Computer Science Journal of Moldova, vol.**3**, no.2(8), 1995

Structural synthesis of parallel programs (Methodology and Tools)

G.Cejtlin E.Jushchenko

Abstract

Concepts of structured programming and propositional program logics were anticipated in the systems of algorithmic algebras (SAAs) introduced by V.M.Glushkov in 1965. This paper correlates the SAA language with the known formalisms describing logic schemes of structured and unstructured programs. Complete axiomatics is constructed for modified SAAs (SAA–M) oriented towards the formalization of parallel computations and abstract data types. An apparatus formalizing (top–down, bottom–up, or combined) design of programs is suggested that incorporates SAA–M, grammar and automaton models oriented towards multiprocessing. A method and tools of parallel program design are developed. They feature orientation towards validation, transformations and synthesis of programs.

1 Systems of Algorithmic Algebras and Their Modifications

The distinguishing feature of sequential and parallel control systems consists in the presence of the controlled object feedback. The basis of the abstract control model (ACM) $\langle U, P\rangle$ consists in the following scheme of interaction between control structure U and operating structure P of automaton structures. Sets of values and predicates which characterize the current state of operating structure P enter the inputs of automata U. According to these sets automata from U generate operators which transform the operating structure P into a new state.

Interaction between structure U and structure P according to the given scheme continues up to the transition of all automata from U into finite states. A set $\mathfrak{M}$ of all states of P will be called the information set.

The ACM $\langle U, P\rangle$ is associated with two–sorted algebraic systems $\langle \mathfrak{A}, \mathfrak{B}; \Omega\rangle$ where $\mathfrak{A}$ is a set of operators, $\mathfrak{B}$ is a set of predicates (three-valued, in the general case), $\Omega = \Omega_1 \cup \Omega_2$ is a signature of operations. Operators: $A : \mathfrak{M} \to \mathfrak{M}$ are partial mappings of $\mathfrak{M}$ into itself; predicates $\alpha : \mathfrak{M} \to E_3$ where $E_3 = \{0, \mu, 1 : 0(falsity), \mu(uncertainty), 1(truth)\}$. Ω_1 consists of operations generating elements from $\mathfrak{B}$: generalized Boolean operations — disjunction $(\vee)$, conjunction $(\wedge)$, negation $(\neg)$ which are defined on binary sets as their Boolean analogs; the disjunction is commutative $1 \vee \mu = 1, 0 \vee \mu = \mu$; the conjunction is defined in the dual manner; $\bar{\mu} = \mu$; the left–hand multiplication $\beta = A\alpha$ of condition α by operator A for any $m \in \mathfrak{M}$ means $\beta(m) = \alpha(m')$ where $m' = A(m)$, i.e. the value of the predicate β coincides with the value α after application of operator A. By the given operation it is possible to forecast the computation process on the ACM and to form verification post–conditions in program logics. Ω_2 consists of operations generating elements from $\mathfrak{A}$: composition $A \times B$ — sequential application of operators $A, B \in \mathfrak{A}$; α–disjunction (alternative structure)

$$[\alpha](A \vee B) = \begin{cases} A & \text{for } \alpha = 1; \\ B & \text{for } \alpha = 0; \\ N & \text{for } \alpha = \mu, \end{cases}$$

where N is an indefinite operator; α — iteration (cyclic structure) ${}_{\alpha}\{A\}$ consisting in the iterative application of the operator A for false α until α becomes true.

Systems $\langle \mathfrak{A}, \mathfrak{B}; \Omega\rangle$ were called systems of algorithmic algebras (SAAs), and superpositions $F(\Sigma) \in \mathfrak{A}$ of operations from Ω over the basis $\Sigma \in \mathfrak{A} \cup \mathfrak{B}$ consisting of elementary operators and predicates — regular schemes (RS) of algorithms and programs. In RS the control is transferred only forward in the direction from the left to the right, jumps backwards are possible only in α–iterations from closing iterative brackets to the corresponding opening ones. Despite the given restriction the following assertion is valid [1].

Theorem 1 *In arbitrary algorithm (or program) is representable in SAA* $\langle \mathfrak{A}, \mathfrak{B}; \Omega \rangle$ *by the equivalent regular scheme; a constructive procedure of regularization — of reducing the arbitrary algorithms and programs to their regular representation — is constructed.*

Thus, the SAA apparatus may be considered as mathematical foundation of the technology of structural programming [2].

