COMPUTATION MODELS AND ALGEBRA OF ALGORITHMS

Volodymyr K. Ovsyak

Opole University of Technology, Opole, Poland
and
Ukrainian University of Printing, L'vov , Ukraine

An analysis of the existing, intuitive computation models is presented, that is the virtual machines of Turing, Post, Kolmogorov, Schönhage, Aho-Ullman-Hopcroft as well as the algorithms of Markov and Krinitski, and the recursive functions. The need for tools of precise, mathematical formulation and possible transformation of the algorithms is indicated. Consequently, an algebra of algorithms is defined using the axiomatic method. The algebra is based on the operations of sequencing, elimination, paralleling and inverting as well as cyclic sequencing, cyclic elimination and cyclic paralleling, all of them performed on the so-called uniterms.

1. INTRODUCTION

The algorithms or models of computations are the essence of computer systems. In particular, it is important to be able to precisely and formally describe algorithms for the developed operating systems, platforms, programming languages and specialized computer systems, including monitoring, control and diagnostic systems. The importance results from the fact that the cost of location and correction of algorithmic errors when implementing or operating the algorithms is very high.

The well-known existing models of computations are the Turing machine [1,2], Post machine [3,4], Kolmogorov machine [5,6,7], Schönhage machine [8,9], Markov algorithms [10,11], recursive functions [12,13,14,15], the Aho, Ullman and Hopcroft machine (with random access to RAM memory) [16] and the analytical algorithm theory of Krinitski [17]. However, such machines in reality do not exist. Indeed, these are specific tools for algorithm presentation that occurred between the thirties and nineties of the 20^{th} century. It is well known [18] that the above models are equivalent in that they compute 'the same functions'. It is also known [18] that the algorithms are described in those models in an intuitive way. In practice, there appears the need for precise, formal description of the algorithms. This could give means for transformation of the algorithms and their optimization in the sense of reduction of a number of operations and memory occupation, thus contributing to more effective implementation of the algorithms. Such yields are not possible in case of the intuitive descriptions of the algorithms within the framework of the existing models.

In order to explain the reasons that cause the descriptions of the algorithms with the existing models to be informal, it is desirable to assume a certain concept of a brief analysis of the existing computation models. Its essence relies on verification whether in the known methods there exist tools for formal description of relations between their operations. The analysis is presented in Section 2, with brief descriptions of the existing, intuitive computation models as well as the rationale for formal description and transformation of the algorithms highlighted. The alphabet and operations of the originally introduced algebra of algorithms are given in Section 3. Section 4 provides two illustrative examples of application of the algebra of algorithms and new results of the paper are summarized in Section 5.

2. ANALYSIS OF METHODS FOR FORMALIZATION OF ALGORITHMS

2.1. Turing machine

According to Ref. [1,2], the Turing machine is the sextuple of elements

$$N = <Y, S, \Pi, q_0, q_1, a_0>,$$

where Y - external alphabet (nonempty finite set of characters),

S - internal alphabet (alphabet of internal states) – nonempty finite set of characters,

Π - machine program, which is a sequence of the commands

$$K_1$$
$$K_2$$
$$\cdot$$
$$\cdot$$
$$\cdot$$
$$K_m$$

being of type

$$q_a \longrightarrow rb\ ,$$
$$q_a \longrightarrow rbR\ ,$$
$$q_a \longrightarrow rbL\ ,$$

where $q, r \in S$ and $a, b \in Y$, whereas the symbols $\longrightarrow$, R and L belong neither to Y nor to S, and the commands are to be respectively read in the following way:

- "being in the state q and seeing the sign 'a', go to state r and write the sign 'b'",
- "being in the state q and seeing the sign 'a', go to state r, write the sign 'b' and go to the right",
- the third command is analogous to the second one except of "… and go to the left",

with

q_0 - initial state of the machine,
q_1 - terminal state of the machine,
a_0 - empty sign.

An algorithm is a machine program that transfers the machine from its initial to terminal state, which is described by a sequence of the machine commands.

2.2. POST MACHINE

The Post machine consists of a tape and head (or carriage) [3,4]. The tape is infinite and divided into cells. Each cell may be either empty, when nothing is written into, or written or labeled with some sign, e.g. "+". The cells are ordered like integers. One of them is distinguished and called the initial point.

The head is a device that performs writing, reading and erasing of labels in the cells. During one step the head can be moved by one cell to the left or right. The movement of the head starts from the initial point.

The tape contains 1) input data on which certain operations are performed and 2) results obtained during machine operation.

The following operations are performed by the head:

a) writing a label into a cell,
b) erasing a label from a nonempty cell,
c) moving by one cell to the right,
d) moving by one cell to the left,
e) read out – recognition whether a cell is either labeled or empty.

A set of the so-called instructions is introduced in order to control the head, forming a list of operations and specifying the succession of their execution. Hence, the head is controlled with the instructions. Each instruction has a unique integer number. An instruction controlling the operation of the head located in the initial point is always numbered as 1.

