Computer Science Journal of Moldova, vol.**2**, no.3(6), 1994

Mathematical theory of multiprocessor control systems and its applications

G.Cejtlin E.Jushchenko

Abstract

The peculiar feature of cybernetic control systems is a feedback interaction with a controlled object. The abstract model of such an interaction may be based on the control and operating automata composition originating in the abstract computer model suggested by V.M.Glushkov [1]. The model functioning can be formalized by the technique of systems of algorithmic algebras (SAA). The paper is concerned with mathematical theory of multiprocessor control systems (MCS) based on the modified SAA technique and intended for solving the problems of formalization of the parallel processes semantics, their modification, optimization by selected criteria, in particular.

1 The abstract model of the control system

The abstract model of the control system consists of a composition of control automation U and operating automation P. The set $\tilde{\alpha}$ of logical values of conditions which characterize the current state of P being input and perceived, the automaton P outputs the operator A transferring P to a new state. By way of cyclic functioning according to the given scheme, U can attain the final state thus indicating the control process completion.

When using this model for MCS it is supposed that the number of channels for interaction with the control object coincide with the number of automata forming a part of the control structure of the given model. The modified SAA oriented to multiprocessing formalization [2] (see also [3,4]) are associated with abstract MCS models.

2 Regular schemes on abstract memory types (AMT)

The SAA $\langle \mathfrak{A}, \mathfrak{B} \rangle$ represent dual–basis algebraic system and consist of a set of operators $\mathfrak{A}$ and a set of logical conditions $\mathfrak{B}$ with determined signature of operations Ω. Operators are represented by mappings (possibly partial) of information set $\mathfrak{M}$ onto oneself and logical conditions are predicates over the set $\mathfrak{M}$ adopting truth values of three–valued logic $E_3 = \{0, 1, \mu\}$ where 0 is falsity, 1 is truth, μ is uncertainty. The signature of SAA operations $\Omega = \Omega_1 \cup \Omega_2$ consists of Ω_1 logical operations adopting the values in the set $\mathfrak{B}$ and the system Ω_2, i.e. operations adopting the values in the set of operators $\mathfrak{A}$. Graphical means of the modified SAA inherent to the signature Ω are compared to programmer control structures and operations of program logic in Table 1. A strict definition of SAA and their modifications oriented to formalizing parallel and non–determined program schemes and also a comparison of principal notions of SAA theory with program logics are given in [1–4].

It should be noted that the signature of SAA $\langle \mathfrak{A}, \mathfrak{B} \rangle$ contains operations corresponding to successive (A, B), distributive (**if** α **then** A **else** B), cyclic (**while** α **do** A) control structures. Thus, SAA means and their modifications represent a mathematical base for structural programming technologies [5].

Representation of operators from $\mathfrak{A}$ in SAA $\langle \mathfrak{A}, \mathfrak{B} \rangle$ by way of superposition of operations belonging to the signature Ω is named the regular schemes. In the regular schemes control is realized in forward direction only, from the left to the right; backward jumps are possible only for α–iterations from closing iteration brackets to opening ones. Despite such limitedness as compared to traditional schematics, V.M.Glushkov [1] proved the validity of the following statement.

Theorem 1 *An arbitrary algorithm or a program is representable in the SAA* $\langle \mathfrak{A}, \mathfrak{B} \rangle$ *by an equivalent regular scheme: a constructive procedure of regularization (reduction to the regular scheme) is developed for arbitrary algorithms and programs.*