To formalize structured parallel processes the modified SAAs (SAA–M) $\langle \mathfrak{A}, \mathfrak{B}; \tilde{\Omega} \rangle$ are suggested with signature $\tilde{\Omega} \supset \Omega$ containing additionally the following operations:

- filtration $\underbracket{\alpha} = [\alpha](E \vee N)$, where E is identical and N is indefinite operators, which generates operators–filters $\underbracket{\alpha}$ adequate to Pratt tests and Dijkstra sides [2];

- synchronous disjunction $A \vee B$ where operators A and B produce similar results or one of them becomes indefinite, then the other completes computations;

- nondeterministic disjunction $A \mid B$ consisting in nondeterministic application of operators A and B;

- asynchronous disjunction $A \dot{\vee} B$ where operators A and B function in parallel on non–intersecting substructures of operational structure P; the functioning of parallel branches is synchronized by synchronizers $S(\alpha)$ and control points $T(\alpha)$. The synchronizer $S(\alpha)$ is the iteration by condition α which becomes true at the moment of attaining one of the control points $T(\alpha)$. They are placed on branches parallel to the branch containing $S(\alpha)$. In more detain the apparatus SAA–M is given in [3,4].

The Table 1 correlates operations entering into the signature $\tilde{\Omega}$ of SAA–M with corresponding program constructions and operations of program logics.

Example 1 *Let us consider the process $\overset{\rightarrow\!\times\!\leftarrow}{D}$ of asynchronous symbol–by–symbol counter–processing of the chain $x = a_1 a_2 \ldots a_n$. Pointers $\rceil$and $\lceil$ which fix the phases of two–way processing move from opposite direction along x. Introduce predicates*

$$\alpha_1 = \begin{cases} 1, & \textit{if } lr = 0; \\ 0 & \textit{otherwise} \end{cases}$$

and

$$\alpha_2 = \begin{cases} 1, & \textit{if } lr \leq k; \\ 0 & \textit{otherwise} \end{cases}$$

where lr is the distance between pointers, k is the coefficient of synchronization.

Operator $\overset{\rightarrow\!\times\!\leftarrow}{D}$ is representable in the SAA by the following RS

$$\overset{\rightarrow\!\times\!\leftarrow}{D} =_{\alpha_1} \{\overrightarrow{V}\}T(\alpha_1)\ \dot{\vee}\ _{\alpha_2}\{\overleftarrow{V}\}S(\alpha_1)$$

where $\overrightarrow{V}$ ($\overleftarrow{V}$) is the operator of the processing of the symbol directly from the right (left) of the pointer $\rceil$($\lceil$) with subsequent shift of the given pointer by one symbol to the right (left, respectively).

2 Schematology of Structured Parallel Programming

In the theory of SAA–M among the most important problems is the problem of axiomatization consisting in the development of complete systems of identical relations which characterize the main properties of operations entering into signatures of SAA–M. Solvability of the problem of RS equivalence for corresponding classes of SAA–M is one of principal consequences of finite axiomatizability.

Solution of the problem of axiomatization of SAA–M is closely connected with the study of three–valued algorithmic logic $\tilde{\mathfrak{B}}$ with a signature consisting of generalized Boolean operations and left–hand multiplication $\beta = A\alpha$. An apparatus of identical relations and a procedure of reducing logical functions to equivalent q–polynomials (analogs of

DNF in the algebra of logic) is developed for $\tilde{\mathfrak{B}}$. Perfect q–polynomials $D_c = D_1 \vee D_\mu$ where

$$D_1 = \bigvee_{i=1}^{r} \tilde{V}_i; \quad D_\mu = \bigvee_{j=1}^{s} K_j^\mu; \quad \tilde{V}_i = V_{i_0} \wedge [\wedge_1 V_{i_1}] \wedge \ldots \wedge [\wedge_k V_{i_k}];$$
$$\tilde{K}_j^\mu = K_{j_0} \wedge [B_1 K_{j_1}] \wedge \ldots \wedge [B_l K_{j_l}];$$

V_{ip} are elementary conjunction, K_{jt} are constituents μ in which the factors of the form $x \wedge \bar{x}$ $(p = 0, 1, \ldots, k;\ t = 0, 1, \ldots, l)$ are admissible, serve as canonical forms in $\mathfrak{B}$. Note that the laws of the excluded middle $x \vee \bar{x} = 1$ and contradictions $x \wedge \bar{x} = 0$ are not satisfied in $\tilde{\mathfrak{B}}$. The following assertion is valid [5].