The set of instructions comprise the following:

a) start from the initial point and perform the instruction 1,
b) perform an operation O_i (one of the operations a) through d)), where i is the number of operations, and go to an instruction J_i, where J is the number of instruction written in the instruction i,
c) perform operation e) and depending on a result ("yes" or "no") go to performing the instruction J_i^1 or J_i^2, the numbers of which are written in instruction i,
d) stop.

An algorithm constitutes the following program of the Post machine:

$$\begin{array}{ll} 1 & O_r \\ 2 & O_p \\ 3 & O_s \\ & . \\ & . \\ & . \\ W & O_k, \end{array}$$

where O_r, O_p, O_s, . . . are the instruction of type a), b) and c) whereas O_k is the instruction of type d.

Operations performed by the Post machine can be denoted with various symbols, e.g.

$\rightarrow$ - moving the head by one cell to the right,
$\leftarrow$ - moving the head by one cell to the left,
- marking a label in a cell,
$\times$ - erasing a label from a cell,
? - read out – recognition of contents of a cell,
$\dashv$ - stopping the machine.

2.3. MARKOV ALGORITHMS

The Markov algorithm (normal algorithm) [10,11] over any word (or string) p in the alphabet A is

$$\begin{array}{l} q^1{}_1 \longrightarrow q^1{}_2 \\ q^2{}_1 \longrightarrow q^2{}_2 \\ \quad . \\ \quad . \\ \quad . \\ q^i{}_1 \longrightarrow\!\bullet\ q^d{}_2 \end{array}$$

where $q^i{}_1$ and $q^i{}_2$ are the words in the alphabet A and

$i \in \{1, 2, \ldots, d\}$,
A – alphabet, that is a finite set of symbols,
p – word over alphabet A, that is a finite sequence of symbols from alphabet A,
$\longrightarrow$ and $\longrightarrow\!\bullet$ - symbols of operations.

The essence of the operation $q^i{}_1 \longrightarrow q^i{}_2$ is as follows: the word p is analyzed from the left to the right and it is verified if it contains the subword $q^i{}_1$. If so then the first found word $q^i{}_1$ is replaced with the subword $q^i{}_2$ and the next operation $q^{i+1}{}_1 \longrightarrow q^{i+1}{}_2$ is performed. Otherwise, the operation $q^{i+1}{}_1 \longrightarrow q^{i+1}{}_2$ is immediately entered.

The operation $q^d{}_1 \longrightarrow\!\bullet\ q^d{}_2$ differs from $q^i{}_1 \longrightarrow q^i{}_2$ in that if the subword $q^d{}_1$ is contained in the word p, then this former operation terminates the execution of the algorithm.

The subword q is contained in the word p only if $p=p_1qp_2$, where p_1 and p_2 are the words (possibly empty).

A sequence of substitutions presents an algorithm making use of the Markov method.

2.4. KOLMOGOROV MACHINE

The Kolmogorov machine [5,6,7], like those of Turing and Post, is a virtual machine. It operates on a set of data and uses specific operations. Data sets are the so-called complexes, that is *B*-complexes and (*B*, *k*)-complexes, where *B* is the finite alphabet of symbols and *k* is the number of symbols in the alphabet *B*. A word created from the alphabet symbols can be both a single symbol itself and a finite set of symbols. An operation of combining the symbols is introduced in order to form words (or strings). A graph can be obtained by encircling the symbols and operations of their combination as well as introducing relations between them. There are two intermediate "address" vertices introduced between each pair of graph vertices. The first additional vertex contains the number of a symbol used in the combination operation and the second one – the number of the combination operation. A connected graph is obtained such that, for each vertex, all the neighboring vertices have different numbers. A vertex containing the first symbol is assumed as the initial one. The obtained graph is a complex.

A program consists of local Kolmogorov operations. A set of all operations is finite and is denoted by the direct-processing operator Ω. Operation of the machine according to the program is reduced to transferring from the initial state A^0 to $A^1=\Omega(A^0)$, from A^1 to $A^2=\Omega(A^1)$ and so on until the stop signal. Thus, the operation program for the Kolmogorov machine is as follows

1 $A^1=\Omega(A^0)$,
2 $A^2=\Omega(A^1)$,
.
.
.
d $A^d=\Omega(A^{d-1})$

2.5. RECURSIVE FUNCTIONS

An algorithm based on recursive functions [12,13,14,15] is presented as a translator of characters and words from a finite and nonempty alphabet (input words) to characters and words of alphabet *A* or to characters and words of finite and nonempty alphabet *B* (output words). The input and output words are enumerated with natural numbers. Processing of the input words into the output ones can be described with integer arithmetic functions.

In order to develop the algorithm, two finite sets are created, one for arithmetic functions and the other one for rules enabling to create complex arithmetic functions from the elementary ones.

The following elementary arithmetic functions are selected: the function identically equal to zero (for all natural arguments), the identity function $f(x_i) = x_i$, the function of direct consequence $f(x_i) = x_i+1$ defined for all natural values of its argument.

The following operating rules are selected: superposition, elementary recursion and least root. The superposition operation consists in substitution of an arithmetic function for other arguments of arithmetic functions.