Signature of operations of modified SAAs	Programmer structures	Signature of propositional program logics
Logical structures		
generalized conjunction	AND	Usual propositional
generalized disjunction	OR	connections
generalized negation	NOT	
left multiplication by operator $A\alpha$	—	$[A]\alpha$
composition $A \times B$	$A; B$	$A; B$
synchronous disjunction $A \vee B$	—	—
asynchronous disjunction $A \dot{\vee} B$	A, B	—
Operator structures		
non–determined disjunction $A \vert B$	—	$A \cup B$
α–disjunction ${}_\alpha A \vee B$	**if** α **then** A **else** B	$(\alpha?; A) \cup (\neg\alpha?; B)$
switch $\prod \begin{pmatrix} \alpha_1, \ldots, \alpha_k \\ A_1, \ldots, A_k \end{pmatrix}$	**CASE**	—
α–iteration $\{{}_\alpha A\}$	**while** α **do** A	$(\alpha?; A)^*; \neg\alpha?$
inverse α–iteration $\{A\}_\alpha$	**do** A **while** α	—
filtration $\underline{\alpha}$	—	$(\alpha?)$
synchronizer $S(\alpha)$	descent function	—

Table 1

Within the framework of multilevel structural design of MCS of interest are the regular schemes on AMT conceptually close to abstract data types widely spread in modern programming. Let $\prod(n)$ be a multidimensional memory space in whose integer vertices $\tilde{q}$ the cells $r_{\tilde{q}}$ are situated, assuming values from the set $\mathfrak{M}(r_{\tilde{q}})$, i.e. the alphabet of the given space. By AMT the system $\langle N, S\rangle$ is meant where $N \subseteq \prod(n)$ is a memory medium, $S = \{f_i(\tilde{A}, \tilde{X}) | \quad i = 1, 2, \ldots k\}$ is an access signature consisting of operators $f_i(\tilde{A}, \tilde{X})$ performing access to the memory and presented by regular schemes which depend on the sets $\tilde{A} = \{A_1, A_2, \ldots, A_m\}$ and $\tilde{X} = \{X_1, X_2, \ldots, X_m\}$ of elementary operators and conditions, respectively. The notion of AMT can be interpreted as a hypothetic machine whose instructions are represented in the form of regular schemes $f_i(\tilde{A}, \tilde{X})$ similar, in a certain sense, to microprograms. And what is more, elements of the sets $\tilde{A}$ and $\tilde{X}$ are interpreted as microinstructions. In this connection it is mentioning that initially the SAA were called the microprogrammed algebras [1]. AMT covers, in particular, the elastic tape which serves as a framework to classify the well–known memory structures: stack, queue, double ended queue, etc. The asynchronous multiprocessing and mark cells technique makes it possible to simulate AMT on the elastic tape, with AMT being a variety of combinations of the enumerated structures reflecting the characteristic features of standard data structures multiprocessing.

Heterogeneous periodically defined (HPD) transformations on abstract registers whose terms are used for formalization of synchronous multiprocessing semantics provide another and very essential type of AMT. Let $R = (r_i | \quad -\infty < i < \infty)$ be bilaterally infinite abstract register separated into sections each having n elements. HPD–transformation on register R is assigned by a combination of generating functions f_q and coefficients $k_q \qquad (q = 1, 2, \ldots, n)$. Each f_q determines new states of the q–th elements in each of these sections according to the states of elements occurring in the neighborhood. The latter is periodically defined over the whole register depending on coefficient k_q. A register transition to a new state as a result of HPD–transformation is carried out synchronously (in one cycle). For example, for $n = 1$ we obtain PD–transformation with the unique generating function f and

coefficient k which realizes homogeneous multiprocessing of contents of the register R.

Note the conceptual similarity of the PD–transformations to the Neumann–Church multidimensional automata, Moore honeycomb structures, interactive networks playing an important part in the theory of homogeneous structures. The further development of the theory of Post modified algebras [6] is related to the semi–group of HPD–transformations. Thus, using the terms of the regular schemes on AMT it is possible to formalize the operation of abstract models of MCS combining the asynchronous and synchronous strategies of multiprocessing, and in particular of abstract models of conveyor, rotating and building–berth multiprocessing ascending from the well–known industrial analogues. Introducing the dimensionality into the space $\prod(n)$ it is possible to obtain AMT of various configurations oriented towards application to problems of CAD, computer–assisted technological preparation of processes, machine graphics, etc.

3 "Writers–readers" problem

One of the popular problems of parallel programming is the "writers–readers" problem which consists in the following.