Theorem 2 *In logic $\tilde{\mathfrak{B}}$ an arbitrary three-valued function $f \not\equiv 0$ is uniquely representable by the perfect q–polynomial, $f = D_c$.*

For the logic $\tilde{\mathfrak{B}}$ the problems of axiomatization and equivalency are solved; the duality principle for transfer from disjunctive forms of representations to conjunctive forms and vice versa is formulated.

Multiprocessor and multiprogram computation control is connected with checking the conditions of completion of particular stages of data processing. Similar conditions were called closed [3]. On becoming true, they continue to be true independent of the further development of computation process; this defines their conceptual closure to monotonous operators oriented towards solution of the problem of semantics formalization and to verification predicates suggested by Lamport [6] in connection with the problem of justification of correctness of parallel programs.

By $\tilde{S}$ algebras are meant SAA–M $\langle \mathfrak{A}, \mathfrak{B}; \tilde{\Omega} \rangle$ with closed elementary predicates entering into the basis Σ. Local closure of conditions is a natural generalization of the property of closure; such conditions can change the value 1 for 0 only at the moment of entering into the body of α–iteration at its next loop. Such conditions are applicable for organization of the functioning of the body of a cycle and, in particular, for the interaction of parallel branches in the cycle. The SAA–M with locally closed base predicates were called $\tilde{S}(L)$–algebras. The apparatus of identical relations and constructive procedures of reducing the RS to canonical forms, standard parallel polynomials (SPP) were developed for $\tilde{S}$–algebras and $\tilde{S}(L)$–algebras.

Criteria of clinch and of a fictitious character of iterative structures in RS are defined; efficient algorithms of testing the indicated properties are developed.

Theorem 3 *In classes of $\tilde{S}(L)$ algebras oriented towards deterministic, nondeterministic and asynchronous computations an arbitrary RS $F(\Sigma)$ is uniquely reducible to SPPs. Direct and inverse formal transformations of synchronous RS into asynchronous are implementable.*

An important consequence of the obtained results is solution the problems of axiomatization and equivalency for the considered classes of SAA–M.

In the framework of the development of schematology of structured programming of interest is the comparison of representational power of the RS language with the known formalisms for description of logical structure of programs: Dijkstra schemes, Janov schemes, graph–schemes of algorithms, etc. Consider $\langle\mathfrak{A},\mathfrak{B};\Omega\rangle$ under the assumption that predicates from $\mathfrak{B}$ are two–valued. Let $\langle\mathfrak{A},\mathfrak{B};\Omega_0\rangle$ be the SAA–M with signature $\Omega_0 \supset \Omega$ incorporating the operation $\Pi(\alpha)_{\downarrow m}$ of the jump to the label m for $\alpha = 0$, and for $\alpha \neq 0$ the operations $\Pi(\alpha)_{\downarrow m}$ generates operator E. In particular, $\alpha \equiv 0$ we have the unconditional jump to label m. Here the labels mark some operator occurrences in the program schemes. As usual, properties of signature operations are formalized by relations $\mathcal{F}(\Sigma) = G(\Sigma)$ among which the principal properties, axioms, are fixed. Let $\mathcal{X}$ and $\mathcal{X}'$ be some classes of program schemes. The axiomatic system $\mathcal{P}$ will be called transformationally complete with respect to classes $\mathcal{X}$ and $\mathcal{X}'$, $\mathcal{P}[\mathcal{X} \to \mathcal{X}']$ if there exists a constructive procedure of translating type, based on $\mathcal{P}$, which transforms the arbitrary scheme $\mathcal{F} \in \mathcal{X}$ into functionally equivalent scheme $G \in \mathcal{X}'$. Consider signatures $\Omega', \Omega'' \subset \Omega_0$ incorporating jump operation $\Pi(\alpha)_{\downarrow m}$ and, respectively, α–disjunction and α–iteration in addition to Boolean operations and composition. Superpositions $G(\Sigma)$, $H(\Sigma)$ of operations from Ω' and Ω'' will called nonstructured schemes (NS) and

