In conclusion note that the principle of location from considerations of maximum entropy can be utilized to solve the n-dimensional location problem and also various applied problems which are interpretable as location of dependent ones in a chain of matrices.

LITERATURE CITED


ASYNCHRONOUS INTERPRETATION OF PARALLEL MICROPROGRAMS

O. L. Bandman

UDC 681.322.0:681.3.06

1. STATEMENT OF THE PROBLEM

In view of the rapid advances in integrated microelectronics, considerable attention is being recently focused on hardware for parallel processing of large data arrays. The mathematical model of systems of this kind — a cellular automaton [1] — was formally described and carefully studied in a number of previous publications [2, 3]. Its algorithmic generalization is represented by the parallel substitution algorithm [4], which is directly interpreted by an automaton network and therefore constitutes a practicable base for the synthesis of homogeneous microprogrammed structures [5].

The main property of the parallel substitution algorithm is its locality, and it is therefore classifiable as a so-called "myopic" algorithm [6]. Its locality is manifested in the dependence of each elementary action of a particular substitution only on some pre-specified subset of processed data. In the interpreting automaton network, this property is embodied in the fact that each automaton has a bounded neighborhood and may interact only with its neighbors.

The set of substitution is a parallel microprogram [5] which differs from an ordinary microprogram in that it determines neither the sequence of execution of microinstructions (substitutions) nor the point of their application. Instead, each microinstruction contains a local applicability condition and is executed wherever it is applicable at the particular moment of time.

Two forms of parallel microprogram interpretation by automaton networks are possible: synchronous interpretation and asynchronous interpretation (Fig. 1).

Synchronous interpretation corresponds to the rules of application of microinstructions in the parallel substitution algorithm [4] prescribing simultaneous and ubiquitous execution of all the microinstructions which are applicable at the given moment. This requires a clocking signal in the network in order to control all the automata. However, the clocking signal destroys the locality-of-interaction principle. It is therefore advisable to consider possible ways for eliminating overall synchronization and yet avoiding the difficulties which generally accompany asynchronous parallel processing.

Asynchronous interpretation of a parallel microprogram is regarded as an automaton network in which each automaton receives on its inputs every change as soon as it occurs and
Fig. 1. Schematic relationships between synchronous and asynchronous interpretations of parallel microprograms.

responds to it immediately. The time required for the automaton to pass to a new state is either not specified (i.e., it is only stipulated that this transition time is finite) or is bounded from above by a certain number of conventional automaton cycles. A cycle in an asynchronous network is defined as any time interval between two successive state changes of at least one automaton in the network. In an asynchronous network, the same initial network state (initial data array) may naturally produce a multiplicity of different state sequences, since the order in which the automaton states change is not specified. In other words, asynchronous parallel structures may produce nondeterministic computations. How to ensure determinancy is the main issue in the synthesis of parallel asynchronous structures. In Fig. 1 the transition from synchronous to equivalent (in the sense of end results) asynchronous interpretation is marked by arrow 1. This transition was constructed [7, 8] for the particular case of one-dimensional networks interpreting so-called "asynchronous grammars." Other studies [6, 9, 10] attempt to devise techniques for the transition from synchronous to asynchronous networks on the automaton realization level (arrow 2 in Fig. 1).

In this article, we construct the transition 1 for the given case, i.e., we demonstrate how to pass from a given synchronous interpretation of a parallel microprogram to an equivalent asynchronous interpretation, and investigate the cost associated with the rejection of external synchronization in parallel microprogram structures.

2. PARALLEL MICROPROGRAMS AND THEIR INTERPRETATION

Parallel microprograms operate on data arrays represented in the form of cellular set. A cellular set is a set of pairs \((a_i, m_i) \in A \times M\) (\(A\) is the alphabet, \(M\) is the name set), called cells, such that no two cells have the same name. Finite cellular sets are called words and have the form \(W = \{(a_1, m_1)(a_2, m_2)...(a_n, m_n)\}\). Sometimes it is convenient to index (arbitrarily) the elements of the set \(M\). Then each word may be characterized by two vectors, called its projection: the first projection \(Pr_1(W) = (a_1, ..., a_n)\); the second projection \(Pr_2(W) = (m_1, ..., m_n)\). In what follows, we only assume finite sets \(M\).

The set of words \(S = \{W_1, ..., W_j, ..., W_n\}\), where \(W_j = S(x_j) = \{(a_i, \Psi_i(x_j))\} (i = 1, 2, ..., n)\), is called a configuration if \(Pr_1(W_j) = (a_1, ..., a_n)\), \(Pr_2(W_j) = \{\Psi_1(x_j), ..., \Psi_n(x_j)\}\) for any \(W_j \in S\), and \(\Psi_i(x)\) are functions such that \(\Psi_i(x) \neq \Psi_k(x) (k, i \in \{1, 2, ..., n\}, x \in M)\).

