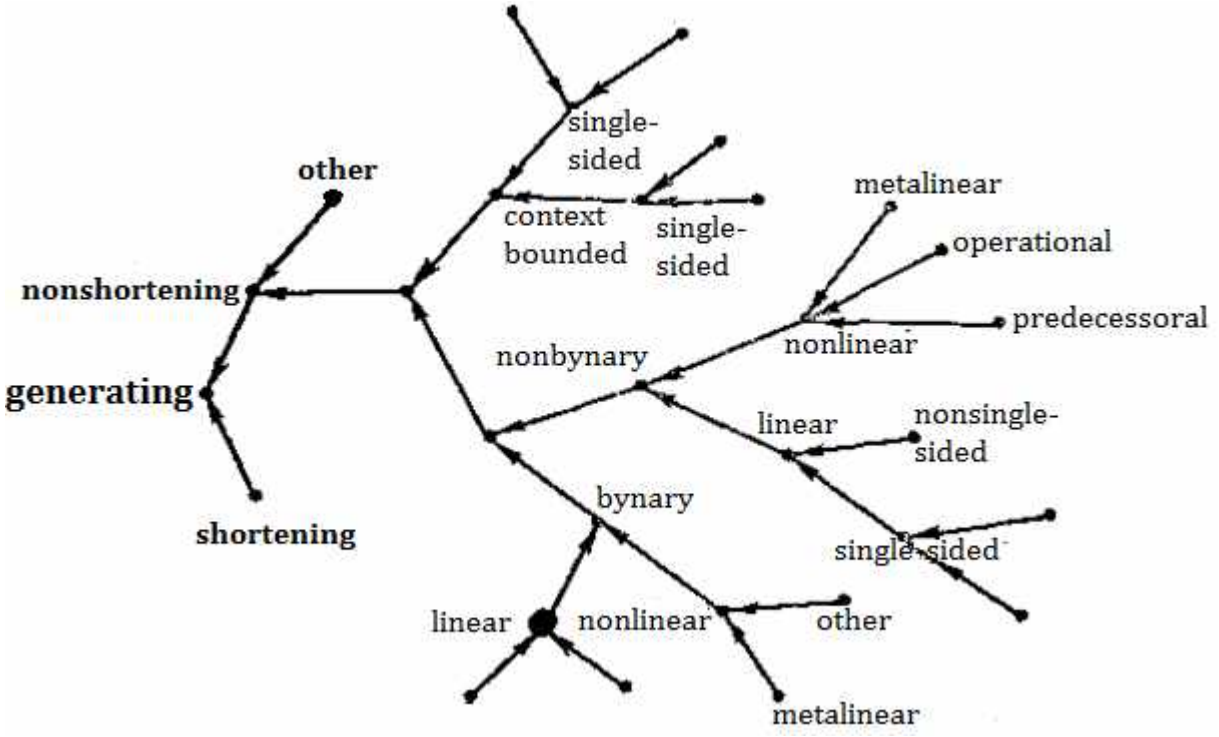


Fig. 4

1) KS-grammar are a definition basis almost all common programming languages.

2) All actions of system of parse for natural languages are based on KS-grammars.

3) It is unique type of the grammar which theory is studied and checked practically up.

4) All transfomational grammars are constructed on KS-grammars.

5) All considered types грамматик generate four types of languages: NC-language, KS-language; linear language, A-language, not considering the language generated by the most general type of grammar with unlimited rules of output by grammar of type 0.

The interrelation between the languages generated by considered types of grammar will be following:

$$L(O) \supset L(HC) \supset L(KC) \supset L(Л) \supset L(A).$$

2.4. BASES OF THE THEORY OF FORMAL LANGUAGES

2.4.1. Properties of formal languages

Let's result a number of the theorems characterizing the basic properties of languages, generated by four basic types грамматик.

The theorem 20 (Post). Any language of type *0* is recursively listed (though, probably, and not recursive) set попочек. Any recursively-countable the set of chains is type *0* language.

Owing to this theorem, the theory of languages of type *0* is covered by the general theory of recursive functions and consequently usually in the theory of languages - type *0* languages are not considered.

The theorem 2.1. (Chomsky). Type *1* languages, *2, 3* are recursive sets of chains, i.e. for each language from the specified chips there is an algorithm allowing on set grammar (that language to distinguish an accessory of any chain to language. The converse is incorrect, i.e. there are the recursive sets that are not languages. It has given the basis for carrying out of special researches of these languages.

The theorem 2.2. (Chomsky). Type *3* languages are regular sets of chains. Therefore they are sometimes called automatic.

Thus, if the class of languages of type *0* has appeared so wide, that its theory has coincided with the general theory of recursive functions (the theory of algorithms) the class of languages of type *3* has appeared, on the contrary, excessively narrow, coinciding with well studied class “regular sets” (in the theory of finite state machines). Therefore the special attention at construction of the theory of languages was given to/type languages / and *2.*

The theorem 2.3. There is the language of type *0* which is not language: type *1.* This theorem follows from the theory of recursive functions.

The theorem 2.4. There is the language of type *1* which is not language – type *2.*

As example of such language can serve $L\{a^n b^m a^n b^m c^3\}$.

The theorem 2.5. There is the language of type *2* that is not language of *type 3.*

Examples of such languages are languages

$$\{a^n b^n\} \subseteq \{xx^T\}.$$

The resulted number of theorems defines the following relation between various types of languages:

$$\{L \ \text{òèï} \ \ 3\} \subseteq \{L \ \text{òèï} \ \ 2\} \subseteq \{L \ \text{òèï} \ \ 1\} \subseteq \{L \ \text{òèï} \ \ 0\}.$$

Languages of types 0-*3* form system, their grammar represent system of rules of uniform type with consistently increasing restrictions. However, they do not settle construction possibilities of grammars of the same kind, but with other restrictions that generate the languages that distinct from are already considered.

In some works, there are described subclasses of languages for *which* interesting laws and the properties that do not have places for a class as a whole can be established.

So, in grammars type *1* the language named language of Larin which grammar has additional restrictions is considered: the kind $\varphi_1 A \varphi_2 \to \varphi_1 B \varphi_2$ i.e. rule rules including replacement of one *non-terminal* symbol by another are forbidden. Though this language is of interest for mathematical linguistics, it is studied insufficiently.

From all four types of languages the most interesting are type *2* languages — context-free.

In many respects, it is defined by possibility of their use, for research of programming languages. As is known, the program of the digital computer can be considered as a chain of symbols in some alphabet. Then some programming language represents infinite set of such chains. This language has grammar and a finite set corrected, programs defining construction.

At the same time at researches of natural languages are used so-called categorical grammars that allow selecting from among all possible sentences — correctly constructed. The class of the languages defined categorial grammars coincides with a class of the languages generated grammars of type 2.

Type 2 languages are closest to languages of regular events (type 3), with exhaustive completeness investigated in the theory of finite state machines. Besides, type 2 languages have received adequate representation by means of mathematical model of the automatic machine with store memory.

At last, languages of this type are more accessible to mathematical studying, than, for example, languages of type 1 or Parique language and consequently are most of all used. At research of widespread programming languages and in mathematical linguistics. From told becomes clear why for type 2 languages the greatest quantity of results is received and the big number of their versions is investigated, each of which is a special case of context-free language.

Let's consider one more subclass of context-free language that is defined by means of additional restrictions on system of rules.

Let there is a terminal dictionary $V_T = \{a_1,...,an\}$. We will designate through the expanded V'_T dictionary in which to each terminal symbol, a_i the symbol is compared $a'_i : V'_T = \{a_1, a'_1,..., a_n, a'_n\}$. The language generated by grammar $G = (V'_i, \{s\}, P, S)$ whose rules look like

$$P: S \to \lambda,$$

$$S \to Sa_iSa'_iS.$$

Sentences of this ω language possess following property: gels each pair of the next $a_i a'_i (i = 1,2,...,n)$ symbols containing in to allow ω to replace with an empty symbol for $\wedge$ each sentence there will be ω such sequence of these replacements, c by which help it will be reduced ω to an empty chain. Interest to Dick languages speaks that they are evidently connected with brackets structures, usual for natural and artificial languages.

Let's imagine a set of formulas of some mathematical calculation or the program that has been written down in language of type ALGOL. It will be the text in which there will be the signs always used only in pairs: the left and right brackets of all kinds (round, square, figured, broken) or the operational brackets consisting of words "beginning" and "end". We will eliminate from the

text everything, except signs on the specified type. The new text constructed by some strict rules will turn out. Similar texts give representation about languages.

2.4.2. Operations over formal languages

To languages, as well as to any sets, various operations can be applied. Before to consider operations over languages, we will define property of isolation of set. It is said that the set is closed concerning some operation, if result of its application to any element of set or to any; to steam of elements contains in this set.

For languages, the usual image defines operations of association, crossing and operation of addition concerning the fixed dictionary V_T.

Association of languages L_1 and L_2 (designation: $L_1 \cup L_2$) is called a set of all words belonging at least to one of languages.

This operation represents usual theoretically plural association; it is also comutative and associative:

$$L_1 \cup L_2 = L_2 \cup L_1, \quad (L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3). \tag{22}$$

Under the same conditions, intersection of two languages (a designation: $L_1 \cup L_2$) is called a set of all words belonging simultaneously to both languages.

This operation represents usual theoretically plural crossing; it is also comutative and associative.

Addition to language L is called the set of all words belonging to V^*_T but not belonging to L.

The set V^*_T is the language which addition to V^*_T is an empty language.

Let's underline, that the language containing an empty word, e is not empty.

Addition operation let's consider on example

$V_T = \{a, b\};$

$L = \{a^m b^n \mid m \geq 1, n \geq 1\};$

$V^*_T \setminus L = (L_1 \cup L_2 \cup L_3)$, where — L_1 set of all words beginning with, b — the set of all words beginning with and, $a^m b^n a$ $L_3 = \{a\}^*$ i.e. L_3 is set of all words consisting only from a.

The relation of languages of types *0, 1, 2, 3 to* Boolean to operations following four theorems that we furnish without the proof define.

The theorem 2.6. A class of languages of type *0* will close concerning - association and crossing operations. The problem of that definition is algorithmically unsoluble, whether is addition of language of type *0* concerning the fixed dictionary also type *0* language.

The theorem 2.7. The class of languages of type 1 closed under operations of association and intersection.

The question on belongs to what class addition of languages of type *1* in relation to the fixed dictionary, stay opened.

The theorem 2.8. A class of languages of type *2* will close concerning association operations, but will not close concerning addition operation in relation to the dictionary containing not less of two symbols, and also in relation to crossing operation.

The theorem 2.9 (*Kleene*). A class of languages of type *3* will close concerning all of boolean operations.

Except boolean operations over languages operations of multiplication, iterations, transpositions (mirror display of language) and some other are considered also.

Product of two languages (designation: $L_1 \cdot L_2$) is a set of all words that can be received in the next way is called: some word undertakes from and L_1 some word joins it on the right from, L_2 i.e.

$$L_1 L_2 = \{ X_1 X_2 \mid X_1 \in L_1, X_2 \in L_2 \}.$$

This operation (named multiplication of languages) does not coincide with the Cartesian multiplication; it is associative, but not commutative.