Table 1

Signature of SAA–M operations	Program constructions	Signature of operations of program logics
	Logical structures	
Generalized conjunction	AND	Usual propositional links
Generalized disjunction	OR	
Generalized negation	NOT	
Left multiplication	is absent	$[A]\alpha$
	Operator structures	
Composition	$A;\ B$	$A;\ B$
Synchronous disjunction		
Asynchronous disjunction	$A,\ B$	
Nondeterministic disjunction $A \mid B$		$A \cup B$
α–disjunction	if α then A else B	$(\alpha?; A) \cup (\rceil\alpha?; B)$
Switch $\Pi\binom{\alpha_1,\ldots,\alpha_k}{A_1,\ldots,A_k}$		
α–iteration ${}_{\alpha}\{A\}$	while α do A	$(\alpha?; A)^*; \lceil\alpha?$
Inverse iteration $\{A\}_{\alpha}$	do A while $\bar{\alpha}$	
Filtration $\underbrace{\alpha}$		$(\alpha?)$
Synchronizer $S(\alpha)$	Descending function	WAIT

Dijkstra schemes, respectively. Denote by $\mathcal{R}, \mathcal{G}, D$ the classes of operators representable by RS, NS, Dijkstra schemes, respectively.

Theorem 4 *Constructed are transformationally complete axiomatics* $\tilde{\Delta}[\mathcal{G} \to \mathcal{R}]$ *and* $\tilde{\Delta}'[D \to \mathcal{G}]$*; here the following strict inclusions hold:* $D \subset \mathcal{G} \subset \mathcal{R}$.

As distinct from the known results by Boehm–Jacopini [7] the suggested structurization procedures relate to the level of logical program schemes and do not require memory structures and an assignment operator. The apparatus of relations developed in the SAA–M makes possible the structurization and optimization of chained cycles — standard nonstructured constructions [8] which are a source of many errors in composing the program in nonstructured manner when they are treated intuitively. The conducted study results in the following:

1. Program design languages based on the SAA–M apparatus are richer in representational power than formalisms based on NS, and the latter outperform the languages based on Dijkstra schemes.

2. Formalization of program semantics in terms of RS improves the understandability of both small and large programs being designed.

3. Using the apparatus of relations developed in the SAA–M makes it possible to optimize the designed programs by obtaining their structured representations no less compact than their nonstructured equivalents.

4. Formalization of semantics of standard nonstructured constructions in terms of RS permits to use them where it is justified in combination with structured constructions that provides the necessary flexibility of the process of program design.

The obtained results essentially refine the concept of structured go to programming suggested by Knuth [9]. It should be noted that all results obtained in the given section are based on axiomatics including parametric schemes of axioms and the unique derivation rule — substitution. The solution of the axiomatization problem in such substitution is of interest by the following reasons: 1) the use of substitution as the unique derivation rule makes possible the complete investigation of properties of α–iteration — one of the most interesting operations entering into SAA–M signatures; 2) the substitution is implemented in a number of systems of analytical transformations that permits automating the process of formal transformation of RS; 3) the top–down, bottom–up, or combined program design strategies are formalized by substitution. Note that the problem of axiomatization for the algebra of regular events (the simplified prototype of SAA) with the use of schemes of axioms and the unique derivation rule, substitution, so far remains open that confirms the nontriviality of the obtained results.

The SAA–M apparatus was approved on a number of characteristic problems of symbol multiprocessing: reader–writers, arrow problem, dynamic exchange, parallel sorting, syntactical analysis, translation, etc.