Several processors–writers record into the buffer and several processors–readers read the data out of this buffer. Simultaneous access to the buffer for any arbitrary number of readers is possible though only one writer can operate with it at the given time instant. Besides, the writers are granted with buffer access priority as compared to readers. Interaction between the writers–readers processes is formalized by the following $PRS: \quad RWP = \{{}_{\alpha}FQW\dot{\vee}FRR\dot{\vee}BA\}$ where α is a condition which becomes true after completion of the process; FQW is an operator of forming the queue of names of active writers, FRR is an operator of forming the register of names of the active readers; BA is an operator of buffer access. The operator FQW loads the queue of active register RR consisting of elements containing the names of active readers, so when the buffer is open for reading it is accessible to achieve readers through RR in a parallel way.

The operator $BA = \{_\beta READ(WQ) \times WRI_S BUF\} READBUF$; here β is a condition true when the queue of writers is empty; $READ(QW)$ is an operator of reading of the active writer name S out of QW; $WRI_S BUF$ is an operator of operation of the writer S with the buffer; $READBUF$ is an operator of access of all active readers from RR to the buffer.

Note that when the buffer is properly organized it is possible to solve the problem of dependencies between the writers–readers processes. For this purpose it will suffice to represent the buffer in the form of two queues BS_f and BS_u of addresses of filled in and unfilled pages, respectively. When access to the buffers is realized in the write mode the writer obtains the address of unfilled page from BS_u, fills it in and records the given address to BS_f. At the same time the writers read the address of the filled in page from vertex of BS_f and on emptying in input the address into the queue BS_u.

Such a buffer organization is an extension of the scheme of circular memory functioning suggested in [4].

Organization of more flexible interaction between the writers–readers processes can be realized through the scheme of dynamic exchange [4]. The "writers–readers" problem is very close to the concept of ports through which interaction between suppliers and consumers is realized, the concept common to asynchronous programming.

4 Structural parallel programming semantics

As it was earlier the SAAs and their modifications serve as a formal basis for the structural programming semantics. In SAA theory the problem of axiomatization is among the most important ones; it consists in the development of complete systems of identity relations characterizing the main properties of operations belonging to the signatures of SAAs and their modifications. Solubility of the equivalence problem of regular schemes for the corresponding classes of SAAs is an important corollary of finite axiomatization.

Let $\langle \mathfrak{A}_0, \mathfrak{B}_0 \rangle$ is a SAA with the signature of operations Ω_o determined by the following formulas: $\alpha \wedge \beta$, $\alpha \vee \beta$, $\overline{\alpha}$, $\beta = A\alpha$, $A \times B$, $(_\alpha A \vee$

$B)$, $\{_\alpha A\}$.

Solution for the axiomatization problem for SAAs $\langle \mathfrak{A}_0, \mathfrak{B}_0 \rangle$ and for their modifications is closely related to investigation of the three–valued algorithmic logic $\tilde{\mathfrak{A}}$ with the signature of operations $\alpha \wedge \beta$, $\alpha \vee \beta$, $\overline{\alpha}$, $\beta = A\alpha$. Methods of identical transformations of logical functions into equivalent q–polynomials, similar to DNF in the algebra of logic, were developed for $\tilde{\mathfrak{B}}$. Canonical forms of representation in $\tilde{\mathfrak{B}}$ are perfect q–polynomials $D_c = D^1 \vee D^\mu$ where $D^1 = \bigvee_{i=1}^k B_i^1$, $D^\mu = \bigvee_{j=1}^r K_j^\mu$ with B_i^1 being the generalized elementary conjunction assuming the value 1 on the corresponding sets of values of variables, and K_j^μ being the generalized constituent assuming the value μ on the corresponding sets. The following statement is valid [7].

Theorem 2 *An arbitrary three–valued function $\varphi \not\equiv 0$ is uniquely representable by the perfect q–polynomial in the logic $\tilde{\mathfrak{B}}$.*

Corollary 1. The problem of axiomatization with the use of the unique derivation rule, i.e. the traditional substitution, is solved for $\tilde{\mathfrak{B}}$.