Let $V_T = \{a,b,c\}$. Let's consider the language $\{a\}$ consisting of one single letter word a. Then product is

$$L = V^*_T.$$

There is a set of all words beginning with a.

Iteration operation (Kleene operation). As operation of multiplication of languages is associative, we can erect the given language in 1! degree:

$$L^2 = LL, L^3 = (LL)L = L(LL),...$$

Kleene suggested to consider union

$$L^* = E \bigcup L \bigcup L^2 \bigcup L^3 \bigcup ... \bigcup L^n \bigcup ...$$

of all consecutive degrees of language L. This union is designated L^* and called iteration of language L.

For example, let's consider the alphabet

$$V = \{a,b,...\}.$$

as the language consisting of single-letter words. Then V^2 is a set of all two-letter words; V^3 is a set of all three-letter words etc. Therefore $E \bigcup V \bigcup V^2 \bigcup V^3 \bigcup ...$ is a set of all words over V i.e. V^*.

It is proved isolation of classes of regular and context-free languages concerning multiplication iteration. Language L called regular if there is a finite state machine A such, that $L = L(A)$.

There is a following theorem concerning languages 0, 1, 2, 3.

The theorem (30). Classes of languages of types 0, 1, 2, 3 are closed concerning the operation of mirror display defined as follows.

Let language $L \subset V^*_T$ to be given, through L^T it is designated a language consisting of references of all words of language L:

$$L^T = \{ X^T \mid X \in L \}.$$

This operation is involutive (i.e. it coincides with its return operation):

$$(L^T)^T = L.$$

Besides, it is connected with multiplication of languages by a following correlation:

$$(LM)^T = M^T L^T.$$

For example, language $V^*_T (V^*_T)^T$ coincides with, V^*_T as $(V^*_T)^T = V^*_T$ and $E \in V^*_T$.

Let's give a concept for operation of *homomorphism.*

Let's put in conformity to element a of dictionary V_T of finite dictionary V_{Ta}. Let's designate V^*_{Ta} through a set of all chains from the dictionary V_{Ta} and through $\tau(a)$ - any chain from V^*_{Ta}. Thus, function τ is defined on separate symbols a from dictionary V_T. Define function τ on sentences from the dictionary V_T as follows: if. $x = ai_1 ai_2 ... ai_s$ then $\tau(x) = \tau(ai_1)\tau(ai_2)...\tau(ai_s)$

Certain function τ reflects a subset of chains L from V^*_T into some subset of chains from *, $(\bigcup_a V_{Ta})^*$ i.e. $\tau(L) = (\bigcup_a V_{Ta})^*$.

Operation $\tau(L)$ of reflection for language L by means of function τ is called operation of homomorphism.

The theorem 2.10 (Bar-Hillel, Pearls and Shamir). Classes of languages of type 0, 2 and 3 are closed concerning operation homomorphism.

For languages of type 1 (contextual) the following theorem is given.

The theorem 2.11. If V_T contains at least two elements than a class of contextual languages is not closed concerning the operation of homomorphism.

Language projection. For each word in the alphabet $X \times Y$.

$$< x(1)y(1) >< x(2)y(2) > ... < x(t)y(t) >$$

its projections in X and Y are accordingly called words

$$x(1)...x(t);$$

$$y(1)...y(t).$$

In other words, if language L in the alphabet $X \times Y$ is given than a projection of language L in X is called a language consisting accurately from projections in X of words from language L.

The language cylinder. Let the alphabet Y and language L in alphabet X. Y-cylinder of language L is called language L' consisting of all words in the alphabet $X \times Y$ X whose projections $X \times Y$ X belong to L.

Properties of languages in relation to considered operations are given in Tab. 1.

Table 1

Order	Operation	Language type			
		0	1	2	3
1	Association	1	1	1	1
2	Crossing	1	1	0	1
3	Addition	0		0	1
4	Transposition (mirror display)	1	1	1	1
5	Product	1		1	1
6	Iteration	1		1	1
7	Homomorphism	1	0	1	1

In this table unit designates isolation concerning corresponding operation of a class of languages of certain type, zero — a closure failure. The empty cage means, that the question is not solved yet.

2.5. METHODS OF GRAMMARS ANALISYS

Methods of grammatical analysis can be broken into two big classes - ascending and descending - according to order of construction of a tree of grammatical analysis. Descending methods (from top to down) begin methods from the rule of grammar defining an ultimate goal of the analysis from a root of a tree of grammatical analysis and try to increase it that the subsequent knots of a tree corresponded to syntax of the analyzed sentence. Ascending methods (from below upwards) begin methods with finite knots of a tree of grammatical analysis and try to unite their construction of knots increasingly high level until the tree root will be reached.

The basic problem of the theory of languages consists in formally to analyze classes of languages for working out of possible methods as modeling, and effective processing of languages by machine means.

It is reduced to definition of logic structure of languages, i.e. Systems of the rules defining syntax of grammar. If the form of rules (i.e. the syntactic part of grammar) is precisely established, carrying out of a following number of researches on language is possible:

- communication between kinds of languages with their structural trees and form of syntactic rules;
- studying of structural properties of the languages generated by some form of rules of G-grammar;
- relative riches or poverty of various forms of grammars generating languages L.
- different sort of problems of resolvability of language L concerning the set of G-grammar and a class of G-grammars;

– capacity of G-grammar generating modeling language L depending on a context of application of the last one;

– generating ability of G-grammars, and consequently, definition of equivalence generated different grammars sets of languages L;

– measure of complexity, generated by G-grammars of sentences from language L;

– reducibility establishment of G-grammars of various complexity to more simple G-grammars;

– definition of possible methods of construction distinguishing G-grammars for set of languages L and a number of other not less important researches connected with studying of properties of a class of formal grammars with a view of their practical use.

The greatest development was received by a parse problem of grammars and the control corresponding languages, consisting from that for any chain $\omega \in L$ it is defined its formal correctness (or abnormality is defined $\omega \in L$) and syntactic analysis.

The problem of parse and the language control has arisen in connection with requirements of translation — working out the specialized program with whose help the machine translates given the program into a computer language.

Each of compilers is constructed for some concrete pair of languages: one — input and the second — output. Put simply, such compilers work by a dictionary principle: for each concrete combination of the symbols which are making sense in the given source language, this compiler gives out certain sequence of symbols of a output language.

Construction of such compilers represents rather labor-consuming work as it is necessary to consider all possible (making sense) expressions of the source language and to each of them to compare corresponding expressions on a output language.

Definition of making sense expressions of the source language is carried out because of parse of this language. Its essence consists that based on syntactic rules of grammar check of grammatical correctness set the sentence or words of language by grammatical analysis is carried out. Thus, a problem of grammatical analysis is the analysis of sentence from the point of view of an establishment of their grammatical correctness.

Grammatical analysis is understood as process of definition of a sentence structure or a word α $\alpha \in G$ according to the rules defined by G.

The establishment of that fact, that sentence or word is grammatically correct, can be executed not in one way. Grammatical analysis can be carried out in some cases more than in one way.

As a language example for which grammatical analysis can be carried out not uniquely, let's consider the language set by grammar with G rules.

$$HBC \to HBc;$$
$$HBC \to hBC;$$
$$BC \to bc;$$
$$HB \to hb.$$

Let's consider a word hbc. This word can be disassembled in two various ways. Trees of grammatical analysis of this word are given in Fig. 5.

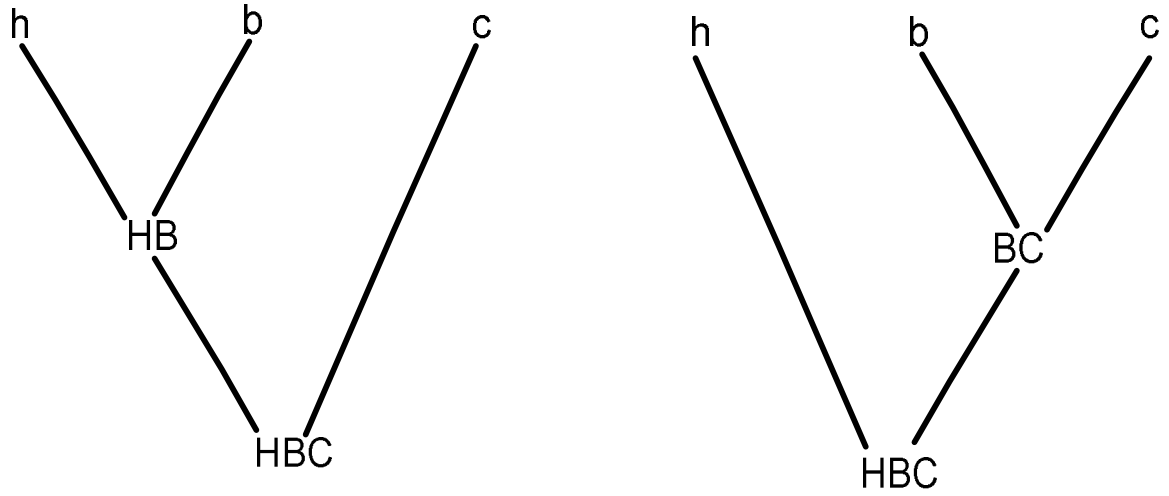


Fig. 5

In the elementary kind grammatical analysis consists in that, having begun with the first rule, to look through the list corrected from top to down, the applicable rule will not be found yet, to apply it and to repeat this process so many times, how many it is necessary. This technique in application to our rules would give the grammatical analysis shown in the right drawing.

Let's consider one more method of grammatical analysis based on an order of viewing of the assorted word or the sentence. Rules of grammar of language look like:

$$A \to BC;$$
$$B \to DE;$$
$$C \to FH.$$

It is necessary to execute grammatical analysis of word $DEFH$.

Analysis of the given word can be made *from left to right* or *from right to left.* In the first, case in a word the first set of symbols DE accessible to replacement according to the given grammars. Instead of it, symbols from some rule are substituted. After that, the received word is looked through again, beginning at the left, for the purpose of search of set of symbols for replacement by grammar rules.

For our example viewing from left to right gives the tree of grammatical analysis given in Fig. 6.