The product of two configurations \(S_1 = (a_1, \Psi_1(x))\) and \(S_2 = (b_1, \Psi_2(x))\) (i = 1, 2, ..., n, j = 1, 2, ..., m) is defined as \(S = S_1 \ast S_2 = (a_1, \Psi_1(x))...(a_n, \Psi_n(x))(b_1, \Psi_1(x))...(b_m, \Psi_m(x))\).

An expression of the form \(\theta:S_1 \ast S_2 \rightarrow S_3\) is called a substitution; \(S_1 \ast S_2\) is the left part of the substitution and \(S_3\) is the right part. In the left part \(S_1\) is the basis configuration, \(S_2\) the context configuration. If \(Pr_2(S_1) = Pr_2(S_2)\), the substitution is called stationary. Stationary substitutions are microinstructions of a parallel microprogram. A microinstruction is applicable to a word \(W\) if there is at least one cell \((a, x)\) in the word \(S_1(x) \ast S_2(x) \subseteq W\). The application of the microinstruction \(\theta_j\) to the word \(W\) with respect to \(x\) is called a microoperation (it is denoted by \(\theta_j(x)\)). The result of the application of \(\theta_j(x)\) to the word \(W\) is
Fig. 2. Graphs of synchronous (a) and asynchronous (b) computations $\Psi_1$ and $\Psi_2$. $W = \{(a, 1)(b, 2)(a, 3)(c, 4)\}$.

equal to the word $\theta_1(x)\psi(W)$ in which the states of the cells from $S_{i1}(x)$ are replaced with the states of the corresponding cells from $S_{i3}(x)$,

$$\theta_1(x)\psi(W) = (W \setminus S_{i1}(x)) \cup S_{i3}(x). \quad (1)$$

Definition 1. A parallel microprogram is a finite set of stationary substitutions $\Psi = \{\theta_i : S_{i1} \times S_{i2} \rightarrow S_{i3}\}$, $(i = 1, \ldots, v)$ such that for each pair $\theta_i$ and $\theta_j$ we have $S_{i1} \times S_{i2} \neq S_{j1} \times S_{j2}$.

A parallel microprogram is applicable to the word $W$ if the set of substitutions applicable to $W$ is nonempty.

We distinguish between two modes of application of a parallel microprogram to cellular sets and between two corresponding forms of interpretation, synchronous and asynchronous.

Synchronous interpretation $\Psi_1$ of a parallel microprogram $\Psi$ corresponds to the parallel substitution algorithm.[4, 5]. In these algorithms, the rules of application of microinstructions to the processed word $W$ constitute the following iterative procedure: Let $W_k$ be the word generated in step $k$; if $\Psi$ is inapplicable to $W_k$, then $W_k$ is the result of the application of the microprogram $\Psi$ to the word $W$, i.e., $W_k = \psi(W)$; if $\Psi$ is applicable to $W_k$, then

$$W_k^{k+1} = (W_k \setminus \bigcup_{e \in \theta_i X} S_{i1}(x)) \cup \bigcup_{e \in \theta_i X} S_{i3}(x), \quad (2)$$

where $\theta_k$ is the set of microinstructions applicable to $W_k$; $X_i$ is the name set of cells to which $\theta_i$ is applicable.

Definition 2. An asynchronous interpretation $\Psi_2$ of a parallel microprogram $\Psi$ is the microprogram $\Psi$ jointly with the following iterative procedure of its application to the processed word $W$: Let $W_k^k$ be the word generated in step $k$; if $\Psi$ is inapplicable to $W_k^k$, then $W_k^k$ is the result of the application of the $\psi$ to $W$, i.e., $W_k^k = \psi(W)$; if $\Psi$ is applicable to $W_k^k$, then

$$W_k^{k+1} \in \{\theta_k(W_k^{k+1})\}. \quad (3)$$

where $\theta_k$ is any microoperation applicable to $W_k^k$. Here the word $W_k$ is called the predecessor of $W_k$, and the $W_k^{k+1}$ is the successor of $W_k$.

The set which contains the original word $W$, its successors, the successors of its successors, etc., i.e., all the words that can be generated by multiple application of the microprogram $\Psi$ to $W$, is called the set of reachable words (notation, $\hat{\Psi}$).

Definition 3. A computation $\Psi_3$ is the set of reachable words $\hat{\Psi}$ with the relation $R$ defined on it, such that $W_k \Psi_3 W_k^k$ if $W_k$ is a predecessor of $W_k^k$.

A computation may be synchronous ($\Psi_1$) or asynchronous ($\Psi_2$) according to the particular microprogram interpretation implied. A computation is visualized by a directed graph $G = <\hat{\Psi}, E>$, where $\hat{\Psi}$ is the vertex set, $E \subseteq \hat{\Psi} \times \hat{\Psi}$ is the set of arcs, and $(W_1, W_k) \in E$, if $W_1$ is a predecessor of $W_k$. 