Corollary 2. The problem of equivalence of logical functions is solvable for $\tilde{\mathfrak{B}}$.

Note that the generalized DNF in $\tilde{\mathfrak{B}}$ pertain to the perfect q–polynomials, so, unlike the algebra of logic, the following statement is correct.

Corollary 3. If the function in $\tilde{\mathfrak{B}}$ is representable by the generalized DNF then this representation is unique.

The ambiguity of representation of Boolean functions in DNF is known to be associated with the problem of minimization of Boolean functions in the algebra of logic.

The duality principle is formulated for $\tilde{\mathfrak{B}}$ which allows to carry out the transition from disjunctive forms of representation of logical functions to conjunctive ones and vice versa.

Control of multiprogram and multiprocessor computations is associated with the verification of conditions for completion of certain phases of information processing; such conditions were called closed [2].

Once the conditions become true, they remain so henceforth independently of subsequent development of the computational process. This fact determines their conceptual similarity to monotonous operators oriented towards solution of program semantics formalization problem.

By S–algebras are meant the modified SAAs $\langle \mathfrak{A}_1, \mathfrak{B}_1 \rangle$ with closed elementary logical conditions and signature Ω_1 of operations defined by the formulas $\alpha \wedge \beta, \alpha \vee \beta, \bar{\alpha}, A \times B, \underline{\alpha}, \{_\alpha A\}$. For the S–algebras $\langle \mathfrak{A}_1, \mathfrak{B}_1 \rangle$ the axiomatics Σ is developed with the unique rule of derivation, the substitution. This axiomatics underlies the apparatus of identities which is used to transform the regular schemes to the canonical form, i.e., to the standard polynomial C_F whose characteristic feature is the absence of iterative embedding. The following statement is valid [2].

Theorem 3 *An arbitrary regular scheme $F(\tilde{A}, \tilde{X})$ is uniquely representable by a standard polynomial $C_F, F(\tilde{A}, \tilde{X}) = C_F$ in the S–algebras $\langle \mathfrak{A}_1, \mathfrak{B}_1 \rangle$.*

Corollary 1. For the S–algebras $\langle \mathfrak{A}_1, \mathfrak{B}_1 \rangle$ the axiomatization problem is solved with application of the unique rule of derivation, the substitution.

Corollary 2. For the S–algebras $\langle \mathfrak{A}_1, \mathfrak{B}_1 \rangle$ the problem of the regular schemes equivalence is solvable.

The obtained results are extended to the case of modified SAAs oriented towards formalization of non–determined asynchronous processes. The paper [8] investigates the $S(H)$–algebras $\langle \mathfrak{A}_2, \mathfrak{B}_2 \rangle$ with signature Ω_2 of operations defined by the formulas $\alpha \wedge \beta, \alpha \vee \beta, \bar{\alpha}, A \times B, A|B, \underline{\alpha}, \{_\alpha A\}$. Extension of the signature of S–algebras $\langle \mathfrak{A}_1, \mathfrak{B}_1 \rangle$ by asynchronous means results in $\tilde{S}$–algebras treated in [2] (p.3). A number of secondary results were obtained within the framework of solution of the problem of axiomatization and equivalence for the $\tilde{S}$–algebras. These interesting results relate to solution of the problem of dead–ends and elimination of fictitious iterative structures.

Note that the problem of axiomatization of the algebra of regular events with application of the unique rule of derivation, the substitu-

tion, remains open for 25 years that proves the non–triviality of the obtained results.