3 SAA–M and Abstract Types of Data (ATD)

The SAA–M apparatus makes it possible to formalize schemes of data structures (data descriptions) intended to generate and recognize (to check) these structures during the multiprocessing. Let S be a totality of elementary data structures, W be an alphabet of separators. Consider a semi–group $F(T)$ with a unit e over the united alphabet $T = S \cup W$. Elements of the semi–group $F(T)$ are formatted data structures representing sequences of elementary structures and separators. The set $O \subseteq F(T)$ will be called an object. On the elements of the semi–group $F(T)$ we define conditions–predicates (three-valued, in the general case) each of which is associated with objects $O^1(\alpha), O^0(\alpha), O^\mu(\alpha)$ respectively, domains of truth, falsity and uncertainty of the predicate α. Two–sorted algebraic system $\langle \mathcal{O}, \mathfrak{B}, \overset{\approx}{\Omega} \rangle$ will be called algorithmic algebras of data structures and will be denoted by SAA–M/DS where $\mathcal{O}$ is a set of objects, $\mathfrak{B}$ is a set of predicates, $\overset{\approx}{\Omega}= \Omega_1 \cup \Omega_2$; here Ω_1, Ω_2 are totalities of operations generating elements from sets $\mathfrak{B}$, $\mathcal{O}$, respectively. Ω_1 consists of generalized Boolean operations defined as in SAA; the domains of truth, falsity and uncertainty of composite predicates are specified by superpositions of set–theoretic operation of union, intersection and difference; by operation $\beta = O\alpha$ of left–hand multiplication of predicate α by object O, which consists in forecasting the process of data scanning in the chosen direction. Thus in the left–side scanning by $\beta = O\alpha$ the right–hand context is looked over. The operation $\beta = O\alpha$ is associated with the operation $\beta = \alpha O$ of the right–hand multiplication of predicate α by object O to trace the history of the process of data scanning in the chosen direction. Thus, in the left–side scanning of the operation $\beta = \alpha O$ the scanning of the left–hand context is adequate. Operations $\beta = O\alpha$ and $\beta = \alpha O$ find extensive applications in solution of problems of symbol processing with flexible context control (syntactical analysis, translation, editing, etc). Generation and recognition of data structures will be called scanning. Ω_2 incorporates operations: composition $O_1 \times O_2$ semi–group multiplication of objects, $O_1 \times O_2 = \{k_1 k_2 \mid k_1 \in O_1, k_2 \in O_2\}$; α–disjunction $[\alpha](O_1 \vee O_2)$ — the Hoare discriminated union [10]; α–iteration ${}_{\alpha}\{O\}$ consisting in a semi–group multiplication of object O by itself until α becomes true. Note, that like SAA and SAA–M/DS, among derivatives are operations of inverse α–iteration $\{O\}_{\alpha}$, cycle *do O while α do* and its generalizations. It should be emphasized that in the design of data structures in the process of their detailing the operation of composition may be interpreted by semi–group operations of Cartesian product, concatenation, multiplication of languages, etc.

Let us fix the basis $\tilde{\Sigma} = \tilde{\mathcal{O}} \cup \tilde{\mathcal{X}}$ where $\tilde{\mathcal{O}} = \{O_i \mid i = 1, 2, , \ldots, m\} \subseteq \mathcal{O}$ is a set of elementary objects, $\mathcal{X} = \{\alpha_j \mid j = 1, 2, \ldots, n\} \subseteq \mathfrak{B}$ is a set of elementary predicates. Superposition of operations from $\overset{\approx}{\Omega}$ over basis $\tilde{\Sigma}$ representing some object from $\mathcal{O}$ where $\tilde{q} = \{\tilde{\Pi}, \tilde{P}\}$ is the indication of direction and mode of scanning: $\tilde{\Pi} = \{\overrightarrow{\Pi}, \overleftarrow{\Pi}\}$, $\tilde{P} = \{\overrightarrow{P}, \overleftarrow{P}\}$ in the left–side, right–side generation, respectively, recognition of formatted data structures by the scheme $F_{\tilde{q}}(\tilde{\Sigma})$ will be called an object regular scheme (ORS).

In addition to the above–listed operations, Ω_2 incorporates the following operations: filtration $\underbracket{\alpha} = [\alpha](E \vee N)$ where $E = \{e\}$ is an identity object, N is undefined object; projection $\Pi P^q(\alpha)$ is excision according to the condition α of objects $O^q(\alpha) \quad q = 0, 1, \mu$; synchronous disjunction — the operation adequate to the operation of data structures superposition, $O \vee O' \subseteq O \cup O'$; nondeterministic disjunction $O \mid O'$ — set–theoretic union of objects O and O'; asynchronous disjunction $O \dot{\vee} O'$ — parallel scanning of non-intersecting data structures O and O'. As in the operator case, in the asynchronous interaction between branches in ORS the synchronizers $S(\alpha)$ are used with synchronization condition which fix the moment of passing through check points $T(\alpha)$ placed along parallel branches of ORS. Note, that if the composition is interpreted as the Cartesian product by projection the n–relations, totalities of trains satisfying the corresponding predicates, can be formed. Thus, various data structures can be represented in SAA–M/DS (tables, matrices, graphs, trees, multivariate structures, etc) and the processes of parallel scanning of data structures, in particular, those oriented towards two–way scanning (see, e.g., s.1) and its multilayer generalization [11].