186
A vertex without an incoming arc is the initial vertex; a vertex without an outgoing arc is the result. A computation is called finite if the graph is without cycles. Any subset of vertices constituting an oriented path through the computation graph defines a word sequence. A word sequence which starts at the initial vertex is called the initial sequence.

In a graph representing a synchronous computation, each vertex has at most one outgoing arc and therefore a finite computation has a unique result (Figs. 2a and 3a). An asynchronous computation graph has vertices with multiple inputs and outputs, and therefore, an asynchronous computation may produce a multiplicity of results (Figs. 2b and 3b). Each word sequence defines for any cell with the name a sequence of states . Thus, for instance, if , then . The sequences obtained by replacing subsequences of identical symbols of the form with a single symbol is called the history of the cell (notation, ). The history of a cell characterizes the state changes that the cell underwent in the process of computation. Two word sequences, and of the computation , where , are called equivalent if .

Definition 4. An asynchronous computation is called deterministic if the graph has a unique result.

A deterministic computation is single-valued if two word sequences and such that are equivalent.

Definition 5. An asynchronous interpretation of a parallel microprogram is deterministic if for any the computation is deterministic; if for any the computation is single-valued, then is single-valued.

Important properties of parallel computations are consistency and stability.

Definition 6. A computation is called consistent if for all the following is true: If , then ; ; ; ; .

Condition ensures for each cell univalence of the transition state even when the cell belongs to basis configurations of two applicable microinstructions. A parallel microprogram is called consistent if (5) holds for all its computations (for all ).

The consistency property is equally valid for synchronous and asynchronous interpretations; for the former (parallel substitution algorithms) it was studied in detail in [11].

Definition 7. An asynchronous computation is called stable if for any the following is true: If is applicable to , then for any word sequence starting with there is such that

The stability property implies that if the cell is due to be replaced with as a result of the applicability of to , then this replacement will occur even if the microoperation is not applied in the given realization of the computation . An inconsistent computation is obviously unstable. The stability property is more general and it includes consistency as a particular case.

A parallel microprogram is called stable if all its asynchronous computations are stable. An inconsistent microprogram is not stable.

Example 1. Fig. 2a, b shows graphs of synchronous and asynchronous computations, respectively:
It is easily seen that the computation $\psi_1(W)$ is nondeterministic: It has three possible results, $\psi_1(W) = \{(b, 1), (b, 2), (b, 3), (c, 4)\}$, $\psi_2(W) = \{(b, 1), (b, 2), (b, 3), (a, 4)\}$, and $\psi_3(W) = \{(b, 1), (b, 2), (b, 3), (b, 4)\}$. The computation is consistent but unstable, since for $W^2 = \{(a, 1), (a, 2), (a, 3), (c, 4)\}$ the condition of Definition 7 is not satisfied (when the micro-operation $\theta_{32}$ is performed, the applicability of $\theta_{42}$ to $W^2$ breaks down, so that the cell $(c, 4)$ will never change to $(b, 4)$).

THEOREM 1. An asynchronous interpretation of a parallel microprogram is single-valued if and only if the microprogram is stable.

Proof. Necessity. Let the computation $\psi_W$ be single-valued but unstable, i.e., the condition of Definition 7 is not satisfied for some cell $(a, x) \in S_j(y) \cap W^k$ ($W^k \in W$, $(b, x) \in S_j(y)$). This means that for some word sequences starting with $W^k$ there is no $n$ ($n = 1, 2, \ldots$) such that $(b, x) \in W^{k+n}$, i.e., there are two sequences $\sigma_1$ and $\sigma_2$ in which $\sigma_1(x) = a\ldots b\ldots$, $\sigma_2(x) = a\ldots c\ldots$ ($c \neq b$). Therefore, $h_1(x) \neq h_2(x)$, which contradicts the definition of single-valued asynchronous computations (Definition 5).

Sufficiency. If $\psi_W$ is stable, then for each cell $(a, x) \in S_j \cap W^k$ ($W^k \in W$, $(b, x) \in S_j(y)$) the state sequence for all $\sigma$ starting with $W^k$ starts with $a\ldots b\ldots$, which ensures equality of the corresponding histories, i.e., univalence of $\psi_W$. Since these considerations hold for all $W = A \times M$, the theorem is proved. Q.E.D.

3. SIMULATION OF SYNCHRONOUS COMPUTATIONS BY ASYNCHRONOUS COMPUTATIONS

The problem of transforming a synchronous interpretation $\Phi_W$ of a parallel microprogram $\Phi$ to an equivalent asynchronous interpretation $\Psi_W$ will be solved by constructing a microprogram $\Psi_W$ such that each asynchronous interpretation $\Psi_W$ simulates the corresponding synchronous computation $\Phi_W$. Univalence and therefore equality of the results will be ensured by using an appropriate definition of simulation.