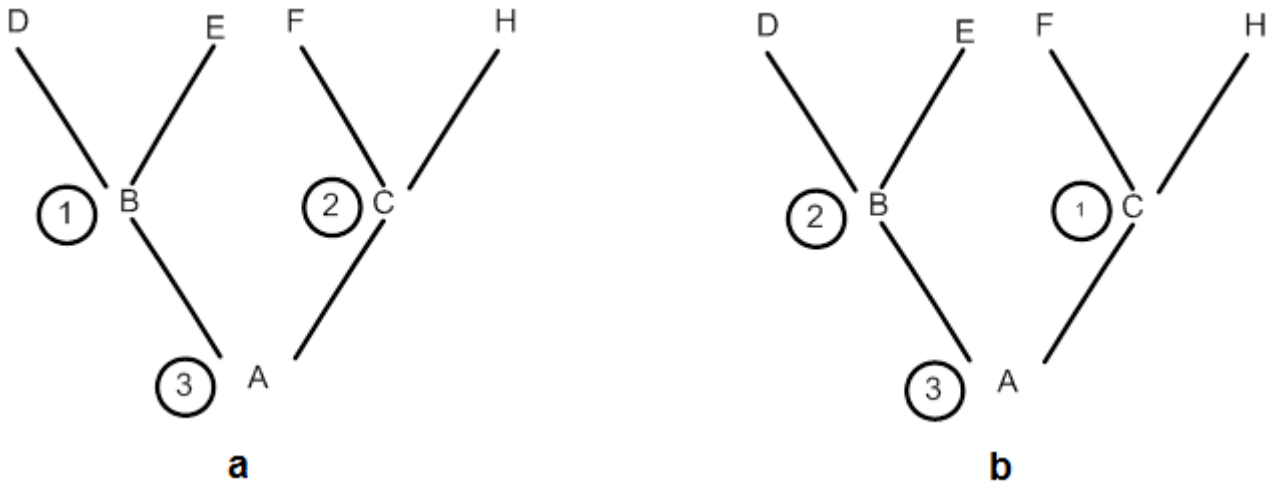


Fig. 6

Figures in mugs mean an analysis order. Productive trees are identical. The difference consists only in the course of analysis: for a - to the left of the line, for b - to the right of the line.

This difference in an analysis method can be excluded by concept *canonically ordered grammatical analysis.*

The canonical form of grammatical analysis is an analysis that is applied from left to right on a line. Thus, the extremely left part of the sentence first of all is understood, if it is possible before to promote on a line to the right for search of a situation accessible to analysis. In fig. *a* it is given initial grammatical analysis of the sentence. However, canonically ordered grammatical analysis cannot always be used. Let's consider examples:

Example 1. It is given grammar

$$\begin{matrix} A \to x \\ A \to Ax \end{matrix}$$ (Left recursion)

and line $xxxx$. Canonical analysis is given in Fig. 7.

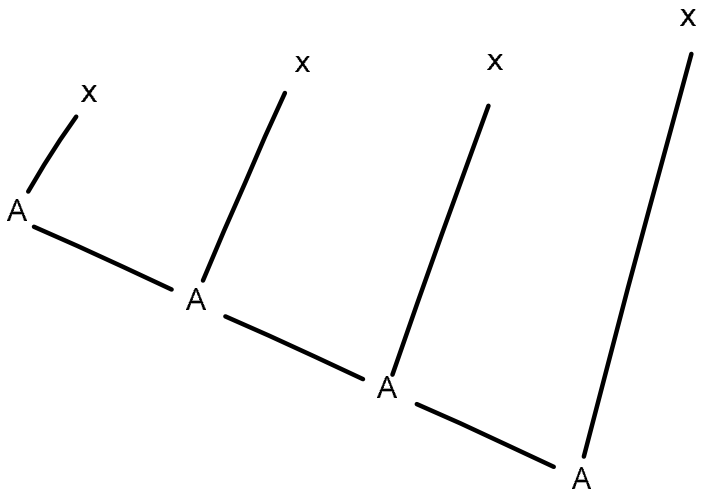


Fig. 7

Example 2. It is given grammar

$$\begin{matrix} A \to x \\ A \to xA \end{matrix}$$ (Right recursion)

and word $xxxx$. Grammatical analysis for the given word cannot be canonical such as it cannot be executed (results in deadlock). Grammatical analysis in this case can be executed only at analysis to the right of a line (Fig. 8).

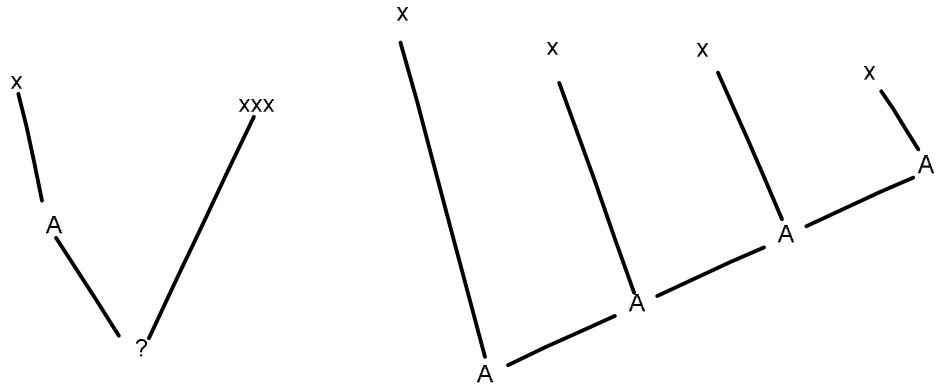


Fig. 8

Example 3. It is given grammar is

$$\begin{matrix} A \to x \\ A \to xAx \end{matrix}$$ (Central recursion)

and word $xxxx$. Make grammatical analysis. In this case, it cannot be successfully finished at sentence view from left to right (initial analysis), at viewing from right to left. In this case, analysis at each stage begins with the middle of the assorted sentence (Fig. 9).

From the considered examples 1, 2, 3 it is possible to draw output, that the way of analysis of the sentence is defined by type of recursion in output rules of grammar. Right recursion predetermines the grammatical analysis beginning to the right.

Left recursion predetermines a successful canonical form of grammatical analysis.

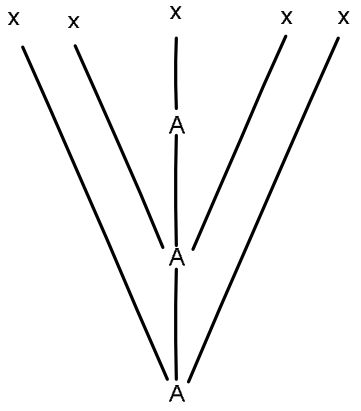


Fig. 9

Central recursion predetermines the grammatical analysis beginning with the middle of the sentence. However, not always the way of grammatical

analysis can be easily established by grammar rules as it was in the resulted examples. We will consider the following example.

Example 4. It is given the following grammar rules:

$A \to \omega x;$

$B \to Ay;$

$C \to Bz \mid \omega D;$

$D \to xE;$

$E \to yv.$

It is necessary to make grammatical analysis of lines ωxyz and ωxyv.

Analysis of the first line can be successfully executed because of application of a canonical form of analysis.

The result of such analysis is shown in Fig. 10, a. Use of initial analysis for the second line leads up a blind alley (Fig. 10, b).

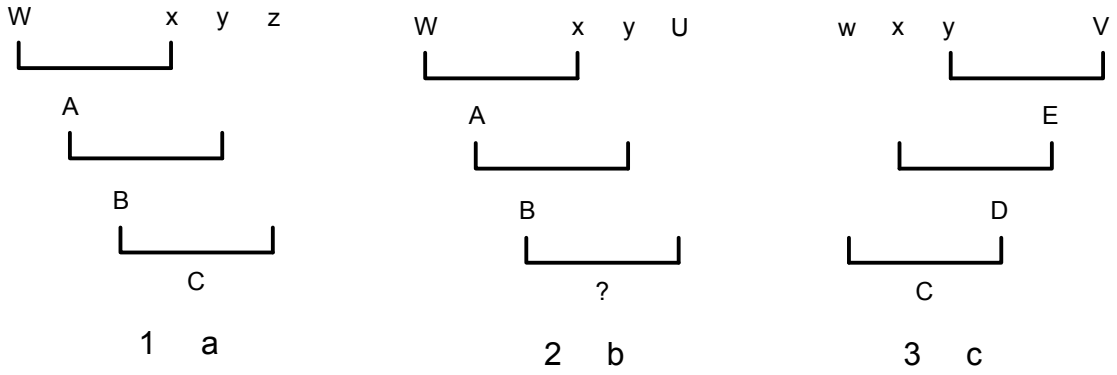


Fig. 10

Successful performance of grammatical analysis of the second line is carried out under a state if analysis is begun with following, beginning at the left, subline, i.e. with yv. Results of such analysis are given in Fig. 10, c. Thus, a problem of grammatical analysis is successful carrying out of the analysis of sentences. The order of grammatical analysis depends on rules of output (syntax) and a kind of analyzed lines or sentences.

Using formalization as the criterion of the classification, all existing methods of grammatical analysis can be divided on ***heuristic*** and ***formalized.***

Formalization of methods consists in ordering of the rules defining correctness or an inaccuracy of an initial line concerning set grammar at application of the given method on each step (stage) of grammatical analysis.

Heuristic methods are not systematized in relation to each step, i.e. they inform only, whether the given line, only after definitive passage of the analyzed text is correct.

The heuristic method is known under the name of a trial and error method, search and substitutions as the correct way of generations is after check of all possible ways of the decision (analysis). Limitation of use of heuristic methods consists that:

1. Correctness or an inaccuracy of a line is defined not at once on each step of analysis, and only after the termination of the analyzed text.

2. Correct way of generation is after check of all possible ways of analysis.

3. Choice of a false way demands return returning to last, correctly certain state.

4. At realization by the machine the sheaf of semantic rules with the syntactic is very difficultly carried out.

All it negatively affects a method from the point of view of time loss.

At the same time some advantages are inherent in heuristic methods also: application possibility to all languages that does their universal; orientation or «on the purpose», or «from the purpose», that defines two directions of a heuristic method - "from top to down" and «from below upwards».

Grammatical analysis by a method «from below upwards» lines s of language L generated with grammar, $G=<V_T,V_N,S,P>$ begins with line s and consists in viewing the sequences received because of analysis, conducting to S (to the purpose). Formally, the purpose of such analysis can be written as: $s => S$ i.e. because of grammatical analysis it is defined whether the given line is a sentence.

All examples of grammatical analysis considered earlier implicitly demonstrated this method.

Grammatical analysis by a method "from top to down", named a "descent" method (or «recursive descent»), begins from the purpose S (i.e. from a starting rule of generation or output), and it is considered the sequence of such generations which would lead further to s. The formalized representation of such analysis is $S => s$. Because of grammatical analysis by the method it is defined the structure of sentence of language is "from top to down”.

Both methods it is possible can be described by the same trees of generations, only in one case a tree root below, in other - above. Therefore, analysis of a word ωxyz, for example 4, resulted by both methods, is given in Fig. 11.

The formalized methods appeared and appear in connection with working out of compilers. Thus in the newest methods sometimes find reflexion good lines of one and lacks of others before the described methods are eliminated.

Working out of each method is adhered to certain classes of machines (small, average or large type) and to certain classes of languages (depending on what languages the machines work with). As all programming languages concern the second class on Chomsky classification, also methods are focused on the second class of languages.