The elementary recursion operation leads to construction of a *d*-variable function from the prespecified (*d*-1)-variable and (*d*+1)-variable functions along with the following scheme:

$$g(x_1,K,x_{d-1}) = f(x_1,K,x_{d-1},0)$$
$$h(x_1,K,x_d,y) = f(x_1,K,x_{d-1},x_{d+1})$$

where $y=f(x_1,\ldots,x_d)$, f is the constructed function and g and h are the prespecified functions.

The least root operation enables to construct a d-variable function $f(x_1,\ldots,x_d)$ from the prespecified $(d+1)$-variable function $g(x_1,\ldots,x_d,y)$. For any prespecified set of variable values $x_1=\alpha_1,\ldots, x_d=\alpha_d$, it is the least natural root $y=\beta$ of the equation $g(x_1,\ldots,x_d,y) = 0$ that is assumed as a value of the function $f(x_1,\ldots,x_d)$. The function is undetermined if such a root does not exist. The function is also undetermined when the least root exists but the equation $g(\alpha_1,\ldots, \alpha_d,y) = 0$ is undetermined for at least one natural number $y = \gamma$ such that $0\leq\gamma\leq\beta$-1. Applying the above rules to elementary arithmetic functions one can construct complex functions called partly recursive ones.

Each elementary arithmetic function can be described with a machine program, e.g. the Post machine program. Then a partly recursive function constructed from elementary arithmetic functions can also be described with the Post machine program.

The construction of a partly recursive function f_p consists of the following. In the first step, a function f_1 is obtained through the application of the above rules to prespecified elementary functions. In the second step, a function f_2 is constructed from the function f_1 and elementary functions, and so on until a function f_p is obtained. In such a case, an algorithm is a sequence of the functions:

$$f_1$$
$$f_2$$
$$\vdots$$
$$f_p.$$

2.6. RANDOM ACCESS MEMORY MACHINE

The RAM machine includes [16]:

a) input tape, read only,
b) output tape, to write,
c) memory.

An input tape is a sequence of cells. Each cell may contain an integer. Having read the integer, the reading head moves by one cell to the right.

An output tape is also a sequence of cells. The output tape is empty before starting the operation of the machine. An integer is written into a cell of the output tape only when the write command is executed. Having executed the write, the writing head moves by one cell to the right.

Memory is an infinite sequence of registers r_0, r_1, r_2, … Each register can store any integer. All the computations are executed in the register r_0 called an adder.

Read-out from the input tape, write to the output tape, computations and other operations like checking the conditions and jumps are executed according to the machine program. A program is a sequence of the commands

$$\begin{array}{ll} m_1 & K_i \\ m_2 & K_j \\ & \vdots \\ m_d & K_k, \end{array}$$

where $m_1, m_2, \ldots, m_d$ are the command labels (may be absent), $K_i, K_j, \ldots, K_k$ are the commands of various types and of finite number, e.g. they can be input/output commands, interruption service commands, transfer/jump commands. Each command includes a code and address of an operation, the latter being a number or label of the command.

The machine has a command counter that is set to a number of the first command before starting the execution of the program. Having executed the *k*-th command, the counter is automatically set to the number of the next command. This will be (k +1) provided that the executed command was not of the unconditional type (nor a conditional jump under a fulfilled jump condition, nor the machine stop command). A number of the counter is the same as the command number prespecified by the unconditional jump command.

In this case, an algorithm constitutes of a sequence of commands (whether or not provided with labels).

2.7. SCHÖNHAGE MACHINE

The Schönhage machine [8,9] consists of an input tape with a reading head, an output tape with a writing head and a memory. Words from the {0, 1} alphabet are stored in the input and output tapes. At each moment of the machine operation, the memory has the *D*-structure which is a finite directed graph, with all the vertices having a unique number of outgoing directed edges. They are uniquely denoted with elements of the *D* alphabet. One of the vertices is distinguished as the initial one or the center of the *D* structure. Addresses are assigned to all the remaining vertices.

Operation of the machine is performed according to a program, which is a finite set of commands. There are three types of commands: operation, control and stop of the machine.

Operation of the machine is started with the execution of the first command, with the input tape head set on the first character. If the operation command is executed, it is followed by the next command. In case the control command has been executed, depending on a result of execution of that command the program is transferred to one of two commands, the numbers of which are located in the control command. Execution of the last command leads to stopping the machine.

A program consists of a sequence of the commands

$$\begin{array}{ll} 1 & K_i \\ 2 & K_j \\ & . \\ & . \\ & . \\ N & K, \end{array}$$

where K_i, K_j, ... are the operation or control commands, and K is the machine stop command.

The program constitutes an algorithm for operation of the Schönhage machine.

2.8. ANALYTICAL THEORY OF THE KRINITSKI ALGORITHMS

There are two types of algorithms distinguished within the framework of the analytical theory of the Krinitski algorithms [17]: basic algorithms and wide-sense algorithms. Basic algorithms are synthesized using a rule of their execution and two languages. The first language is used to write input data and the second one to write basic algorithms.