Definition 8. The sequence $\beta = b_1\ldots b_p$ ($b_j \in \tilde{\Lambda}$) is called an $\xi$-stretching of the sequence $\alpha = a_1\ldots a_q$ ($a_j \in A$) (notation, $\beta = \xi(\alpha)$) if

1) the alphabet $\tilde{\Lambda} = \tilde{\Lambda}_0 \cup \tilde{\Lambda}_1 \cup \ldots \cup \tilde{\Lambda}_{k-1}$, and

$$\tilde{\Lambda}_0 \cap \tilde{\Lambda}_1 \cap \ldots \cap \tilde{\Lambda}_{k-1} = \emptyset \quad (6)$$

and there exists a single-valued mapping $n^\sigma: A \to A_0$

2) in the history $h_\beta = b_1\ldots b_r$, if $b_{i+j} \in \tilde{\Lambda}_k$, then $b_{i+j+n} \in \tilde{\Lambda}_k n_{mod\xi}$;

3) in the sequence $\beta' = b'_1\ldots b'_r$, obtained by deleting from $h_\beta$ all $b_{i+j} \notin A_0$, $j = 1, 2, \ldots, r'$, we have $b'_{ij} = n^\sigma(a_j)$, which for simplicity will be denoted as $\beta' = n^\sigma(\alpha)$.
For example, \( A' = \{a, b\} \), \( A_0 = \{a^0, b^0\} \), and \( a^0 = \eta^0(a) \), \( b^0 = \eta^0(b) \), \( A_1 = \{a^1, b^1\} \), \( A_2 = \{a^2, b^2\} \). The sequence \( \beta = a^0a^1a^0a^1a^0a^1a^0a^1a^0a^1a^0a^1 \) is a 3-stretching of the sequence \( \alpha = abbb \), since the history \( h_\beta = a^0a^1a^0a^1a^0a^1a^0a^1a^0a^1a^0a^1 \). Deleting from \( h_\beta \) the symbols which belong to \( A_1 \) and \( A_2 \), we obtain \( \beta' = a^0b^0a^0 = \eta^0(\alpha) \).

From Definition 8 it follows that any two \( \xi \)-stretchings of the same sequence \( \alpha \) have equal histories and are therefore equivalent,

\[
\xi_\ell(\alpha) = \xi_{\ell}(\alpha).
\]

Definition 9. The computation \( \psi_W' \) (\( W \in A_0 \times M \)) simulates the computation \( \phi_W \) (\( W \in A \times M \)) if for every initial sequence \( \sigma' \) of the computation \( \psi_W' \) there is an initial sequence \( \sigma \) of the computation \( \phi_W \) such that \( \sigma'(x) = \xi(\sigma(x)) \) for all \( x \in M \).

If \( \sigma'(x) = \xi(\sigma(x)) \), then by Definition 8 \( \sigma(x) \) partitions \( \sigma'(x) \) into subsequences which start with the symbols \( a^0 = \eta^0(a) \)

\[
a^0 \ldots a^\ell \ldots a^\ell \ldots a^\ell \ldots a^\ell,
\]

\((a^\ell \in A_\ell, \ldots, a^\ell \in A_{\ell-1})\). Indexing the subsequences (8) in \( \sigma'(x) \), we will call their index the cell age. We say that within each subsequence the cell remains of age \( T \) and it advances to age \( T + 1 \) when its state changes from \( \alpha \in A_{\ell-1} \) to \( \alpha \in A_\ell \).

Definition 10. An asynchronous interpretation \( \Psi \) of a parallel microprogram \( \Psi \) simulates the synchronous interpretation \( \phi \) if for each computation \( \phi_W \) (\( W \in A \times M \)) in \( \Psi \) there is a simulating computation \( \psi_W \) (\( W \in A_0 \times M \))

THEOREM 2. If the microprogram \( \phi \) is consistent, then any asynchronous interpretation \( \Psi \) simulating the synchronous interpretation \( \phi \) is single-valued.

Proof. Since \( \Psi \) is synchronous, then in each computation \( \psi_W \) every \( W' \in W \) has at most one successor. This implies that either for each \( W' \in W \) there is only one initial sequence \( \sigma_W \) terminating in \( W' \) or for each integer \( q \) and every \( W' \) there is only one sequence \( \sigma_{pq} \) in which a \( q \)-fold repeating subsequence can be identified (the computation cycles).