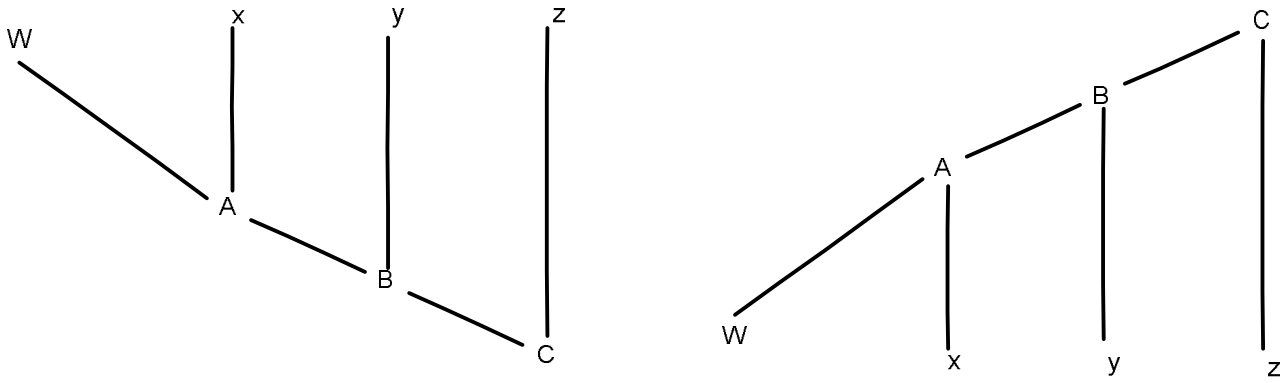


Fig. 11

Some quality monitoring brings restrictions in grammar of language. Usually restrictions are imposed on the right part of rules: or on number of symbols, or on their structure

EXAMPLES AND PRACTICAL TASKS

2.1. Formal grammar

Example — arithmetic expressions

Let's consider the simple language defining the limited subset of arithmetic formulas, consisting of natural numbers, brackets and signs on arithmetic actions. It is necessary to notice, that here in each rule on the left side from an arrow it is necessary one non-terminal symbol. Such grammars are called as context-free.

The terminal alphabet:

$\sum$ = {'0','1','2','3','4','5','6','7','8','9','+','-','*','/','(',')'}.

The non-terminal alphabet:

{FORMUlA, SIGN, NUMBER, FIGURE}.

Rules:

1. FORMULA ⇒ FORMULA SIGN FORMULA	(formula is two formulas, combined sign)
2. FORMULA ⇒ NUMBER	(formula is number)
3. FORMULA ⇒ (FORMULA)	(formula is formula in parentheses)
4. SIGN ⇒ + | - | * | /	(sign is plus or minus or multiply or divide)
5. NUMBER ⇒ FIGURE	(number is the figure)
6. NUMBER ⇒ NUMBER FIGURE	(number is the number and figure)
7. FIGURE ⇒ 0|1|2|3|4|5|6|7|8|9	(figure is 0 or 1 or ... 9)

Initial nonterm:

FORMULA

Conclusion:

Let's deduce the formula (12+5) by means of the listed rules of output. For clarity, the parties of each replacement are shown in pairs, in each pair the replaced part is underlined.

$\underline{\text{FORMULA}} \Rightarrow 3(\underline{\text{FORMULA}})$

$(\underline{\text{FORMULA}}) \Rightarrow 1(\underline{\text{FORMULA SIGN FORMULA }})$

$(\text{FORMULA}\,\underline{\text{SIGN}}\,\text{FORMULA}) \Rightarrow 4(\text{FORMULA}\underline{+}\text{FORMULA})$

$(\text{FORMULA} + \underline{\text{FORMULA}}) \Rightarrow 2(\text{FORMULA} + \underline{\text{NUMBER}})$

$(\text{FORMULA} + \underline{\text{NUMBER}}) \Rightarrow 5(\text{FORMULA} + \underline{\text{FIGURE}})$

$(\text{FORMULA} + \underline{\text{FIGURE}}) \Rightarrow 7(\text{FORMULA} + \underline{5})$

$(\text{FORMULA} + 5) \Rightarrow 2(\underline{\text{NUMBER}} + 5)$

$(\underline{\text{NUMBER}} + 5) \Rightarrow 6(\underline{\text{NUMBER FIGURE}} + 5)$

$(\underline{\text{NUMBER FIGURE}} + 5) \Rightarrow 5(\underline{\text{FIGURE}}\,\text{FIGURE} + 5)$

$(\underline{\text{FIGURE}}\,\text{FIGURE} + 5) \Rightarrow 7(\underline{1}\,\text{FIGURE} + 5)$

$(\underline{1}\,\text{FIGURE} + 5) \Rightarrow 7(1\ \underline{2} + 5)$

Example of regular grammar:

$$G = \{N, T, P, S\}.$$

***N* = {S};**

***T* = {a,b};**

***P*:** $S \to a$; $S \to b$; $S \to aS$; $S \to bS$.

By means of this grammar, the lines of symbols a and b are generated. It is possible to explain sequence of generation of lines with the following scheme:

Applied rule of output	The maintenance of a line
	S
$S \to aS$	aS
$S \to aS$	aaS
$S \to ab$	aab

Result of output is line aab.

Example of context-free grammar:

$$G = \{N, T, P, S\}.$$

***N* = {S};**

***T* = {a,b};**

***P*:** $S \to aSb$; $S \to ab$.

By means of this grammar lines of a kind are generated $a^n b^n$.

Example of output for line $a^3 b^3$.

Applied rule of output	The maintenance of a line
	S
$S \to aSb$	aSb
$S \to aSb$	aaSbb
$S \to ab$	aaabbb

Example of more difficult context-free grammar:

$$G = \{N, T, P, S\}.$$

N = {S};

T = {IF,THEN,ELSE,U,B};

P:S $\to$ *B;*

$S \to IF\ U\ THEN\ S$*;*

$S \to IF\ U\ THEN\ S\ \ ELSE\ S$**.**

This grammar allows forming various conditional operators of Pascal language. For example, the operator IF U THEN IF U THENB ELSE Bbe can generated as follows.

Applied rule of output	The maintenance of a line
	S
S → IF U THEN S	IF U THEN S
S → IF U THEN S ELSE S	IF U THEN S ELSE S
S → B	IF U THEN IF ⋃ THEN B ELSE S
S → B	IF U THEN IF ⋃ THEN B ELSE B

The second variant is possible:

Applied rule of output	The maintenance of a line
	S
S → IF U THEN S ELSE S	IF U THEN S ELSE S
S → IF U THEN S	IF U THEN IF⋃ THEN S ELSE S
S → B	IF U THEN IF ⋃ THEN B ELSE S
S → B	IF U THEN IF ⋃ THEN B ELSE B

Exercises

1. Let - $G = (V_T, V_H, P, S)$ is a generating grammar, where. $V_T = \{a, d, e\}$ $V_H = \{B, C, S\}$ $P = \{S \to aB, B \to Cd, C \to e\}$. Write out the terminal chains generated by the given grammar, and define length of their output.

2. Let - $G = (V_T, V_H, P, S)$, where $V_T = \{a, d, e\}$, $V_H = \{B, C, S\}$, $P = \{S \to aB, B \to Cd, B \to dC, C \to e\}$ Define the terminal chains generated by the given grammar, and length of their output.

3. For grammar G it is known general dictionary $V = \{A, B, C, D, E\}$ and the scheme of rules - $P = \{E \to DCD, E \to A, D \to BC, D \to C, A \to BB\}$. Define structure of terminal and non-terminal dictionaries, the grammar purpose, to construct language $L(G)$ and define length of outputs for each terminal chain.

4. Define, whether the following grammars are generating:

a) $G = (\{A, B\}, \{S, D\}, P = \{S \to AB, S \to ASD, SD \to B, B \to AS\}, S)$;

b) $G = (\{A, B\}, \{S\}, P = \{S \to ASBAS, S \to AB, AS \to B\}, S)$;

c) $G = (\{B, C\}, \{A, S\}, P = \{S \to A, A \to B, A \to CA\}, S)$;

d) $G = (\{B, C\}, \{A, S\}, P = \{S \to A, A \to B, A \to CAC\}, S)$.

5. It is given grammar $G = (V_T, V_H, P, S)$, where $G = (V_T, V_H, P, S)$. $V_T = \{A, B\}$ $V_H = \{S, D\}$ $P = \{S \to AB, S \to ADSB, D \to BSB, DS \to B, D \to \wedge\}$. Prove, that the chain $ABABBAB$ belongs to set $L(G)$.

6. It is given grammar

$$G = (\{a, b, c, d, e\}, \{A, B, C, D, E\}, P = \{A \to ed,$$
$$B \to Ab, C \to Bc, C \to dD, D \to aE, E \to bc\}, C)$$

Define whether the chain $L(G)$ belongs to set $eadbcbc$.

7. It is given grammar $V = \{C, S, a, b\}$ and $V = \{a, b\}$. Define, whether the four (V_T, V_H, P, S) for the following sets of rules a grammar:

a) $P = \{C \to b, S \to aCb\}$;

b) $P = \{b \to a, C \to Sb\}$;

c) $P = \{C \to bCaC, CS \to \wedge\}$;

d) $P = \{C \to bC, CS \to aS, S \to a\}$.

8. Let for every $n \geq 1$ $G = (\{a\}, \{S \to S^n\}, S)$. Prove, that what whatever $n, L(G_n = \otimes)$.

9. It is determined terminal dictionary of grammar. Define grammars generating following languages:

a) language for $a^n b^n a^n$ $n \geq 1$;

b) language a^{n2} for $n \geq 1$;

c) language $a^n b^{n2}$ for $n \geq 1$.

2.2 Context-free grammars

Exercises

1. Let $V_T = \{a, b\}$. Develop grammar, generating the following languages:

Language $L = \{a^n b^n a^n \mid n \geq 1\}$;

Language $L = \{a^{n2} \mid n \geq 1\}$;

Language $L = \{a^n b^{n2} \mid n \geq 1\}$.

2. Grammars G_1 and G_2 are set by the rules:

$$P_1 = \{A \to aAbB, A \to bB, A \to A, A \to b,$$
$$B \to bAbb, B \to B, B \to bAb, B \to ab\};$$
$$P_2 = P_1 - \{A \to A, B \to B\}.$$

Show that $L(G_1) = L(G_2)$.

3. It is given grammar $G = (V_T, V_H, P, S)$ where. $V_T = \{a, b, c\}$, $V_H = \{S, B, S\}$, $P = \{S \to aSBC, S \to aBC, CB \to BC, aB \to ab, bB \to bb, bC \to bc, cB \to cc\}$. Define type of grammar and the language generated by it.

4. Prove, that each linear language is generated by grammar in which each rule is either left or right linear.

5. It is set two grammars G_1 and G_2 with the following rules:

$$P_1 = \{S \to abS, S \to cA, A \to baA, A \to a\};$$
$$P_2 = \{S \to AB, A \to aAb, B \to aBb, A \to c, B \to c\}.$$

Define grammar type.

6. Prove, that language $L = \{a^m b^m a^n b^n\}$ is not linear.

7. Construct an example of Ks-language not being A-language and generated by Ks-grammar where each rule is either let-linear or right-linear.

8. Let $G = (V_T, V_H, P, S)$ where $V_T = \{a, b, c\}$ $P = \{S \to AB, A \to aAb, B \to aBb, A \to c, B \to c\}$. Prove that Ks-language is metalinear, but not linear.

9. Grammar G is set by rules $P = \{S \to aSa, S \to bSb, S \to aaSaa, S \to c\}$. Prove that it is not essentially ambiguous.

10. Language $L = \{a^n b^m c^r \mid n + m \geq r\}$ is Ks-language. Write out the Ks-grammar generating this language. Define, to what narrower class of languages it belongs to.