Input data are sequences of characters, that is words created from symbols of the first language (data language), and quasiwords, that is words with a distinguished sign.

The language of basic algorithms is composed of 17 basic operations, with 13 of them creating one group of operations and the remaining ones being the operations of control of fulfillment of specified conditions.

A basic algorithm is a sentence from an algorithmic language considered together with its execution rule:

1: *Oa*
2: *Ob*
.
.
.
d: *Ok*,

where *Oa*, *Ob*, ... , *Ok* are the basic operations (from the list of the 17 ones), that is sentences from the algorithmic language.

The execution rule for the basic algorithm is as follows [17]:

1. Find the first operation in the basic algorithm and go to item 2.
2. Verify if either the conditional or unconditional jump operation is to be executed. In the first case, go to item 4, otherwise go to item 5.
3. Find an operation code. Execute a suitable operation. Go to item 2.
4. Find in the algorithm, moving from the start to the end, such an operation which label is with send. Go to item 2.
5. Find a condition in the operation. Verify if it is fulfilled. If so, go to item 6, otherwise go to item 7.
6. Assume the first label consisting in the conditional jump operation as a new send. Go to item 4.

Those basic algorithms that are considered without any execution rule and that consist of basic operations only are called natural algorithms.

A wide-sense algorithm is developed using a procedure of its execution and two languages, with processing of several words of input data language being admitted. It is not only input language words but also algorithms themselves that can constitute input data.

Wide-sense algorithms are described in the same way as basic algorithms. In the Krinitski theory, algorithms are sequences of basic operations.

2.9. TOWARDS TRANSFORMATION OF ALGORITHMS

In general, the algorithms in the above described models of computations are presented as the following sequence of operations:

S_0
S_1
.
.
.
S_{n-1},

where S_0, S_1, … , S_{n-1} are the Turing commands, Post instructions, Markov substitutions, Kolmogorov transfers, recursive functions, RAM commands, Schönhage commands and basic operations of the algorithmic language for the Turing machine, Post machine, Markov algorithms, Kolmogorov machine, recursive function algorithms, RAM machine, Schönhage machine and the Krinitski algorithm theory, respectively.

In all the above models of computations, there are defined the executed operations, or algorithm components, whereas no formal relations between the operations/components are considered and no properties of the operations are described (like commutativity, associativity). Therefore, it is not possible to transform the algorithms, e.g. in order for their minimization. Also, it is not possible to perform formal transformations of the algorithms in case of their presentation in form of block diagrams.

There are some tendencies to the development of the ‘metric’ theory of algorithms [10], covering the issues of their computational complexity, but not their transformations aiming at the minimization. In order

to reduce implementation losses for the algorithms it is important to have means for their transformation. The next part of this work presents a description of the algebra of algorithms, including means for mathematical description and transformation of abstract algorithms for the purpose of their minimization. This will essentially contribute to reduction of computational complexity of the algorithms, in addition to a new, nice formal framework.

3. ALGEBRA OF ALGORITHMS

Here we present fundamentals of the algebra of abstract algorithms originally introduced in Refs. [19,20]. Constructed with the axiomatic method, the algebra enables to describe algorithms by means of mathematical formulae.

3.1. ALPHABET

Definition 1. Any symbols to be subject to ordering will be called the **uniterms**.

For instance, the following uniterms can be exemplified: *1, 7, a, c, x,….* Uniterms are divided into the **terminal** (or concrete), e.g. $p=q+2$, $S_1=2x-3$ and **abstract** ones, e.g. $F(x)$, $R(x,y)$. The abstract uniterms are denoted by capital Latin letters (with or without indices).

Definition 2. The **alphabet** of the theory of algorithms consists of the following:

1) symbols of operations: $\frown$ - sequencing; $\vdash\!\dashv$ - elimination; $\sqcap$ - paralleling; $\overline{\ \ }$ - inverting; $\curvearrowleft$ - cyclic sequencing; $\not\supset$ - cyclic elimination; $\varnothing$ - cyclic paralleling; = - equalizing. (Note: Vertical arrangement of the denotations for sequencing, elimination and paralleling is also used.)

2) uniterms not related with the cyclic operations, that is variables, coefficients and constants, with or without indices, denoted with lower-case letters from the beginning of the Latin alphabet:

$$a, b, c_o, c_1, \dots, c_j, c^0, c^1, \dots, c^I, c^i_j, \dots$$

3) uniterms related with the cyclic operations, that is variables, with or without indices, denoted with lower-case letters from the end of the Latin alphabet:

$$x, y, z_o, z_1, \dots, z_j, z^0, z^1, \dots, z^I, z^i_j, \dots$$

4) uniterms dependent on one or more variables, with or without indices, denoted with capital letters from the Latin alphabet:

$$P(a),\ P(a,b), \dots$$

5) conditional uniterms assuming two values (0 or 1)

$$u, u_o, u_1, \dots, u_j, u^i_j$$

6) ***** - empty uniterm;

7) coma (,), semicolon (;), colon (:) (Note: Coma and semicolon are used to separate commutative and noncommutative uniterms, respectively, whereas colon is used to denote either coma or colon.)