If the simulated \( \psi_W \) corresponds to the first case, the uniqueness of \( \sigma_W \) implies that all the sequences \( \sigma_{pq}(x) \) (\( x \in M \)) of the computation \( \psi_W \) terminating with the word \( W'P \) such that \( Pr_1(W'P) = \eta^0(Pr_1(WP)) \) constitute \( \xi \)-stretching of the same \( \sigma_W \). Thus, by (7), they are equivalent and therefore the computation \( \psi_W \) is single-valued. If the simulated \( \psi_W \) corresponds to the second case, then this argument applies to initial sequences \( \sigma_{pq}(x) \) terminating in \( W'P \) which contain \( q \) identical subsequences. Since the choice of \( W \in A \times M \) is not restricted, the argument applies to any pair \( \psi_W \) and \( \psi_W' \), which proves the theorem. Q.E.D.

By Theorem 2, our problem reduces to the construction of a microprogram \( \Psi \) whose asynchronous interpretation simulates the synchronous interpretation of the given microprogram \( \phi \). We will try to solve this problem in the following way: We will devise a constructive procedure for passing from \( \phi \) to \( \Psi \) and then show that the resulting \( \Psi \) simulates the given \( \phi \).

4. CONSTRUCTION OF A SIMULATING MICROPROGRAM

The proposed technique for the construction of a simulating microprogram is based on three propositions, which follow from the definition of simulation and from the asynchronous procedure for the application of microinstructions.

Proposition 1. The applicability of a microinstruction to \( W' \in A \times M \) should be established according to the states of neighborhood cells which have the same age.

This requirement can be satisfied in the following way. Suppose that at any moment of time the lengths of the histories \( L(h(x)) \) and \( L(h(y)) \) of any two neighboring cells \( x, y \in M \) in the computation \( \psi_W \) differ at most by \( \tau \) symbols, i.e., \( |L(h(x)) - L(h(y))| \leq \tau \). Then clearly for two distinct neighbors, \( y_1 \) and \( y_2 \), of the same cell \( x \), the difference of the history lengths is \( |L(h(y_1)) - L(h(y_2))| \leq 2\tau \). Thus, in order to ensure that the entire neighborhood of the cell \( x \) at a certain moment of time has the same age, we must ensure that no cell changes its age during \( (2\tau + 1) \) successive state transitions. In other words, the length of the histories of the subsequences (8) should be at least \( 2\tau + 1 \) and the alphabet \( A \) should consist of \( \xi \geq 2\tau + 1 \) subsets.
The natural preference for short sequences suggests that we adopt the minimal \( T = 1 \), so that \( \xi = 3, \lambda = \lambda_0 \cup \lambda_1 \cup \lambda_2 \).

Proposition 2. The transition of any cell \( x \in M \) to the next \((T + 1)\)-th age should occur only after the following two facts have been established: a) The cell \( x \) received information about the states of all its neighbors of age \( T \); b) the state of the cell \( x \) of age \( T \) is known to all its neighbors.

Consider a computation \( W \) in which the application of \( \theta_{ix} \) replaces \((a, x)\) with \((c, x)\). In the computation \( W \) this replacement corresponds to a sequence of transitions \((a^0, x) \rightarrow (a^1, x); (a^1, x) \rightarrow (a^2, x); (a^2, x) \rightarrow (c^0, x)\) realized by applying three microoperations, \( \theta_{ix}^0, \theta_{ix}^1, \theta_{ix}^2 \). We restrict the applicability of the microinstructions \( \theta_{ix}^0, \theta_{ix}^1 \) and \( \theta_{ix}^2 \) in the following way. We allow the application of \( \theta_{ix}^0 \) only if the states in the neighborhood of the cell \( x \) belong to \( \lambda_0 \cup \lambda_1 \). In this case, the transition to the state \( a^1 \in \lambda_1 \) will signify that the cell \((a^1, x)\) knows what states its neighbors had when they were of the same age \( T \) as the cell itself. The application of \( \theta_{ix}^2 \) is allowed when all the cells in the neighborhood of \((a^1, x)\) have states from \( \lambda_0 \cup \lambda_2 \), and then a transition to \( a^2 \in \lambda_2 \) will signify that all the neighbors know the state of the cell \( x \) of age \( T \).

Now, if all the neighborhood cells are in states from \( \lambda_2 \) or have already passed to age \( T + 1 \), the cell \((a^2, x)\) is allowed to pass to the next-age state \( c^0 \). Since this transition occurs when the neighborhood cells have different ages, and therefore the input states of the cell \((a^2, x)\) do not correspond to the vector \( \Pr_1(S_{i1}(x) \Rightarrow S_{i2}(x)) \) of any microinstruction \( \theta_{i} \in \Psi \), the symbols \( a^2 \in \lambda_2 \) must be endowed with the attributes of both states, \( a \) and \( c \) (\( a, c \in \lambda \)). In other words, \( \lambda_2 \) should be a single-valued mapping \( \eta^2: \lambda \times \lambda \rightarrow \lambda_2 \).