The apparatus of identical relations developed in the SAA–M theory is extended to the SAA–M/DS.

Theorem 5 *Let $\mathcal{F}(\Sigma) = G(\Sigma)$ be an arbitrary identify in SAA–M $\langle \mathcal{O}, \mathfrak{B}; \tilde{\Omega} \rangle$ and $\mathcal{F}'(\Sigma), G'(\Sigma)$ be the corresponding ORSs in SAA–M/DS, $\langle \mathcal{O}, \mathfrak{B}; \tilde{\Omega} \rangle$; then in SAA–M/DS the relation $\mathcal{F}'(\Sigma) = G'(\Sigma)$ holds.*

Thus, the apparatus of relations developed in the SAA–M theory may be used for formal transformation, in particular, optimization by the chosen criteria of control and data structures in the process of algorithm and program design (see s.4).

The SAA–M apparatus refines the concept of ATD, popular in the present–day programming languages, many–sorted algebraic system $\langle \mathcal{K}, \tilde{\gamma}; \tilde{r} \rangle$ where $\mathcal{K}$ is a totality of basic sets, $\mathcal{K} = \{O_i \mid i \in J\}$, $\tilde{\gamma}$ and $\tilde{r}$ are signatures of operations and, respectively, predicates defined on $\mathcal{K}$. Here the sets $O_i \in \mathcal{O}$ are formalized by ORS, and operations and predicates $\tilde{\gamma}$ and $\tilde{r}$ — in terms of RS over abstract types of memory (ATM) $\langle \mathcal{N}, \{\mathcal{F}_j(\Sigma) \mid j = 1, \ldots, k\}\rangle$ where $\mathcal{N}$ is a medium (in the general case, a multivariate space of locations); $\mathcal{F}_j(\Sigma)$ are operators of the access to medium $\mathcal{N}$. In terms of ATM representable are the known memory structures: elastic tape and its special cases — stack, queue, deck, their various combinations, and also the ATM oriented towards synchronous and asynchronous multiprocessing including pipelining, roundrobin and pile processing which go back to the known industrial analogs. With introduction of dimensionality in the medium space the possibility appears to describe various ATMs for solving computer graphics, design and technological preparation [3].

By paraphrasing the Wirth thesis we way note that the computation process is equal to an algorithm plus data plus means of execution. By means of the SAA–M apparatus the algorithms and the processed data are formalized on suitable ATMs in terms of management structures. Thus, the SAA–M is a conceptually integral apparatus oriented towards formalization of principal aspects of the process of both parallel and sequential computations.

4 SAA–M/DS and Grammars of Structured Design (GSD)

Let $\mathfrak{A}^n = \Sigma \cup R$ be a terminal alphabet where Σ is a totality of base conditions, operators, objects, ATM, ATD which define the degree of precise definition of the designed class $\mathcal{G}$ of algorithms and programs, R be a totality of separations–characters of operations of SAA–M signature, brackets, delimiters, etc.; V_H^Π be a non–terminal alphabet to which logical, operator, and object metavariables belong which characterize the degree of abstraction adopted in the design process; $\Delta^\Pi \in V_H^\Pi$ be an axiom identifying the class $\mathcal{G}$ under design; $P^\Pi = \{p_i \mid i \in J\}$ be a set of marked substitutions of logical, operator, object type which detail ATD and ATM used in the design of algorithms and programs of the class $\mathcal{G}$.