OPERATIONS

Definitions of the operations in the algebra of algorithms are given below.

3.2.1. **Equalizing** is an operation on uniterms having the following properties:
1) identity: $A = A$,
2) symmetry: if $A = B$, then $B = A$,
3) transitiveness: if $A = B$ and $B = S$, then $A = S$.

3.2.2. **Sequencing** is an operation having the following properties:

1) commutativeness: $\overset{\frown}{R, S} = \overset{\frown}{S, R}$

Note: Two noncommutative uniterms will be separated by semicolon (rather than coma), so that

$$\overset{\frown}{R; S} \neq \overset{\frown}{S; R}$$

2) associativeness: $\overset{\frown}{R, \overset{\frown}{S, T}} = \overset{\frown}{\overset{\frown}{R, S}, T}$

3) idempotency: $\overset{\frown}{S, S} = S$

4) absorption of the empty uniterm *: $\overset{\frown}{S} = S$, where the symbol ":" means comma or semicolon;

5) extracting a common uniterm:

$$\overset{\frown}{\overset{\frown}{A; B}, \overset{\frown}{A; C}} = \overset{\frown}{A; \overset{\frown}{B, C}},$$

$$\overset{\frown}{\overset{\frown}{A; B}, \overset{\frown}{C; B}} = \overset{\frown}{\overset{\frown}{A, C}; B}$$

3.2.3. **Elimination** is an operation having the following properties:

1) selection of a uniterm :

$$\overset{\vdash\!\!\dashv}{R; S; u - ?} = \begin{cases} R, \text{ if } u = 1; \\ S, \text{ if } u = 0; \end{cases}$$

2) idempotency: $\overset{\vdash\!\!\dashv}{S ; S ; u - ?} = S$**;**

3) selection of a condition:

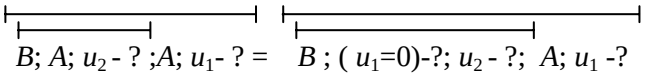
$$\overset{\vdash\!\!\dashv}{R; S ; u_1 - ? ;} \quad \overset{\vdash\!\!\dashv}{R; S ; u_2 - ? ;} \quad u_3 - ? = \overset{\vdash\!\!\dashv}{R; S;} \quad \overset{\vdash\!\!\dashv}{u_1 - ?; u_2 - ?;} \quad u_3 - ?$$

4) absorption of a uniterm :

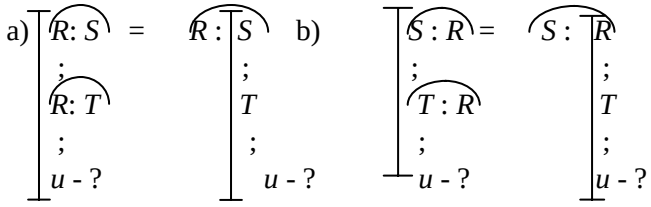
$$\overset{\vdash\!\!\dashv}{A ; \overset{\vdash\!\!\dashv}{B ; C ;} u-? ;} \; u-? = \overset{\vdash\!\!\dashv}{A; C; u - ?}\,,$$

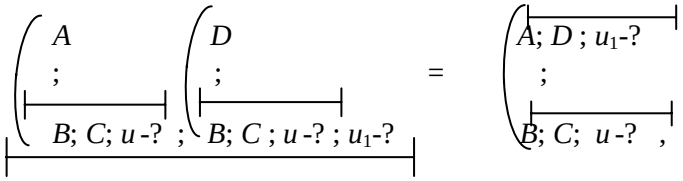
$$\overset{\vdash\!\!\dashv}{\overset{\vdash\!\!\dashv}{A; B; u-? ;} C ; u-?} = A ; C ; u - ?$$

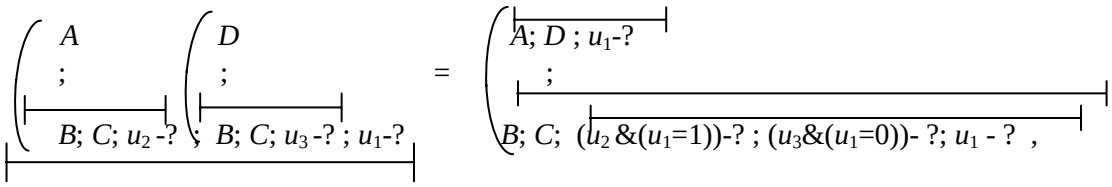
$$A; \quad \overset{\vdash\!\!\dashv}{A; B; u_2 - ? ;} \; u_1 - ? = \overset{\vdash\!\!\dashv}{A; (u_1 = 1) - ?; B ; u_2 - ?;} \; u_1 - \mathbf{?}$$



5) distributiveness:







3.2.4. **Paralleling** is an operation having the following properties:

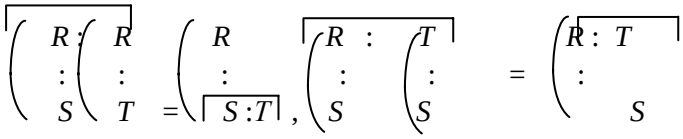
1) idempotency: $\overline{S , S} = S$;

2) absorption of the empty unitherm: $\overline{* , S} = S$;

3) commutativeness: $\overline{R, S} = \overline{S, R}$;

4) associativeness: $\overline{\overline{S, R} , T} = \overline{S , \overline{R, T}}$;

5) extract of a unitherm:



3.2.5. **Inverting** is an operation having the following properties:

1) inverting of sequencing:

$$\overline{A; B} = B; A,$$

2) inverting of elimination:

$$\overline{A; B; u - ?} = B; A; u - ? = A; B; \overline{u -} ?,$$

3) inverting of paralleling:

$$\overline{\ulcorner A; B \urcorner} = \ulcorner B; A \urcorner .$$

3.2.6. **Cyclic sequencing (⊄), cyclic elimination (⊅) and cyclic paralleling (Ø)** are operations having, respectively, the following properties:

1) inverting of a variable related with a cyclic operation:

$\overline{\not\subset x\ A_x} = \not\subset \overline{x}\ A_x$, $\overline{\not\supset x\ A_x} = \not\supset \overline{x}\ A_x$, $\overline{Ø\, x\ A_x} = Ø\, \overline{x}\ A_x$,
where x is the variable and A_x is the unitherm (repeated in each cycle);

2) empty loop:

$$\not\subset x\ *_x = \not\supset x\ *_x = Ø\, x\ *_x = *$$

3.3. ABSTRACT ALGORITHM

An **algorithm** is defined **abstract** or **formal** if it is composed of one or more (possibly all) expressions specified below, executed in a finite number of iterations (in particular, once).

1. If S and R are uniterms, then $\frown S{:}R$ and $\frown S{:}R$ are abstract algorithms.
2. If A and B are uniterms or abstract algorithms, and u is the conditional unitherm, then

$\vdash A;B;u - ?\dashv$ and $\vdash B; A; u - ?\dashv$ are abstract algorithms.

3. If A and B are abstract algorithms, then $\overline{A}$, $\overline{B}$, $\overline{\frown A{:}B}$, $\overline{\ulcorner A{:}B \urcorner}$ and $\overline{\vdash A; B ; u- ?\dashv}$ and $\overline{\vdash B ; A ; u- ?\dashv}$, where u is the conditional uniterm, are abstract algorithms.
4. If A_x is a uniterm or abstract algorithm, x is the variable related to a cyclic operation $\not\subset x\ A_{x,}$

$\not\supset x\ A_x$, $Ø\, x\ A_x$, then $\overline{Ø\, x\ A_x}$, $\overline{\not\subset x\ A_x}$, $\overline{\not\supset A_x}$, are abstract algorithms.

Algebra of algorithms is defined as a system of the above specified operations on the uniterms. It provides means for formal transformation of algorithms and their minimization in terms of a minimum number of uniterms.

4. EXAMPLES

Example 1. Algorithm for solving the quadratic equation $ax^2+bx+c=0$, consisting of the sequencing and elimination stages.

A) Synthesis of sequences (being the terminal uniterms):

- sequence describing calculation of two roots:

$$S_0 = a=w_0, b=w_1, c=w_2 ;\ \Delta=w_1^2-4w_0w_2 ;\ x_1=(-w_1+\sqrt{\Delta})/(2w_0)\ ;\ x_2=(-w_1-\sqrt{\Delta})/(2w_0)$$

- sequence describing calculation of a single root (for $\Delta=0$):

$S_1 =$ $a=w_0, b=w_1, c=w_2$; $x=-w_1/(2w_0)$

- the sequence valid for the case $\Delta<0$:

$S_2 =$ $a=w_0, b=w_1, c=w_2$; M_1

where M_1 is the message "($\Delta<0$) – no real solutions",

- sequence describing calculation of a solution for the case $w_2=0$:

$S_3 =$ $a=w_0, b=w_1, c=w_2$; $x=-w_1/w_0$,

- sequence describing calculation of a solution for the case $w_1=0$:

$S_4 =$ $a=w_0, b=w_1, c=w_2$; $x=\sqrt{(-w_2/w_0)}$,

- the sequence valid for the case $(-w_2/w_0)<0$:

$S_5 =$ $a=w_0, b=w_1, c=w_2$; M_2 ,

where M_2 is the message "($w_1=0$ and $(-w_2/w_0) < 0$) – no solution",

- sequence describing calculation of a solution for the case $w_0=0$ and $w_1\neq0$:

$S_6 =$ $a=w_0, b=w_1, c=w_2$; $x= -w_2/w_1$,

- sequence valid for the case $w_0=0$ and $w_1=0$:

$S_7 =$ $a=w_0, b=w_1, c=w_2$; M_3 ,

where M_3 is the message "($w_0=0$ and ($w_1= 0$)) – no quadratic equation".