The locally closed conditions are the natural generalization of the notion of closure. Let us determine the relation of the order $\succ$ on the set of elementary logical conditions, such that the validity of implication $\alpha \to \beta$ stems from $\alpha \succ \beta$. Condition β is locally closed if it may change the value 1 to 0 only as a result of a jump from the closing iterative bracket to the corresponding opening one in an arbitrary α–iteration such that $\alpha \succ \beta$, retaining its value 1 after completion of this α–iteration. The local closure ensures assignment of initial values to logical variables which organize interaction between processes in the body of the cycle at each access to this body. The modified SAAs with the locally closed elementary logical conditions are called the $S(l)$–algebras. An apparatus of identities was developed for the $S(l)$–algebras which includes the identical relations, true in the S–algebras, and the problems of axiomatization and equivalence were solved. The obtained results permit to establish the validity of the following statement.

Theorem 4 *Direct and inverse formal transformation of synchronous regular schemes into asynchronous ones is feasible in the $S(l)$–algebras.*

In connection with the development of structural schematics it is of interest to conduct a comparative analysis of the descriptive power of SAA facilities and of their modifications with respect to Janov schemes underlying the traditional non–structural schematics [9].

By the right–hand recursion is meant the operation $P(G, H)$ specified by the following set of equations:

$$
\begin{aligned}
&Z_1 \to ({}_{\alpha_1}G \vee A_1({}_{\beta_1}B_1Z_1 \vee C_1Z_2));\\
&Z_2 \to ({}_{\alpha_2}D_1Z_1 \vee A_2({}_{\beta_2}B_2Z_2 \vee C_2Z_3));\\
&\dots\\
&Z_{n-1} \to ({}_{\alpha_{n-1}}D_{n-2}Z_{n-2} \vee A_{n-1}({}_{\beta_{n-1}}B_{n-1}Z_{n-1} \vee C_{n-1}Z_n));\\
&Z_n \to ({}_{\alpha_n}D_{n-1}Z_{n-1} \vee A_n({}_{\beta}D_n \vee H));
\end{aligned}
$$

where $Z_1, Z_2, \dots, Z_n$ are metavariables.

Let $\langle \mathfrak{A}_3, \mathfrak{B}_3 \rangle$ be modified SAAs with the signature Ω_3 of operations defined by the formulas $\alpha \wedge \beta, \alpha \vee \beta, \bar{\alpha}, A \times B, P(G,H), \{_\alpha A\}$. Let us denote by R_i the class of operators representable by regular schemes in the modified SAAs with the signature Ω_i, respectively, $(i = 0, 1, 2, 3)$.

The axiomatics Δ_i is called transformationally complete, $\Delta_i\,[S \to R_i]$ if there exists a constructive procedure of regularization of Janov schemes based on the axiomatics Δ_i (construction of the equivalent regular scheme of the class R_i according to an arbitrary Janov scheme) where S is a class of operators representable by Janov schemes.

Theorem 5 *For the modified SAAs with signature Ω_i there exists transformationally complete axiomatics $\Delta_i\,[S \to R_i]$ $(i = 0, 1, 2, 3)$.*

Corollary. Strict inclusion $S \subset R_i$ is valid for any $i = 0, 1, 2, 3$. Thus, SAAs and their modifications considered earlier exceed formalisms of Janov schemes with respect to graphical possibilities. Interchangeability of operations determined by the formulas $\beta = A\alpha, A \vee B, A|B, P(G,H)$ with respect to Janov schemes stems from the theorem 5 which makes it possible to optimize logical schemes of programs taking account of peculiarities inherent in computing resources. Note that the process of Janov schemes regularization essentially simplifies substantiation of their correctness.

The obtained results are used for optimization with respect to the length of the linked cycles of standard non–structural constructions, parallel algorithms of multilayer sorting, syntactical analysis, translation, partial program verification, etc. Axiomatization of SAAs based on application of the unique derivation rule, substitution, allows for automating the process of analytical transformations of the regular schemes of ANALYST package implemented in ANALYTIC and L2B languages (see review [10]). It should be noted that enumerated languages contain the operator APPLY RELATION that essentially simplifies implementation of the given package.