By GSD is meant the formal system $G^{\Pi} = (\mathfrak{A}^{\Pi}, V_H^{\Pi}, \Delta^{\Pi}, P^{\Pi}, U^{\Pi})$ where U^{Π} is a derivation control mechanism adopted in G^{Π}. A set $L(G^{\Pi}) = \{x \mid \Delta^{\Pi} \overset{*}{\Rightarrow} x\} \subset F(\mathfrak{A})^{\Pi}$ of RS–programs derived from the axiom Δ^{Π} in G^{Π} forms a language associated with class $\mathcal{G}$ generated by the given grammar. U^{Π}, the mechanism of sequential, parallel, or combined derivation control serves in GSD to implement the context memory and data interrelations between the program modules being designed. The decision between the application of substitutions from the left to the right (the refinement), from the right to the left (the aggregation) determines the design strategy: top–down, bottom–up, or combined. The GSD apparatus underlies the method of multilevel structured program design (MSPD) [3]. Matrix GSDs with the sequential, parallel, or combined application of substitutions in generalized matrix productions received primary emphasis. The substitutions used sequentially are written in a row, and those used in parallel — in a column.

Let us show the process of interrelated design of management, memory and data structures on the example of a matrix GSD which will be called the R–program and is the kernel of matrix GSDs which generate a number of classes of multi–layer algorithms and programs. The essence of the multi–layer processing consists in dividing the array MAS of input data into mutually exclusive subarrays which are to be jointly processed with the subsequent assembly of the obtained intermediate results. The R–program scheme consists of the following generalized matrix productions

$$m_0 : \left\| \Delta^{\Pi} \to (\Pi PC) \times ASS[\xi] \right\|,$$

$$m_1 : \left\| \begin{array}{l} MAS \to MAS_1, MAS \\ \Pi PC \to A_1(\tau) \dot{\vee} \Pi PC; \tau \to MAS_1 \\ \xi \to \xi_1, \xi \end{array} \right\|,$$

$$m_1' : \left\| \begin{array}{l} MAS \to MAS_1 \\ \Pi PC \to A_1(\tau); \tau \to MAS_1 \\ \xi \to \xi_1 \end{array} \right\|,$$

$$m_2: \left\| \begin{array}{l} MAS \to MAS_{i+1}, MAS \\ A_i(MAS_i) \dot{\vee} \Pi PC \to A_i(MAS_i) \dot{\vee} A_{i+1}(\tau) \dot{\vee} \Pi PC, \tau \to MAS_{i+1} \\ \xi \to \xi_{i+1}, \xi \end{array} \right\|,$$

$$m_2': \left\| \begin{array}{l} MAS \to MAS_{i+1} \\ A_i(MAS_i) \dot{\vee} \Pi PC \to A_i(MAS_i) \dot{\vee} A_{i+1}(\tau), \tau \to MAS_{i+1} \\ \xi \to \xi_{i+1} \end{array} \right\|,$$

The design according to the considered GSD consists of two stages: 1) the division of the input MAS into the finite number of mutually exclusive subarrays of MAS_i and the formation of the corresponding number of branches $A_i(MAS_i)$ of the parallel regular scheme (PRS); 2) the formation of the necessary number of variables ξ_i in the ASS operator which are assigned with intermediate results obtained by parallel branches. Each of parallel branches $A_i(MAS_i)$ may be further interpreted by matrix productions for realization of the process of left–side, right–side, or combined data processing of corresponding subarrays MAS_i. In particular, by joining the following matrix productions to the R–program productions we obtain the GSD $MULT$ generating the class of multi–layer sort algorithms

$$m_3: \left\| A_i(MAS_i) \to \overrightarrow{SORT}(MAS_i), MAS_i \to M_i, \xi \to L_i \right\|,$$

$$m_4: \left\| A_i(MAS_i) \to \overleftarrow{SORT}(MAS_i), MAS_i \to M_i, \xi \to L_i \right\|,$$

$$m_5: \| ASS[L_1, L_2, \ldots, L_k] \to SUM[L_1, L_2, \ldots, L_k] \|,$$

where $\overrightarrow{SORT}$ is the algorithm of the left–side sort; $\overleftarrow{SORT}$ is its dual right–side algorithm of the right–side sort; M_i is a subarray to be sorted along the i–th branch; L_i is a subarray sorted already along the given branch; $SUM[L_1, L_2, \ldots, L_k]$ is the algorithm of merging the ordered subarrays, $i, k = 1, 2, \ldots$.

If the interpretation of the components entering into the right–hand sides of the productions of the R–program is different we may obtain matrix GSDs generating the classes of algorithms of the multi–layer syntactical analysis and other classes of algorithms oriented towards distributed asynchronous multiprocessing. By substituting the operation of asynchronous disjunction by composition in the productions of the R–program we obtain the GSD generating classes of algorithms of block–by–block array processing.