2.3 Basic properties of languages

Exercises

1. It is given language $L = \{aab, aaaab, aaaaaab\}$.

Execute operations of multiplication, iteration and transposition over it.

2. It is set a dictionary of terminal symbols $V = \{c, b\}$ and language $V = \{c, b\}$ $L = \{c^n b c^n \mid n \geq 1\}$.

Define language addition.

3. It is set languages $L_1 = \{abbc, ab, abcc, ac\}$ and $L_2 = \{xy, xyz, xz, xzy, yz, yzx\}$. Execute operation of multiplication of these languages.

4. It is set language $L = \{a^n b^n \mid n \geq 1\}$. Execute operations of a transposition, multiplication, and iteration.

5. Let $V_T = \{a, b\}$ $L = \{a^i b^j a^j \mid i, j \geq 1\}$ $M = \{a^j b^j a^i \mid i, j \geq 1\}$ $L_1 = \{a^i b^j a^k \mid i, j, k \geq 1\}$

Prove, that language $L_2 = L \cap M$ is context-free.

6. It is set language $L = \{a^n c b^n \mid n \geq 0\}$. Execute all possible operations over the given language.

CONTROL QUESTIONS AND TASKS FOR SELF-CHECKING

1) What words are called equal?
2) What are the components of language?
3) What is the essence of formalistic approach in?
4) What is problem type that is connected to occurrence and formalistic approach development?
5) What are the features of formal languages?
6) What are the basic constructions of formal languages?
7) Characteristics of unlimited formal grammar?
8) In what is the essence of contextual formal grammar?
9) Give the characteristic of context-free formal grammar?
10) Give the characteristic of regular formal grammar?
11) What is the grammatical analysis?
12) What are the basic types of generating grammars?
13) Classes of formal grammars?
14) What is the basic definition for context-free grammars?
15) Is it possible to give interrelation of grammar classes as a graph?
16) What are the properties of formal languages?
17) Can various operations be applied to any sets?
18) What are the basic classes of methods of grammatical analysis?
19) In what limitation of use of heuristic methods consists?

THEMES FOR INDEPENDENT WORK

1) Chomsky - Schuttsenberzhe. Metalanguage.
2) Forms of Bekus-Naur (FBN).
3) Examples of the description of FBN identifier.
4) Examples of identifier description.
5) Figure Wirth.
6) Examples of description of identifier with figure Wirth.
7) Definition and structure of recognizer.
8) Elementary designs.
9) Examples of elementary designs.

CHAPTER 3. FINITE STATE MACHINES AND THEIR CONNECTON TO LANGUAGES AND GRAMMARS

3.1. GENERAL DEFINITION OF FINITE STATE MACHINE

Finite state machine (FSM) is called the five

A = (N, T, P, S, F),

where N – finite set of states of the automatic machine;

T – alphabet – finite set of symbols;

P – transition function of automaton;

S – initial state $S \in N$;

F – set of finite states $F \subseteq N$.

In the beginning, the automaton is in state S. On input FSM, the symbols belonging to the input alphabet arrive. The sequence of input symbols forms an input chain. Being in some state and having received on an input the next symbol, the automaton passes in the following state defined by value of function of transitions.

Generally function of transitions for given pair of symbols – a state can define some variants of transition. In that case, the automaton is called not determined (NFSM).

If having read an input chain the automaton α stopped in some state B they say, that α converts the automaton in V. If state N is one of finite states, i.e. $F\ B \in F$ then FSM allows chain α.

The set of all chains accepted by the automaton forms language $L(A)$ accepted by the automaton.

The language generated by automatic grammar G, coincides with the language accepted by the corresponding finite state machine

$$L(G) = L(A).$$

FSM can be set by means of the diagram of transitions. For example, the graph of automatic grammar G_8 can be considered transition diagramme A_8.

At transition from automatic grammar a FSM generally receives not determined FSM that complicates its use in a role of recognizer for automatic language. Indeterminacy the automaton is defined by that for some tops of its diagram of transitions there are some arches leaving these tops and marked with the same symbol.

For elimination of ambiguity NFSM translate in DFSM (determined FSM).

The simplest model is the automaton with finite number of states (with finite memory) — the finite state machine.

Determined finite state machine is called the ordered system from five objects — «the ordered five»:

$$A = (X, S, s_0, \delta, F).$$

Where $X = \{x_1, x_2, ..., x_r\}$ is a set of input symbols (input alphabet), $S = \{s_1, s_2, ..., s_r\}$ is s set of inner states, r, n are finite, $s_0 \in S$ is initial state, δ is a function displaying $S \times X$ in S that is usually written as $\delta : S \times X \to S$. This function unequivocally puts in conformity to pair of symbols (s_i, x_j) some symbol - $s_k \in S$, $F \subseteq S$ some allocated subset of states of the automaton (finite states).

This automaton can be interpreted as it is shown in Fig. 12.

The automaton consists of the actuation device, a reading out head and infinite to the right the input tape divided into cages. In the beginning on a tape the input chain is so written down, that in each cage of a tape contains on one symbol of a chain. The initial symbol is written down in an extreme left cage, and all cages of that part of a tape which is located more to the right of last symbol of record of an input chain, are empty, i.e. In each of these cages the "empty" symbol A is written down. The head during the initial moment is located against an extreme left cage of a tape, and the actuation device is in an initial state.

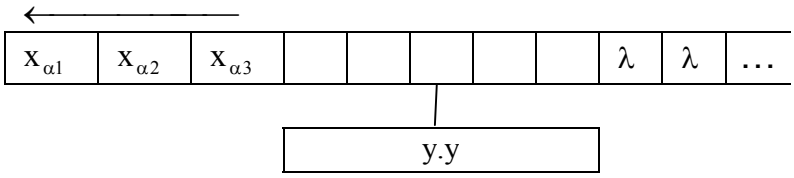


Fig. 12

In each next step, the head perceives a symbol on a tape, the actuation device changes the state according to display δ and the tape moves on one cage to the left. If the head perceives an empty symbol $\wedge$ that is written down more to the right of last symbol of input sequence it will mean automaton cessation of work, i.e. the actuation device state does not change any more.

Admissible or comprehensible is called an input sequence possessing following property: when the head perceives last symbol of this input sequence, the automaton passes in one of set states F.

The set of admissible input sequences of the automaton A or set of sequences, representable in the automaton A by set of states F is designated $L(A)$. Such set, being set of chains of symbols from the finite alphabet X will be from the point of view of the theory of languages to some languages over the terminal dictionary $V_T = X$. Thus, naturally there is a way of comparison of automatons, languages and grammars.

For finite state machines, the set $L(A)$ is allocated with known Kleene theorem.

The theorem 41 (Kleene). For any finite state machine the set A of admissible $L(A)$ sequences is regular, i.e. language of type 3. This theorem

explains, why type 3 languages are called as languages with finite number of states.

As the determined finite state machine with two tapes is called the ordered six $A = (X, Y, S, s_0, \delta, \lambda)$.

Where X, S, s_0 and δ have the same sense as finite state machine with one tape, - $Y = \{y_1, y_2, ..., y_l\}$ is a set of output symbols (output alphabet), l - finite, λ is a function reflects $S \times X$ in Y i.e. $\lambda : S \times X \to Y$

Interpretation of this automaton is given in Fig. 13.

But for input tape, the automaton has infinite to the right a output tape that can move only to one party — from right to left. Each next step in a tape cage is printed a symbol, and the tape moves on one cage. This automaton is called as consecutive machine, or Mealy machine. Mealy machine M to each chain of input symbols u unequivocally puts a chain of output symbols ω that registers in conformity with $\omega\, M(u) = \omega$. It is obvious, that $M(\wedge) = \wedge$.

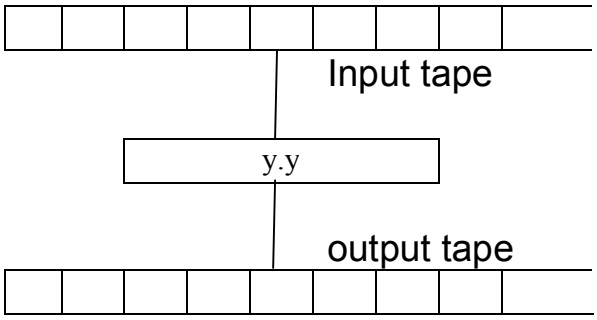


Fig. 13

Nondetermined finite state machine with one tape is called the ordered five $A = (X, S, S_0, \delta, F)$.

Where X, S, F have the same sense, as in definition of the determined automaton, δ set $S \times X$ in set of all subsets of states S and S_0 some allocated (initial) subset of set S.

Representation of set of input sequences in not determined automaton is understood as follows. The set of input sequences U is admissible in not determined finite state machine, if for each sequence of this set there will be such two states $s_i \in S_0$ and $s_i \in F$ and such variant of work (i.e. Such concrete values of function δ), that the sequence U translates the automaton from state s_i into state s_j.

The following theorem is proved.

The theorem 42 (Rabin and Scott). In not determined single-tape finite state machines as well as in determined are admissbible only regular sets of chains. By means of not determined automatons it is impossible to expand a class of representable sets. However not determined finite state machine

usually has smaller number of states in comparison with the determined automaton representing the same set.

The finite state machine with two tapes is not determined, if at the same situation characterised in pair, (s_i, x_j) probably finite number of variants of its action. Hence, such automaton can put to one input chain in conformity finite set of output chains.

The bilateral finite state machine differs from the usual determined automaton that the input tape infinite in both parties and can move not only to the left, but also to the right, and also can remain motionless. Thus displaying function is replaced $\delta : S \times X \to S$ with function, $\delta : S \times X \to S \times d$ where $d = \{-1,0,+1\}$. The symbol d shows in what party the input tape should move: - 1 corresponds to shift on one cage to the right, $+1$ to shift to the left on one cage, 0 the tape remains motionless.

The set $L(A)$ of all admissible sequences of such automaton is defined by the following theorem.

The theorem 43 (*Rabin, Scott).* For each bilateral automaton A it is possible to define effectively such automaton B having the same set of comprehensible input sequences, as the automaton A. Therefore the set $L(A)$ is regular i.e. it is type 3 language.

Thus, the bilateral automaton does not expand possibility of finite state machines.

3.2. MEALY MACHINE AND MOORE MACHINE

Mealy machine is a finite state machine, which output sequence (unlike Moore machine) depends on a state of the automaton and input signals. It means that in a state graph to each edge there corresponds some value (a output symbol). In tops of the graph of Mealy machine leaving signals register, and to arches of the graph attribute a state of transition from one state in another, and also input signals. Mealy machine can be described the five, (Q, X, Y, f, g) where Q set of states of the automaton, X set of input symbols, Y set of output symbols, $q = f(Q, X)$ function of states, $y = g(Q, Y)$ function of output symbols.