5 Tools for the method of multilevel structural program design (MSPD)

The suggested formal means can form the basis of design technology for MCS. With this in mind, the method MSPD was developed at V.M.Glushkov Institute of Cybernetics of the Ukrainian Academy of Sciences. The method is based on the program structure formalization being designed in terms of regular schemes on AMT in combination with the chosen strategy of multilevel design: descending, ascending or combined one [10]. The use of the set of regular schemes of programs permits to coordinate the MSPD method with well–known programming technologies and creates possibilities to apply this method at the upper stages of programs and algorithms development which allows to use later well developed means for data structures formalization as well as language and program means characteristic to the given technologies. It should also be noted that the MSPD method as compared to program logic, flowchart means, EKZEL and PDL languages is characterized by complex way of usage within the framework of the unique formalism of structurization, parallelism, memory standartization, developed means of formal transformations and the necessary set of facilities. Due to the correspondence of graphical means of SAA to the main programmer constructions it is possible to jointly design algorithms and programs by the MSPD method which is comparable with respect to its significance to the joint design of computer hardware and software.

Main principles of program development by the MSPD method are implemented in the MULTIPROCESSIST system. A coordinated family of the system input languages is based on the set of regular program schemes, i.e., the SAA–schemes, and is associated with main stages of designing the programmed product. Thus, the control program structure is formalized in terms of the language of logical SAA–schemes. The concept of AMT underlies the language of SAA–schemes on memory. Interpretive languages are intended for detalization of operators and conditions of SAA–schemes in conformity with specific features of the implemented algorithms. Transition from interpreted SAA–schemes

to the base language program is realized by way of recording and assembling to the given scheme. The given modules form syntactically correct fragments of a program; the subsequent recording which can be carried out automatically results in generating the syntactically correct program in the base language. At present, the first stage of MULTIPROCESSIST system (MLT–1), i.e., the program synthesizer in the base language using the description of programs in the language of SAA–schemes (SAA–1) was realized. The text in the SAA–1 language (logical SAA-scheme) represents documentation for the control structure of the created program. The peculiar features of the SAA–1 language for various ways of development of syntactically correct and graphically clear SAA-schemes are:

- multilevel structure of the text of the SAA–scheme representing the list of equations situated in such a way that each subsequent equation specifies by SAA means some operators and conditions whose identifiers occur in one of the previous equations;
- the presence of various forms of syntactic representation of one and the same object and language operations;
- an arbitrary length of identifiers, operators and conditions and the possibility of random repetition of the definite symbols of alphabet which makes it possible to use the chains of these symbols for partitioning the scheme text.

The software of the synthesizer MLT–1 is intended, firstly, for automatic output of the control program structure from its SAA–scheme (and here, the given control structure corresponds to its documentation at any moment of time) and, secondly, for subsequent automated program design through attaching to its control structure the elementary operators and conditions of the initial schemes from the file of the design program which are realized in the base language. (The file contains libraries of the initial and object modules.) The synthesizer MLT–1 is oriented to the class of problems characterized by complex logic and a relatively simple data structures. Parametric character of the system consists in the possibility of tuning it to the class of input

and output (base) languages. Tuning of the system to an interpreted input language of SAA–schemes intended for formalization of a certain class of algorithms is performed by means of library composition reflecting the specialization of the input language. Switching of the new base language to the system is performed by way of positioning the switches which control, at the phase of semantic interpretation, the process of assignment of the corresponding semantic modules. At present the base languages of the system are PL–1, ASSEMBLER and PASCAL.

The abstract model of MCS was employed to represent the worker–machine tool system that underlies the automated system TPP TEKHNOLOG, based on MLT–1 and designed to develop processes of technological preparation of production by MSPD method. With this end in view, a version of the system TEKHNOLOG was worked out that depends upon the MLT–1 orientation towards technological processes of sheet–metal stamping. Problems of choice of the sequence of technological operations and problems of estimation of process characteristics were dealt with. The level–by–level analysis of the above problems by the MSPD method resulted in the elaboration of algorithms of their solution for one type of parts and technological operations. A language adopted for the special purpose of design of technological processes on the base of the MLT–1 library provided with corresponding program modules is realized. This enables the programs of design of technological processes of the class under consideration to be synthesized automatically. The library is composed of modules of the choice of the next elementary operation of the type of piercing, bending, cutting, center–popping, the estimation of the technological parameters of execution of these operations, the development of some technological transitions and the process as a whole on the basis of elementary operations of forming the input technological documents. The synthesized programs rely on the drawing of a part as well as instructions of the product engineer and allow constructing a flowchart for the processes of sheet–metal stamping. It should be noted that the distinctive features of the class of parts at hand, of the working conditions and the technological decision making are formalized in terms of the SAA schemes which en-