Suppose that in the configurations \( S_{i1} \) and \( S_{i2} \) of all the microinstructions, \( \Psi_{i1}(x) = x \). This assumption does not detract from the generality of our analysis, since no restriction is imposed on the form of the functions \( \Psi_{i,j} \). The neighbors of the cell with the name \( x \) are cells with the names \( y \in M \) for which one of the following four equalities holds: \( \Psi_{i,j}(x) = y, \Psi_{i,j}(x) = y, \Psi_{k,l}(x) = y, \Psi_{k,l}(x) = y \). The set \( N(x) \) of the neighbors of the cell \( x \) constitutes its neighborhood. A microinstruction is called normalized if its context configuration \( S_{i2}(x) \) is augmented by configurations of the form \{\( \lambda, \Psi_{i,k}(x) \)\} in such a way that for any \( x \in M \) the second projection \( \Pr_2(S_{i1}(x) \Rightarrow S_{i2}(x)) = (y_1, ..., y_q) \) includes all the cells of the neighborhood \( N(x) = \{y_1, ..., y_q\} \).

Thus Proposition 2 indicates that any microinstruction \( \theta_{i} \in \Phi \) of the form

\[
\theta_i: \{(a_{i1}, x) (a_{i2}, x) ... (a_{in}, x) \} \ast (b_{i1}, x) \Rightarrow (c_{i1}, x) \}
\]

is normalized during the construction of \( \Psi \) by augmenting the context configuration \( S_{i2}(x) \) to \( S'_{i2}(x) = \{(b'_{11}, x), (b'_{12}, x) ... (b'_{1q}, x) \} \) is then replaced with three microinstructions:

\[
\theta^0_1: \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \} \ast \left( \begin{array}{l}
\{(b'_{i1}^0 \vee b'_{i2}^0, x), (b'_{i3}^0 \vee b'_{i4}^0, x) \} \Rightarrow \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\{(b'_{i1}^0 \vee b'_{i2}^0, x), (b'_{i3}^0 \vee b'_{i4}^0, x) \} \Rightarrow \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\end{array} \right)
\]

\[
\theta^0_2: \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \} \ast \left( \begin{array}{l}
\{(b'_{i1}^0 \vee b'_{i2}^0, x), (b'_{i3}^0 \vee b'_{i4}^0, x) \} \Rightarrow \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\{(b'_{i1}^0 \vee b'_{i2}^0, x), (b'_{i3}^0 \vee b'_{i4}^0, x) \} \Rightarrow \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\end{array} \right)
\]

\[
\theta^0_3: \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \} \ast \left( \begin{array}{l}
\{(b'_{i1}^0 \vee b'_{i2}^0, x), (b'_{i3}^0 \vee b'_{i4}^0, x) \} \Rightarrow \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\{(b'_{i1}^0 \vee b'_{i2}^0, x), (b'_{i3}^0 \vee b'_{i4}^0, x) \} \Rightarrow \{(a'_{i1}^0, x) (a'_{i2}^0, x) ... (a'_{in}^0, x) \};
\end{array} \right)
\]

where \( a'_{ij}^{P} = \eta^{P}(a'_{ij}); b'_{ik}^{P} = \eta^{P}(b'_{ik}); c'_{ij}^{P} = \eta^{P}(c'_{ij}); \eta^{P} = \eta^{P}(\lambda) \).

Proposition 3. In the simulating computation \( W \), each cell goes to the next \((T + 1)\)-th age irrespective of whether it changes its state in the transition from \( W^T \) to \( W^{T+1} \) in the computation or not.

Therefore, the microprogram \( \Psi \) should include microinstructions \( \theta_{i}^0, \theta_{i}^1, \text{ and } \theta_{i}^2 \) which ensure for every cell \((a, x)\) a transition from age \( T \) to age \( T + 1 \) \((a^0 \rightarrow a^1 \rightarrow a^2 \rightarrow a^3)\) even if \((a, x) \notin S_{iX}(1 = 1, 2, ..., v)\). In order to show how to complete \( \Psi \) with these microinstructions, we introduce the following concepts.

Suppose that in a normalized microinstruction the vector \( \Pr_1(S_{i1}(x) \Rightarrow S_{i2}(x)) = \eta_1 \) contains \( k \) symbols \( \lambda \). We say that \( \eta_1 \) covers \( |A|^k \) collections of the form \( a_1 ... a_k (a_l \in \lambda) \) which
differ from $v_i$ in those components which are equal to $\lambda$ in $v_i$. The set $V = \{v_1, ..., v_l\}$ is termed complete if for each of the $|A|q$ collections there is a covering vector $v_i \in V$.

A microinstruction is called an identity microinstruction if it has the form $S_1 \ast S_2 \ast S_3$.