B) Synthesis of eliminations.

Eliminating the sequences S_1 and S_2 under the condition $\Delta=0$ we obtain the uniterm

$$E_1 = \vdash S_1; S_2; (\Delta = 0) - ?\dashv$$

which concerns the solutions for $\Delta\leq 0$.

Eliminating S_0 and E_1 under the condition $\Delta>0$ we obtain

$$E_2 = \vdash S_0; E_1; (\Delta > 0) - ?\dashv$$

which covers the solutions for $\Delta>0$ and $\Delta\leq 0$.

Eliminating E_2 and S_3 under the condition $w_2\neq 0$ we obtain

$$E_3 = \vdash E_2; S_3; (w_2 \neq 0) - ?\dashv$$

which covers the solutions for $w_2\neq 0$ and $w_2=0$.

Eliminating S_4 and S_5 under the condition $(-w_2/w_0)\geq 0$ we obtain

$$E_4 = \vdash S_4; S_5; (-w_2/w_0 \geq 0)-?\dashv$$

which covers the solution for $(-w_2/w_0)\geq 0$ or communicates on the lack of solutions.

Eliminating E_3 and E_4 under the condition $w_1\neq 0$ we obtain

$$E_5 = \vdash E_3; E_4; (w_1 \neq 0) - ?\dashv$$

which covers the solution for $w_1\neq 0$.

Eliminating S_6 and S_7 under the conditions $w_1\neq 0$

$$E_6 = \vdash S_6; S_7; (w_1 \neq 0) - ?\dashv$$

produces the solution for $w_1\neq 0$ and $w_0=0$.

Finally, eliminating E_5 and E_6 under the condition $w_0\neq 0$

$$E_7 = \vdash E_5; E_6; (w_0 \neq 0) - ?\dashv$$

we obtain all the solutions to the equation under that condition.

C) Transformation of the algorithm

Accounting for the distributiveness property for the elimination operation, we extract the uniterms a=w_0, b=w_1, c=w_2 from the elimination symbol in E_1, thus yielding the minimized expression

E_{1min} =

a=-w_0

,

b=w_1

,

c=w_2

;

x=-w_1/(2w_0); M_1; (Δ=0) - ?

Proceeding in a similar way we can obtain minimized expressions for the remaining elimination operations E_2 through E_7, thus ending up with the following minimized version of the algorithm for solving the quadratic equation.

E_{7min} =

a=w_0

,

b=w_1

,

c=w_2

;

; S ; (w_0≠0)-?

; R ; (w_1≠0)-?

Δ=w_1^2 – 4w_1w_2; x=-w_1/w_0; (w_2≠0)-?

;

Z; E; (Δ>0) - ?

where

Z = x_1= (-w_1+Δ)/(2w_0)

;

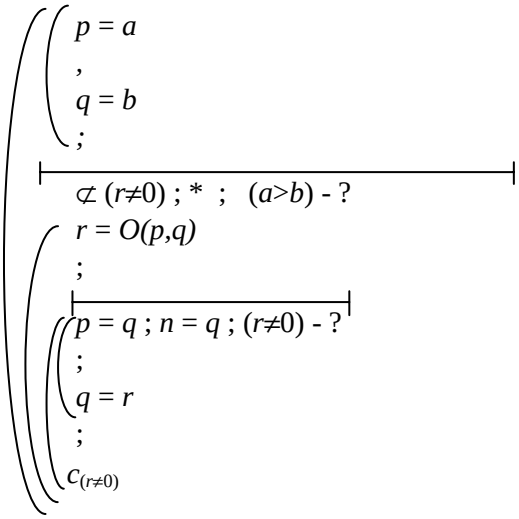
x_2= (-w_1-Δ)/(2w_0),

E = x=-w_1/(2w_0); M_1; (Δ=0) - ? ,

$$R = \overline{\vdash\quad x=\sqrt{(-w_2/w_0)};\ M_2;\ (-w_2/w_0 \geq 0) - ?\quad\dashv}\ ,$$

$$S = \overline{\vdash\quad x=-w_2/w_1;\ M_3;\ (w_1\neq 0) - ?\quad\dashv}\ .$$

Example 2. Construct an algorithm for calculation of the greatest common divisor for natural numbers a and b such that $a > b$. A classical, Euclidean solution to the problem has been analysed in a number of publications, to mention a recent contribution of Ref. [21]. Here we present a slightly modified solution, with the division operation substituted for the original subtraction one. In terms of our algebra of algorithms, the modified Euclidean solution can be expressed as



where $c_{(r\neq 0)}$ denotes a return to the start of the cycle.

When the condition $a > b$ is not fulfilled, the message „$a \leq b$" is printed and the execution of the algorithm is stopped. Otherwise, a remainder $r = O(p, q)$ from division is calculated and is compared with zero by means of $(r\neq 0)$ - ?. If the remainder is equal to zero, then q is the greatest common divisor ($n = q$), otherwise $p = q$ and $q = r$ and return to the start of the cycle.