dow the elaborated algorithms with transparency and provide a means for their convenient modification with a change of a class of problems to be solved. The described modules were also synthesized in MLT–1 when the system library composed of functional program modules performing the search for objects of a given type in the description of part drawing, the editing of texts of input documentation, etc. The synthesis of functional modules, in its turn, was executed through the use of the system libraries with the modules of data base look–up, of operation with stacks and queues.

Thus, the elaborated facilities enable the problem–oriented functional and the system programming products to be synthesized automatically which fact attests to their flexibility and efficiency. The total volume of the developed technological library components of the MLT–1 system reaches up to 2000 operators of PL–1.

The MSPD method was also applied to develop some components of AMS, CROSS–systems of software support for special–purpose mini– and microcomputers, and monitor operating systems for HCS. Experimental employment of these facilities in practice has shown the possibility of 3 to 4 times speedup in development of program product and enhancement of its quality owing to the developed apparatus of formal transformation of programs in SAAs. Let us note also, that segmentation of the program, being developed by the MSPD method, is carried out during the multilevel design, compilation and catalogueing of program modules.

In the nearest future the MSPD facilities will be enhanced through the development of a flexible archive with problems. system, and personal user's libraries, the development of an interpreter system for editing and debugging the program product as part of the process of its design, the elaboration of methods and system of training in the MSPD method.

References

[1] V.M.Glushkov. Theory of Automata and Formal Transformations of Microprograms. Kibernetika, 1975, No.5, pp.1–10

[2] G.E.Cejtlin. Problem of Identity Transformations of Structured Program Schemata with Closed Logical Conditions. Kibernetika, 1978, No.3, pp.50–57; 1979, No.4, pp.10–18; No.5, pp.44-51

[3] V.M.Glushkov, G.E.Cejtlin, E.L.Jushchenko. Algebra. Sprachen. Programmierung. Akademie–Verlag, Berlin, 1980, p.340

[4] V.M.Glushkov, G.E.Cejtlin, E.L.Jushchenko. Methods of Symbol Multiprocessing. Kiev, Naukova Dumka, 1980, 252 p.

[5] R.Linger, H.Mills, B.Witt. Structural Programming: Theory and Practice. Moskva, Mir, 1983, 406 p.

[6] G.E.Cejtlin. The Theory of the Modified Post Algebras and Multidimensional Automata Structures. Lecture Notes in Computer Science, 1975, No.32, pp.418–424

[7] G.E.Cejtlin. Schematics of Structural Parallel Programming and its Applications. Mathematical Institute, Czechoclovak Academy of Olomous MFCS'79. Lecture Notes in Computer Science, Vol.74, 1979, pp.474–481

[8] Ju.A.Jushchenko. Problem of Identity Transformations of Structured Program Schemata with Closed Logical Conditions. Kibernetika, 1982, No.2

[9] A.P.Ershov. State–of–the–Art of Theory of Program Schemata. Problemy Kibernetiki, 1974, issue 27, pp.87–111

[10] V.M.Glushkov, G.E.Cejtlin, E.L.Jushchenko. Multilevel Structural Program Design: Method Formalization, Spectrum of Applications. Kibernetika, 1981, No.4, pp.46–60

G.E.Cejtlin, E.L.Jushchenko
V.M.Glushkov Institute of Cybernetics
Ukrainian Academy of Sciences,
Kiev, 252207, Ukraina

Received 30 June, 1994