**Definition II.** A completion set of microinstructions $\Theta'$ for the microprogram $\Phi$ is a set of identity microinstructions of the form

$$\Theta': \{(a_i \ast x) \ast \{(b_{ij}, \psi_j(x))\}, ..., (b_{ij}, \psi_j(x))\} \rightarrow \{(a_i \ast x)\},$$

where $\{x, \psi_1(x), ..., \psi_q(x)\} = N(x)$, and the set $V = \{Pr_1(S_i1(x) \ast S_i2(x)) : \theta_i \in \Phi \cup \Theta'\}$ is complete.

The definition of $\Theta'$ for $\Phi$ is naturally based on the representation of the set $V_i = \{Pr_1(S_i1(x) \ast S_i2(x)) : \theta_i \in \Phi\}$ in d.n.f. of the logical function $F$, which is equal to 1 on all the collections covering $V_i$ and to 0 on all other collections. Any d.n.f. of the function $F'$ such that $F \lor F' = 1$, $F \land F' = 0$, represents the set $V_i = \{Pr_i(S_i1(x) \ast S_i2(x)) : \theta_i \in \Phi\}$.

With each term of the d.n.f. of $F'$ we associate a normalized identity microinstruction $\theta_i'$. The left part of $\theta_i'$ is obtained by overlapping the projections $Pr_1(S_i1(x) \ast S_i2(x)) \in V'$, and $Pr_2(S_i1(x) \ast S_i2(x))$ and isolating from the resulting configuration a cell with the name $x$, which constitutes its basis and right part. Then from the context part we delete all the cells with the states $\lambda$. As a result, $\theta_i' \in \Theta'$.

It is easily seen that the microprogram $\Phi \cup \Theta'$ is consistent if $\Phi$ is consistent, since it is completed with identity microinstructions which do not alter the applicability conditions for any $\theta_i \in \Phi$.

**THEOREM 3.** Let $\Phi$ be a consistent microprogram and $\Theta'$ its completion. Then the microprogram $\Psi$, which is obtained by replacing each $\theta_i \in \Phi \cup \Theta'$ with the three microinstructions $\theta_i \ast \theta_i \ast \theta_i$ from (9) is single-valued and its asynchronous interpretation $\tilde{\Psi}$ simulates the synchronous interpretation $\Psi$.

**Proof.** By Theorem 1, the microprogram $\Psi$ is single-valued if it is consistent and stable. In order to prove consistency, we will show that for each pair $\theta_k, \theta_j \in \Psi$ such that $S_k(x) \cap S_j(x) \neq \emptyset$ condition (5) is satisfied. Since the cell states of the basis parts of $\theta_k, \theta_j$, and $\theta^0$ belong to nonintersecting alphabets (condition (6)), the basis parts will have a non-empty intersection only for the pairs $\theta P_j, \theta P_k$ from one subset $\theta P (p = 0, 1, 2)$, whose consistency follows directly from the consistency of $\Phi \cup \Theta'$ and the univalence of the transformations $(\Phi \cup \Theta') + \theta P, \theta P_1, \theta P_2$. We will now show that the stability conditions (Definition 7) are satisfied for any computation $\Psi W'$ ($W' \in \Lambda_0 \ast M$). Let $\theta_j \in \theta P$ be applicable to $W$ and $(a, x) \in S_j(y), (c, x) \in S_j3(y), (x, y) \in M$. Consider all the possible consequences of the application of an arbitrary $\theta_kz \in \Psi$ to a successor of the cell $(a, x)$.

1. The cell $(a, x) \notin S_k1(z) \cap S_j1(y) \ast S_j2(y)$. The application of $\theta_kz$ does not affect the applicability of $\theta_j y$.

2. The cell $(a, x) \in S_k1(z) \cap S_j1(y)$. Since $\Psi$ is inconsistent, $(c, x) \in S_k3(z) \cap S_j3(y)$ and therefore the next state of the cell $(a, x)$ is independent of which particular microoperation $\theta_j y$ or $\theta_kz$ is applied first.

3. The cell $(a, x) \in S_k1(z) \ast S_j2(y)$. By (9), if $a \in \Lambda_p$, then after the application of $\theta_kz$ the state of the cell with the name $x$ will be $b \in \Delta P \ast \theta P$, which does not break the applicability of $\theta P_j y$, since the context parts of $\theta P1 \in \Psi$ are not affected by replacement of $\eta P(b)$ with $\eta P \ast \eta (b)$.

Thus, in none of the possible cases is the cell history affected by the order in which the applicable microoperations are performed. Since this is true for any $\tilde{\Psi} W'$, $\tilde{\Psi}$ is stable. From consistency and stability we conclude by Theorem 1 that $\tilde{\Psi}$ is single-valued, which proves the first proposition of the theorem.

To prove the second proposition ($\tilde{\Psi}$ simulates $\tilde{\Phi}$), consider the computations $\tilde{\Psi} W$ and $\tilde{\Psi} W'$ such that $W' = \eta^0(W)$, where $\eta^0$ indicates that if $(a, x) \in W$, then $(a, x) \in W'$.