The law of functioning of Mealy machine is set by the equations:

$$a(t+1) = \delta(a(t), z(t)); \; w(t) = \lambda(a(t), z(t), t = 0,1,2,...).$$

The law of functioning of Moore machine is set by the equations:

$$a(t+1) = \delta(a(t), z(t)); \; w(t) = \lambda(a(t)), t = 0,1,2,...).$$

From comparison of laws of functioning, it is visible, that, unlike Mealy machine, the output signal in Moore machine depends only on a current state of the automaton and in an explicit form does not depend on an input signal. For the full task of Mealy and Moore machines in addition to functioning laws, it is necessary to specify an initial state and to define internal, input and output alphabets. Mealy machines - automatic machines of 1st sort, R a-bus Moore

machines – automatons of 2nd sort, S a-bus $C = R + S$ the-combined automatons.

3.2.1. Synthesis of mealy machine

At a stage of reception inputs of the tops following for operational, mark a flowgraph of algorithm symbols $a_1, a_2, \ldots$ following rules:

1) 1)symbol a_1 marks an input of the top following for initial, and also an input of finite top;

2) inputs of all tops following for operational, should be noted;

3) inputs of various tops, except for finite, are marked by various symbols;

4) if the top input is marked, only one symbol.

For carrying out of marks the finite number of symbols $a_1, \ldots, a_m$ is required. Result of the first stage is noted the flowgraph of algorithm which forms a basis for the second stage - transition to the graph or tables of transitions-exits.

At the second stage, from noted algorithm flowgraph, the graph of the automaton or the table of transitions-exits is built. For this purpose they believe, that in the automaton there will be so much states how many symbols i a_i t was required at a mark an algorithm flowgraph.

On a drawing plane it is marked all states of the automaton a_i. For each of states a_i define on noted an algorithm flowgraph all ways which are conducting in other states and passing necessarily only through one operational top.

On the basis of noted an algorithm flowgraph it is possible to construct the table of transitions-exits. For microprogram automatons the table of transitions-exits is under construction in the form of the list and direct and return tables differ. For the given automaton the direct table is given to Tab. 2, return - in Tab. 3.

Table 2

Am	As	X	Y
a_1	a_2	1	y_1, y_3
a_2	a_5	$\overline{x_2}$	y_6
	a_7	x_2	y_4
a_3	a_4	1	y_2
a_4	a_5	$\overline{x_5}$	y_6
	a_6	x_5	y_7, y_{10}
a_5	a_6	1	y_7, y_{10}
a_6	a_8	$\overline{x_4}$	y_2
	a_9	x_4	y_2, y_4
a_7	a_9	1	y_2, y_4

Table 3

Am	As	X	Y
a_{22}	a_1	$x_6, \overline{x_5}$	y_1, y_9
a_{23}		$\overline{x_5}$	y_1, y_9
a_{24}		x_2	-
a_1	a_2	1	y_1, y_3
a_{10}	a_3	$\overline{x_5}, x_6$	в3,в6
a_3	a_4	1	y_2
a_{12}		$\overline{x_2}$	y_2
a_2	a_5	$\overline{x_2}$	y_6
a_4		$\overline{x_5}$	y_6
a_4	a_6	x_5	y_7, y_{10}

Ending of Table 2

Am	As	X	Y
a_8	a_{10}	1	y_3, y_6
a_9	a_{11}	1	y_7
a_{10}	a_3	$\overline{x_5}, x_6$	y_3, y_6
	a_{11}	x_5	y_7
	a_{12}	$\overline{x_5}, \overline{x_6}$	y_8
a_{11}	a_{12}	x_1	y_8
	a_{13}	$\overline{x_1}$	y_1, y_9
a_{12}	a_4	$\overline{x_2}$	y_2
	a_{16}	x_2	y_2, y_4
a_{13}	a_{16}	1	y_2, y_4
a_{14}	a_{15}	1	y_3, y_6
a_{15}	a_{17}	x_5	y_7
	a_{18}	$\overline{x_5}, \overline{x_6}$	y_8
	a_{20}	$\overline{x_5}, x_6$	y_3, y_6
a_{16}	a_{17}	1	y_7
a_{17}	a_{18}	x_1	y_8
	a_{19}	$\overline{x_1}$	y_1, y_9
a_{18}	a_{21}	$\overline{x_2}$	y_3, y_4
	a_{22}	x_2	y_6, y_9
a_{19}	a_{21}	1	y_3, y_4
a_{20}	a_{22}	1	y_6, y_9
a_{21}	a_{23}	1	y_8
a_{22}	a_1	$x_6, \overline{x_5}$	y_1, y_9
	a_{14}	$\overline{x_6}, x_1$	y_2
	a_{24}	$\overline{x_6}, \overline{x_1}$	y_7
a_{23}	a_1	$\overline{x_5}$	y_1, y_9
	a_{24}	x_5	y_7
a_{24}	a_{15}	$\overline{x_2}$	y_3, y_6
	a_1	x_2	-

Ending of Table 3

Am	As	X	Y
a_5		1	y_7, y_{10}
a_2	a_7	x_2	y_4
a_6	a_8	$\overline{x_4}$	y_2
a_6	a_9	x_4	y_2, y_4
a_7		1	y_2, y_4
a_8	a_{10}	1	y_3, y_6
a_9	a_{11}	1	y_7
a_{10}		x_5	y_7
a_{10}	a_{12}	$\overline{x_5}, \overline{x_6}$	y_8
a_{11}		x_1	y_8
a_{11}	a_{13}	$\overline{x_1}$	y_1, y_9
a_{22}	a_{14}	$\overline{x_6}, x_1$	y_2
a_{14}	a_{15}	1	y_3, y_6
a_{24}		$\overline{x_2}$	y_3, y_6
a_{12}	a_{16}	x_2	y_2, y_4
a_{13}		1	y_2, y_4
a_{15}	a_{17}	x_5	y_7
a_{16}		1	y_7
a_{15}	a_{18}	$\overline{x_5}, \overline{x_6}$	y_8
a_{17}		x_1	y_8
a_{17}	a_{19}	$\overline{x_1}$	y_1, y_9
a_{15}	a_{20}	$\overline{x_5}, x_6$	y_3, y_6
a_{18}	a_{21}	$\overline{x_2}$	y_3, y_4
a_{19}		1	y_3, y_4
a_{18}	a_{22}	x_2	y_6, y_9
a_{20}		1	y_6, y_9
a_{21}	a_{23}	1	y_8
a_{22}	a_{24}	$\overline{x_6}, \overline{x_1}$	y_7
a_{23}		x_5	y_7

3.2.2. Synthesis of moore machine

At a stage of reception a flowgraph of algorithm the marking is made for Moore machine according to following rules:

1) the symbol a_1 marks initial and finite tops;

2) various operational tops are marked by various symbols;

3) all operational tops should be noted.

The table of transitions-exits of Moore machine is given in Tab. 4 (straight line). Usually for Moore machine in the table of transitions-exits the additional column for output signals not used also a output signal registers in a column where the initial state am or transition states S is underlined.

Table 4

Transitions-exits of the Moore machine

Am(y)	**As**	**X**
$a_1( )$	a_2	1
$a_2(y_1, y_3)$	a_4	x_2
	a_5	$\overline{x_2}$
$a_3(y_2)$	a_5	$\overline{x_5}$
	a_6	x_5
$a_4(y_4)$	a_7	1
$(a_5(y_6)$	a_6	1
$a_6(y_7, y_{10})$	a_7	x_4
	a_8	$\overline{x_4}$
$a_7(y_2, y_4)$	a_{10}	1
$(a_8(y_2)$	a_9	1
$a_9(y_3, y_6)$	a_{10}	x_5
	a_{12}	$\overline{x_5}, \overline{x_6}$
	a_{13}	$\overline{x_5}, x_6$
$a_7(y_7)$	a_{11}	$\overline{x_1}$
	a_{12}	x_1
$a_{11}(y_1, y_9)$	a_{14}	1
$a_{12}(y_8)$	a_{14}	x_2
	a_3	$\overline{x_2}$
$(a_{13}(y_3)$	a_3	1
$a_{14}(y_2, y_4)$	a_{16}	1
$a_{15}(y_3, y_6)$	a_{16}	x_5
	a_{18}	$\overline{x_5}, \overline{x_6}$
	a_{19}	$\overline{x_5}, x_6$
$a_{16}(y_7)$	a_{17}	$\overline{x_1}$
	a_{18}	x_1
$a_{17}(y_1, y_9)$	a_{20}	1

Ending of Table 4

Am(y)	As	X
$a_{18}(y_8)$	a_{20}	x_2
	a_{22}	$\overline{x_2}$
$a_{19}(y_3)$	a_{22}	1
$a_{20}(y_3, y_4)$	a_{21}	1
$a_{21}(y_8)$	a_{23}	$\overline{x_5}$
	a_{24}	x_5
$a_{22}(y_6, y_9)$	a_{23}	$x_6, \overline{x_5}$
	a_{24}	$\overline{x_6}, \overline{x_1}$
	a_{25}	$\overline{x_6}, x_1$
$a_{23}(y_1, y_9)$	a_1	1
$a_{24}(y_7)$	a_1	x_2
	a_{15}	$\overline{x_2}$
$a_{25}(y_2)$	a_{15}	1

3.2.3. Transformation of mealy machine to moore machine

Between Mealy and Moore machiness (Fig. 14) there is conformity, allowing transforming the law of functioning of one of them to another or back. Moore machine can be considered as a special case of Mealy machine, meaning, that the sequence of states of exits of Mealy machine advances sequence of states of exits of Moore machine on one step, i.e distinction between Mealy and Moore machiness consists that in automatons of Mealy the exit state arises simultaneously with a state of an input causing it, and in Moore machines - with a delay on one step, suh as in Moore machines input signals change only an automaton state.