5. CONCLUSIONS

In this paper, a number of the existing, intuitive computation models have been analyzed, including the virtual machines of Turing, Post, Kolmogorov, Schönhage, Aho-Ullman-Hopcroft as well as the Markov and Krinitski algorithms and the recursive functions. A new concept for the analysis of the resulting computation algorithms is proposed, which is based on creation of resources for description of *relations* between the algorithm components, rather than determination of finite sets of various components. When applying the concept to the existing computation models it has occurred that they do not contain a tool for description of sequences and parallel/nonparallel branches of the algorithm components. This has identified the reason for an intuitive, rather than formal description of the algorithms with the existing methods.

A universal notion for the algorithm component has been introduced, which is called a uniterm. Using the axiomatic method, various operations on the uniterms have been defined, thus contributing to the definition of an abstract or formal algorithm and the introduction of a new idea of the algebra of algorithms. The algebra provides tools for formal transformation and possible minimization of the

algorithms. Two examples illustrate the potential of the proposed theory and the underlying methodology for processing of the algorithms.

1. Turing A. M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of London Mathematical Society, series 2, vol. 42 (1936-1937), pp. 230-265; correction, ibidem, vol. 43, pp. 544-546. Reprinted in [13 Davis M., pp. 155-222] and available online at http://www.abelard.org/turpap2/tp2-ie.asp. 2. Kleene S.C.: Turing's analysis of computability, and major applications of it. In Rolf Herken, Editor, The universal Turing machine: A half-century story, Oxford University Press, 1988, pp. 17-54. 3. Post E. L.: Finite Combinatory Processes - Formulation 1. Journal of Symbolic Logic, 1, pp. 103-105, 1936. Reprinted in The Undecidable, pp. 289ff. 4. De Mol L.: Closing the circle: an analysis of Emil Post's early work. The Bulletin of Symbolic Logic, Vol. 12, Issue 02, June 2006, pp. 267 — 289. 5. Kolmogorov A.N.: On the concept of algorithm (in Russian). Uspekhi Mat. Nauk 8:4 (1953), pp. 175-176; translated into English in Uspensky V.A., Semenov A.L.: Algorithms: Main Ideas and Applications, Kluwer, 1993. 6. Kolmogorov A. N., Uspensky V.A.: On the definition of algorithm (in Russian). Uspekhi Mat. Nauk 13:4 (1958), pp. 3-28; translated into English in AMS Translations 29 (1963), pp. 217-245. 7. Gurevich Y.: Kolmogorov machines and related issues. In G. Rozenberg and A. Salomaa, Editors, Current Trends in Theoretical Computer Science, World Scientific, 1993, pp. 225-234; originally in Bull. EATCS 35 (1988). 8. Schönhage A.: Universelle Turing Speicherung. In J. Dörr and G. Hotz, Editors, Automatentheorie und Formale Sprachen, Bibliogr. Institut, Mannheim, 1970, pp. 369-383. 9. Schönhage A.: Storage modification machines. SIAM Journal on Computing, 9 (1980), pp. 490-508. 10. Markov A.A.: Theory of algorithms (in Russian). Editions of Academy of Sciences of the USSR, vol. 38, 1951, pp. 176-189; translated into English in American Mathematical Society Translations, 1960, series 2, 15, pp. 1-14. 11. Markov A.A., Nagorny N.M.: The Theory of Algorithms (Mathematics and its Applications). Springer, 2001. 12. Church A.: An unsolvable problem of elementary number theory. American Journal of Mathematics, vol. 58 (1936), pp. 345-363. 13. Constable R.L., Smith S.F.. Computational foundations of basic recursive function theory. Theoretical Computer Science, 121, pp. 89-112, Dec. 1993. 14. Sieg W. Step by recursive step: Church's analysis of effective calculability. The Bulletin of Symbolic Logic, 3:2 (1997), pp. 154-180. 15. Kleene S.C.: Origins of recursive function theory. Annals of the History of Computing, 3:1 (January 1981), pp. 52-67. 16. Aho A.V, Hopcroft J.E, Ullman J.D.: The design and analysis of computer algorithms. Addison-Wesley Publishing Company, 1974. 17. Krinitski N.A.: Algorithms around us (in Russian). Mir, Moscow, 1988; also translated to Spanish (Algoritmos a nuestro alrededor). 18. Uspensky V.A., Semenov A.L.: Algorithms: Main Ideas and Applications. Kluwer, 1993. 19. Owsiak V., Owsiak A., Owsiak J.: Theory of Abstract Algorithms and Mathematical Modelling of information systems (in Polish). Opole University of Technology Press, Opole, Poland, 2005. 20. Ovsyak V.: Algebra of Algorithms. Modern problems in radio engineering, telecommunications and computer science. Proceedings of the International Conference TCSET'2006. February 28 – March 4, 2006. L'viv – Slavske, Ukraine. Pp. 66-67. 21. van den Dries L., Moschovakis Y.N.: Is the Euclidean algorithm optimal among its peers? The Bulletin of Symbolic Logic, *Vol. 10, No. 3 (Sep. 2004), pp. 390-418.*