The possibility of using the apparatus of relations developed in the SAA–M theory makes it possible to transform the GSD productions with the purpose of modification of classes of algorithms and programs under design. In particular, the optimization, by the chosen criteria, of $\overrightarrow{SORT}$ and $\overleftarrow{SORT}$ productions entering into right–hand sides of productions of GSD $MULT$ results in the proper number of algorithms of multi–layer sort generated by the given GSD. Thus, the system of relations $\tilde{L}$ which is the basis of the formal transformation of the algorithm of shuttle sort was developed in [12], and as a result the algorithm of parallel pipelined asynchronous sort was obtained. In the process of this transformation a number of new sequential and parallel sort algorithms were obtained which exceed the source algorithms in high speed.

The process of formalized design based on the given apparatus serves as the substitution of the validity of corresponding algorithms and programs. The system $\mathfrak{L}$ upon which the transformation process is based consists of 37 relations, 22 of which represent the properties of operations of the SAA–M signature and only 15 relations characterize the chosen enterprise (the sort task). This means that the considered process of formal transformation may be extended into a more broad class of algorithms and programs.

The distinguishing feature of the GSD apparatus which is the basis of the MSPD method consists in its orientation towards substantiation (formalization of semantics, stage–by–stage verification and optimization) of algorithms in the process of designing their classes.

5 Automation of Structural Synthesis of Programs

The system of automation of structural synthesis of programs, MULTIPROCESSIST [3], is the basis of MSRD tools. Its input language is a language of structured schemes of parallel programs based on the SAA–M apparatus. Using a program design in the input language and implementations of its prime modules, MULTIPROCESSIST performs the synthesis, i.e., the program interpretation in one of the target programming languages. The synthesizer incorporates an interactive subsystem for analytical transformations and stage–by–stage verification of programs under design. The development of this system and its operating experience shattered the myth about inefficiency of realization of program design and specification languages.

As distinct from the known methods of program design (CDL–2, PDL, IBM pseudocode, etc.) the MSPD method and its tools are based on the mathematical theory of SAA–M and this provides the conceptual integrity of these tools, the simplicity and ease of their use.

References

[1] Glushkov V.M. Theory of Algorithms and Formal Transformations of Micriprograms. Kibernetika, 1965, No.5, p.1–10

[2] Dijkstra E.W. A Discipline of Programming. – Prentice–Hall, Inc. Englewood Cliffs, N.J., 1976, 280 p.

[3] Jushchenko E.L., Cejtlin G.E. Multilevel Synthesis of structured Programs. Kibernetika, 1982, No.5, p.11–21

[4] Gluschkow V.M., Zeitlin G.E., Jushchenko E.L. Algebra. Sprachen. Programmierung. Academie–Verlag. Berlin, 1980, p.340 (Glushkov V.M., Cejtlin G.E., Jushchenko E.L. Algebra. Languages. Programming. 3 izd., pererab. i dop. Kiev: Naukova Dumka, 1989, 376 p. (Russian))

[5] Cejtlin G.E. Formal Transformations in Three–Valued Logic of Structural Programming. Proc. of Structural Programming, 12th Int. Symp. on Multiple–Valued Logic, Paris, 1982, p.336

[6] Lamport L. The “Hoare Logic” of Concurrent Programs. Acta Informatica 1980, No.4, p.21–37

[7] Boehm C., Jacopini G. Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules. Comm. ACM, 9, p.336–371, 1966

[8] Cejtlin G.E. Formal Aspects of go to Structural Programming. Programmirovanie, 1984, No.1, p.3–16

[9] Knuth Donald E. Structured Programming go to Statements. Computing Surveys, Vol.6, No.4, 1974, p.292

[10] Hoare C.A.R. Data Structures. Current Trends in Programming Methodology. Vol.4, Englewood Cliffs, 1978, p.1–11

[11] Cejtlin G.E., Jushchenko E.L. Several Aspects of Theory of Parametric Models of Languages and Parallel Syntactic Analysis. Lecture Notes in Computer Science, 1977, No.47, p.231–245

[12] Cejtlin G.E. Formal Transformation of Structured Sort Algorithms. Programmirovanie, 1985, No.2, p.79–91

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

Received 30 June, 1995