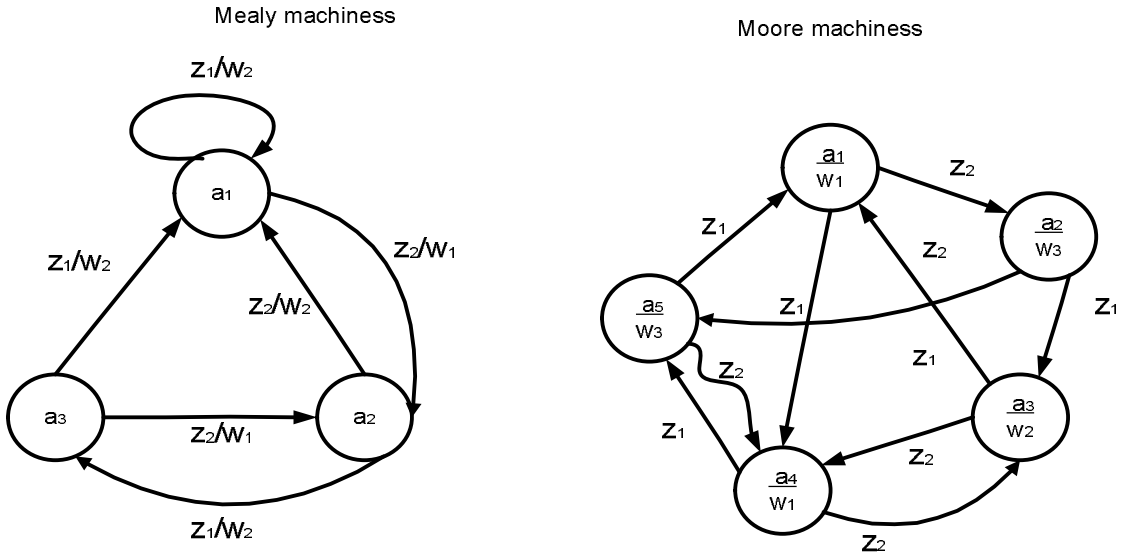


Fig.14

Let it be necessary to transform Mealy machine to Moore machine.
The graph of Mealy machine is on Fig.15:

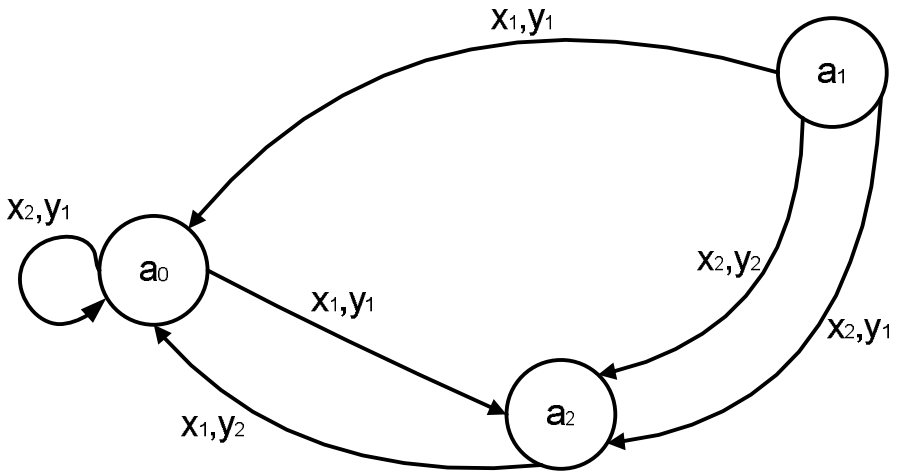


Fig. 15. The graph of Mealy machine

In Mealy machine $X_a = \{x_1, x_2\}, Y_a = \{y_1, y_2\}, A_a = \{a_0, a_1, a_2\}$.
In equivalent Moore machine $X_b = X_a = \{x_1, x_2\}, Y_b = Y_a = \{y_1, y_2\}$.
Let's construct set of states of the automaton A_b of Moore for what we will find sets of the pairs generated by each state of the automaton S_a.

Condition	Generated pairs
a_0	$\{(a_0, y_1), (a_0, y_2)\} = \{b_1, b_2\}$
a_1	$\{(a_1, y_1)\} = \{b_3\}$
a_2	$\{(a_2, y_1), (a_2, y_2)\} = \{b_4, b_5\}$

From here they have sets of states A_b of Moore machine $A_b = \{b_1, b_2, b_3, b_4, b_5\}$. To find function of outputs L_b with each state representing to steam of a kind, (a_i, y_g) let's identify the output signal which is the second element of this pair. The result is as follows:

$$b(b_1) = l_b(b_3) = l_b(b_4) = y_1;\ b(b_2) = l_b(b_5) = y_2.$$

Let's construct function of transitions d_b. Such as in the automaton S_a from state a_0 there is a transition under the influence of a signal x_1 in state a_2 from delivery from y_1 set of the states generated, $\{b_1, b_2\}$ a_0 in the automaton S_b there should be a transition in state $(a_2, y_1) = b_4$ influenced by signal x_1. Similarly, from $\{b_1, b_2\}$ under the influence of x_2 there should be a transition in state $(a_0, y_1) = b_1$. From $(a_1, y_1) = b_3$ under the influence of x_1 in transition $(a_0, y_1) = b_1$ and x_2 - under the influence of $(a_2, y_2) = b_5$. At last from states $\{(a_2, y_1), (a_2, y_2)\} = \{b_4, b_5\}$ under the influence of x_1 in $(a_0, y_2) = b_2$ and

x_2 - under the influence of $(a_1, y_1) = b_3$. As a result we have a graph (Fig. 16) and the table of transitions of the equivalent Moore machine.

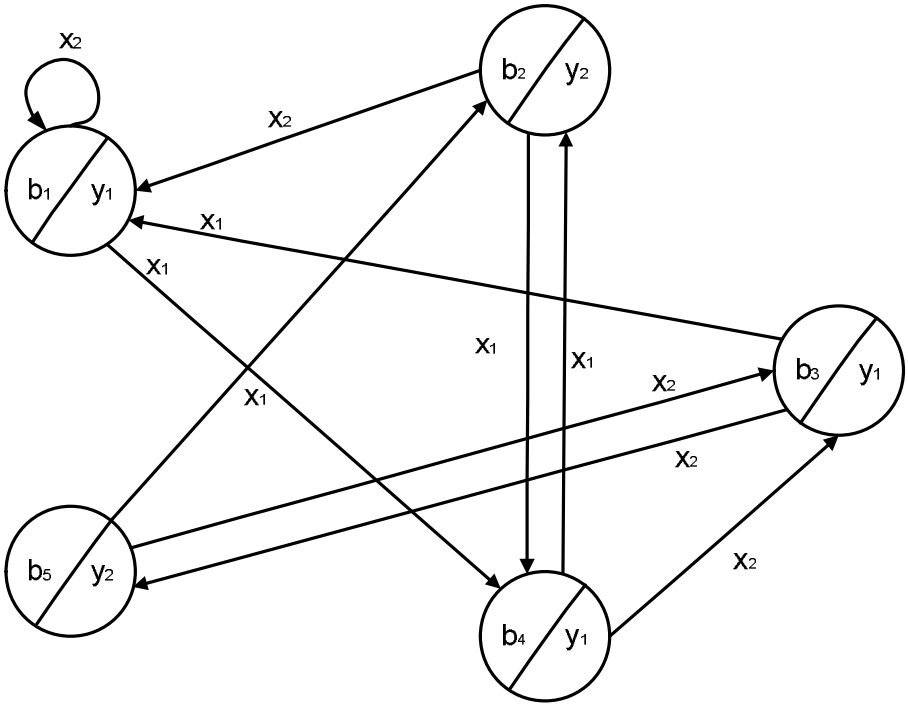


Fig. 16. The graph of the equivalent Moore machine

y_g	y_1	y_2	y_1	y_1	y_2
$x_j \backslash b_j$	b_1	b_2	b_3	b_4	b_5
x_1	b_4	b_4	b_1	b_2	b_2
x_2	b_1	b_1	b_5	b_3	b_3

As an initial state of the automaton S_b it is possible to take any of states b_1 or b_2 such as both of them are generated by state a_0 of automaton S_a.

3.2.4. Connection between mealy and moore machines

Two automatons with identical input and output alphabets are called equivalent if after their installation in their initial state, their reactions to any input word coincide.

Despite the fact that automatons function differently, it is always possible to construct the automaton of one model of other model equivalent to the automaton in the sense that their reactions to one and the same input chains will be identical. The general approach to developing automaton equivalents:

1) let it be given Moore automatic machine that it is necessary to transform Mealy machine to equivalent.

$S_a = (A_a, Z_a, W_a, \delta_a . \lambda_a, a_1 A)$ — Moore machine,

$S_b = (A_b, Z_b, W_b, \delta_b . \lambda_b, a_1 B)$ — Mealy machine.

2) let's require: $A_b = A_a, Z_b = Z_a, W_b = W_a, \delta_b = \delta_a.\lambda_a, a_1 B = a_1 A, \lambda_b - ?$.

To transform from Moore machine to Mealy machine it is necessary in the column of Moore machine to take out a symbol of the output alphabet from considered top and to attribute it to all arches entering into this top. In the equivalent Mealy machine quantity of states are the same, as well as in Moore machine.

Transformation from Mealy machine to equivalent Moore machine is more complicated. It is due to the fact that in Moore automatic machine only one output signal is developed. The single restriction imposed on possibility of such transformation is that the initial automatic machine of Mealy should not have unattainable states:

1) Let it be given Mealy machine which it is necessary to transform into equivalent Moore machine.

$S_a = (A_a, Z_a, W_a, \delta_a.\lambda_a, a_1 A)$ - Mealy machine,

$S_b = (A_b, Z_b, W_b, \delta_b.\lambda_b, a_1 B)$ - Moore machine.

Let's require $Z_b = Z_a, W_b = W_a$.

2) Let's define the set of states of Moore, A_b for this purpose each state, $A_S \in A_a$ is put in conformity set A_S which represents every possible steams of a kind (a_S, w_g) where w_g - are output signals which have been put down along arches of Mealy machine, entering into top a_S.

Set A_b is an association $A_S = A_b \; S \in (1, M)$ Generally, quantity of tops in Moore machine is more than in Mealy machine. Output function λ_b and transition function W_b are defined as follows: to each state of Moore machine, represents pair, (a_S, w_g) we put in conformity with output signal w_g. If in Mealy machine S_a there was a transition from fish-traps a_m under the influence of signal in z_f in top a_S i.e. $S_a(a_m, z_f) = a_S$ herewith it makes a signal $\lambda_a(a_m, z_f) = w_k$ to Moore machine there will be a transition from set of states, $S_a(a_m, z_f) = a_S$ under the influence of w_k the same input signal z_f.

3.3. REPRESENTATION OF FORMAL GRAMMARS POSSIBILITIES IN THE FORM OF FINITE STATE MACHINES

The resulted definitions from section 3.1 of the description of the nomenclature of definitions and unequivocal conformity with formal languages of various types are reflected in Tables 5, 6.

Conformity between principal views of automatic machines and representable languages in them are shown in Tab. 5.

Table 5

Order	Automaton names	Type of language
1	Determined and not determined finite state machine	$\longleftrightarrow 3$
2	Determined push down machine	$\longleftrightarrow 2$
3	Not determined push down machine	$\longleftrightarrow 2$
4	Determined liner limited automaton	$\leftarrow -2$ $- \rightarrow 1$
5	Not determined liner limited automaton	$\longleftrightarrow 1$
6	Determined and not determined Turing machine	$\longleftrightarrow 0$

In this table the following designations are used. The arrow $\longleftrightarrow$means, that language $i(i = 0,1,2,2^D,3)$ when and only when we will give in the type of automatic machine $A_k (k = 1,2,...,6)$. The arrow means, $- \rightarrow$ that if language we will present in the automaton A_k is language i. The arrow means, $\leftarrow -$ that if language is if type i then there will be automatic machine A_k where it is representable.

In summary we consider possible treatment of concepts and results of the general theory of languages with reference to three basic areas where these results are used: to mathematical linguistics, programming languages and the theory of automatic machines.

In Tab. 6 similar concepts from these three areas are collected and concepts of the general theory of languages corresponding to them are specified.

Table 6

Designation	The general theory of languages	Mathematical linguistics	Programming	The mathematical Models
V_T	The terminal dictionary	The basic dictionary of	Output symbols	The output alphabet
V_H	The non-terminal dictionary (alphabet)	Auxiliary grammatical terms	Set of commands	Set of states of the actuation device
S	Initial non-terminal symbol	Sentences	The program	Initial condition
P	System	Syntactic rules	Operations	Displaying function

EXAMPLES AND PRACTICAL TASKS

3.1. Finite state machines

Exercises

1) On each of resulted below finite state machines construct A right liner Ks-grammar generating set- $T(A)$.

$A = (S, X, p, \delta\{p_3\}), \delta$ Tab. 7 is set $A = (S, X, p, \delta\{p_3\}), \delta$.

$A = (S, X, p_1, \delta\{p_4, p_5\}), \delta$ Tab. 8 is set $A = (S, X, p_1, \delta\{p_4, p_5\}), \delta$.

Table 7

	A	B
P_1	P_2	P_5
P_2	P_3	P_4
P_3	P_2	P_5
P_4	P_3	P_1
P_5	P_2	P_4

Table 8

	A	B	C
P_1	P_2	P_3	P_1
P_2	P_1	P_6	P_6
P_3	P_4	P_4	P_2
P_4	P_2	P_3	P_1
P_5	P_4	P_6	P_3
P_6	P_1	P_2	P_6

On crossing of line p_i and column x it is given value $\delta(p_i, x)$.

2) Develop MT-automaton M_1 such that $X = \{a, b\}$ and $T(M)$— set of all chains containing identical number of input symbols and a b.

3) Develop MT-automatic machine, M such that $X = \{a, b\}$ and $T(M) = \{a^n b^n a^i \mid n \geq 1, i \geq 1\} U \{a^n b^{2n} a^i b^{3i} \mid n \geq 1, i \geq 1\}$.

4) Develop MT-automatic machine supposing language, generated by grammar with rules $P = \{s \to TT, T \to Ta, T \to bT, T \to c\}$.

5) It is given a set of commands of the finite state machine supposing language $\{a^n b^n \mid n, m \geq 0\}$:

$$(a_1 S_0) \to S_1;$$
$$(a_1 S_1) \to S_1;$$
$$(b_1 S_1) \to S_2;$$
$$(b_1 S_2) \to S_2;$$
$$(b_1 S_2) \to S_0.$$

Construct the grammar generating this language, and define its type.

6) Construct linear limited automatic machines supposing languages:

$$L_1 = \{ a^n b^n a^n \mid n \geq 0 \};$$
$$L_2 = \{ xcx \mid x \in \{a, b\}^* \}.$$

Language $L = \{a^p b^q c^r \mid p + q \geq r\}$ is KC-language.

Write out the Ks-grammar generating this language. Define what narrower class of languages it belongs to. Construct the determined MT-automatic machine supposing this language.

7) What of the following sets of sequences can be distinguished as the finite state machine:

a) Set of all sequences: 0, 1, 00, 01, 10, 11, 000, 001, 010, …;

b) Numbers 1, 2, 4, 8, …, 2n, …, written down in a binary notation;

c) The same set of the numbers which have been written down in a monadic code: 1, 11, 1111, 11111111, 1111111111111111, …;

d) Set of sequences, in which number of 0t is equal to number of 1;

e) Sequences: 0, 101, 11011, …, 1k01k (k – number of enterins of 1)?

3.2. Mealy and moore machines

Any finite set of words E = {a1.... ak} can be represented in the automatic machine. The idea of construction of the automatic machine on finite set of words is illustrated by the graph in Fig. 17, where finite states q_{n-k}....., q_{n-1} are represented by a double circle. For specific sets this idea is modified because words can have commom beginnings (then the beginnings of corresponding ways need to be united not to break a automation state) or easier to contain in each other (then from one finite state there is a way to other finite state). The example of the automatic machine for *E* = {*ab, the expert, abaa*} with finite states *F* = {3, 5, 6} is given in Fig. 17.

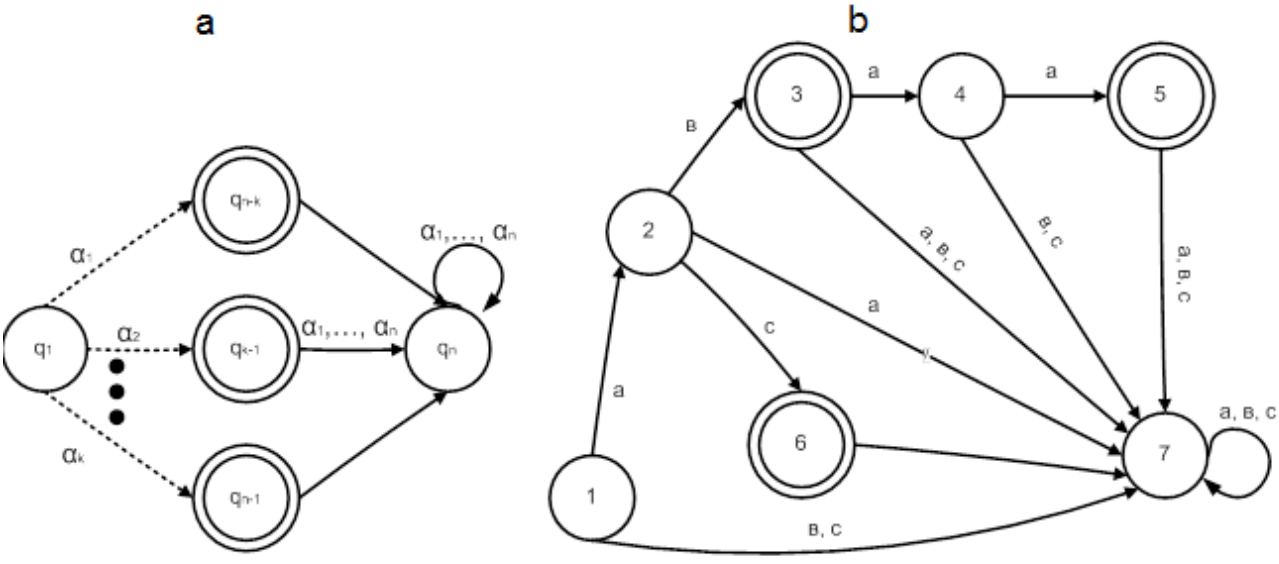


Fig. 17. The idea of construction of the automatic machine on finite set of words

In the automaton representing finite set of words, the way from an initial state in any finite state cannot contain cycles or contain in a cycle as then there would be an infinite set of ways from an initial state in F and corresponding

event would be infinite. Therefore such automatic machine cannot be strongly coherent, it is the device, so to say, disposable action.

a) Independent automatons represent events in single-letter alphabet; words in such events differ only in length. For example, the automaton from example 2.3 with initial state 1 and F = {7} (outputs are ignored) is an infinite event consisting of all words whose lengths at division on 4 give in the rest 3. If to put F = {2} this automaton is of empty event.

b) The automaton, whose graph given in Fig. 18 (F = {1}), is an infinite set *{e, aba, abaaba., {aba}*.

Events are sets of finite words. However it is possible to say, that the automaton distinguishes infinite sequence of letters $\alpha = a_{i_1}, a_{i_1}, a_{i_1}, \ldots$, if it is a set E = {$a_{i_1}, a_{i_1}, a_{i_1}, \ldots$}, consisting of all initial pieces of sequence α.

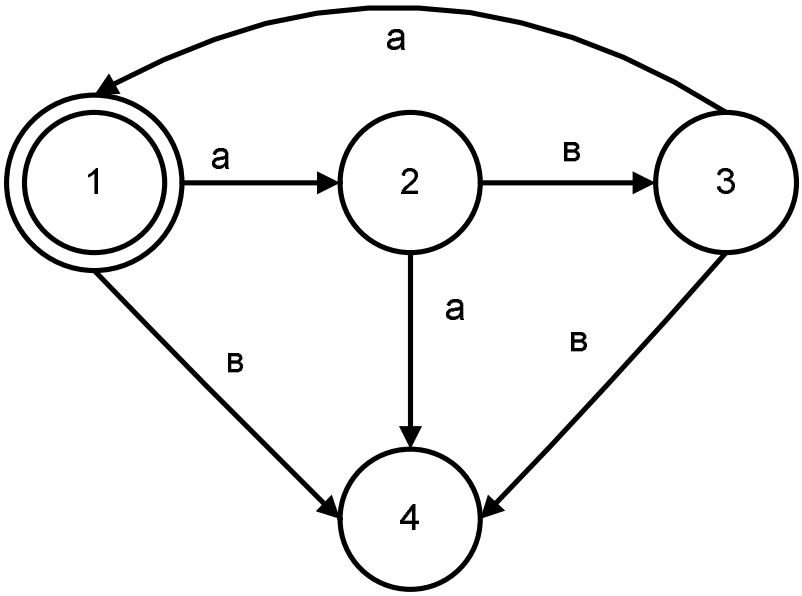


Fig. 18

CONTROL QUESTIONS AND TASKS FOR SELF-CHECKING

1) The general definition of the finite state machine.
2) What the basic essence of finite state machine?
3) Interpretation of the finite state machine.
4) Not determined finite state machine.
5) Finite state machine with two tapes.
6) Mealy machine?
7) Moore machine?
8) How is it possible to convert Mealy machine to Moore machine?
9) What is the connection between Mealy and Moore models?

TOPICS FOR SELF-DEPENDENT WORK

1) The linear-bounded automaton.
2) Connection between linear-bounded automaton and context-sensitive grammars.
3) Kurod theorem.
4) Context-sensitive languages.
5) Push-down automaton.
6) Context-free languages characteristics.
7) Push-down automatons with single-letter transitions.
8) Hayes- theorem.
9) Examples of not determined push-down automaton.
10) Algorithmic problems.
11) Algorithmically unsolvable problems.
12) Algorithmically solvable problems.

BIBLIOGRAPHIC LIST

1. Аляев, Ю.А. Дискретная математика и математическая логика [Текст]: учебник / Ю.А. Аляев, С.Ф. Тюрин. – М. : Финансы и статистика, 2006. – 368 с.

2. Гуц, А.К. Математическая логика и теория алгоритмов [Текст] : учеб. пособие / А.К. Гуц. – Омск: Наследие. Диалог-Сибирь, 2003. – 108 с.

3. Игошин, В.И. Математическая логика и теория алгоритмов [Текст] : учеб. пособие / В.И. Игошин. – М. : изд. центр «Академия», 2008. – 448 с.

4. Колмогоров, А.Н. К определению алгоритма [Текст] / А.Н. Колмогоров, В.А. Успенский // Успехи математических наук. – 1958. – Т. 13. – № 4(82) - С. 3 – 28.

5. http://ru.wikipedia.org/wiki/Формальная_грамматика.

6. Хопкрофт, Джон. Введение в теорию автоматов, языков и вычислений [Текст] / Джон Хопкрофт, Раджив Мотвани, Джеффри Ульман. – М. : Вильямс, 2002. – 528 с.

CONTENT

Навчальне видання

Шостак Ігор Володимирович
Данова Марія Олександрівна
Бутенко Юлія Іванівна
Груздо Ірина Володимирівна

ТЕОРІЯ АЛГОРИТМІВ І ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

(Англійською мовою)

Редактор В.В. Рижкова
Технічний редактор Л.О. Кузьменко

Зв. план, 2013
Підписано до видання 12.04.2013
Ум. друк. арк. 4,4. Обл.– вид. арк. 5. Електронний ресурс

Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001