Let $\theta^0 j y : S^0 j 1(y) \ast S^0 j 2(y) \rightarrow S^0 j 3(y)$ be applicable to $W'$. Then by (9) there is $\theta j \in \Psi$, which is applicable to $W$. Since $\Psi$ is deterministic and using the relationships between the configuration of the microoperations $\theta^0 j y, \theta^1 j y$, and $\theta^2 j y$ we conclude that if $(a, x) \in S_j1(y)$,
\((c, x) \in S_j(y)\), then the applicability of \(g^o_jy\) leads to the following state change sequence in \(c'(x)\): \(a^0 + a^1 + a^2 + c^0\). Since the set of the first projections of the left parts of \(g^o\) is complete, each cell from \(W'\) will go through this state change cycle, and so \(W'\) will include \(W'k = \eta^o(Wk)\), and since \(\theta_j\) is applicable to \(W\), we have \((c, x) \in Wk\). Applying similar consideration to any \(W'k = \eta^o(Wk) (Wk \in \hat{W})\), we conclude that each fragment of the form \(...a^0a^1a^2c^0...\) in the history \(h'(x)\) corresponds to a fragment of the form \(...ac...\) in the sequence \(\sigma(x)\), which confirms that the conditions of Definition 9 are satisfied. Since the choice of \(W\) and \(W'\) was quite arbitrary, the proof holds for any pair \(\hat{W}W'\) and \(\hat{W'}W\), and so \(\psi \approx \psi\).

Example 2  The parallel microprogram \(\phi = \{\theta_1, \theta_2\}, A = \{a, b, c\}, M = \{1, 2, 3, 4\}, \)

\[
\Phi = \begin{cases} 
\theta_1: \{(c, x)\} \rightarrow \{(b, x)\}; \\
\theta_2: \{(a, x-1) (a, x) (a, x+1)\} \\
\rightarrow \{(b, x-1) (b, x) (b, x+1)\}.
\end{cases}
\]

is consistent but unstable. The synchronous and the asynchronous computations \(\bar{\psi}_W\) and \(\tilde{\psi}_W\) for \(W = \{(a, 1)(a, 2)(a, 3)(c, 4)\}\) are shown in Fig. 3a, b. Let us construct the microprogram \(\psi\) in the following way.

1. The alphabet

\[
\tilde{\mathcal{A}} = \tilde{A}_0 \cup \tilde{A}_1 \cup \tilde{A}_2; \tilde{A}_2 = \{a^0, b^0, c^0\}, \tilde{A}_1 = \{a^1, b^1, c^0\},
\]

\[
\tilde{A}_0 = \{a^0, a^1, b^0, b^1, c^0, c^1\}.
\]

2. The completion microprogram \(\Phi'\). With the set \(V = \{ac, ac\}\) we associate the function \(F(x-1, x, x+1): F(a, c, \lambda) = 1; F(a, a, a) = 1; F(\lambda, b, \lambda) = 0; F(\lambda, c, \lambda) = 0; F(\lambda, \alpha, \alpha) = 0; F(a, a, \alpha) = 0; (a \lor b \lor c); \)

\[
\Phi' = \begin{cases} 
\theta_2: \{(b, x)\} \rightarrow \{(c, x)\}; \\
\theta_1: \{(c, x)\} \rightarrow \{(a, x-1)\} \rightarrow \{(c, x)\}; \\
\theta_0: \{(a, x)\} \rightarrow \{(c, x)\} \rightarrow \{(a, x-1)\} \rightarrow \{(a, x)\}.
\end{cases}
\]

3. The simulating microprogram \(\psi\). For each \(\theta_i \in \Phi \cup \Phi'\), write out the three microinstructions according to (9). For \(\theta_1\), say,

\[
\theta_1: \{(c^2, x)\} \rightarrow \{(a^2 \lor a^1, x-1) (a^1 \lor a^0, x+1)\} \rightarrow \{(c^1, x)\};
\]

\[
\theta_2: \{(c^1, x)\} \rightarrow \{(a^1 \lor a^0, x-1) (a^0 \lor a^2, x+1)\} \rightarrow \{(c^0, x)\};
\]

\[
\theta_3: \{(c^0, x)\} \rightarrow \{(a^2 \lor a^1, x-1) (a^1 \lor a^2, x+1)\} \rightarrow \{(b^0, x)\}.
\]

It is clear from the foregoing that the complexity of the microprogram \(\psi\) (in terms of the number of microinstructions) is at least three times the complexity of \(\Phi\). However, this ratio is not necessarily preserved for the hardware realization. From the structure of the microprogram \(\psi\) we see that it can be represented as a parallel composition of two microprograms: 1) the microprogram \(\Phi \cup \Phi'\); 2) a microprogram of the \text{mod}_3 counter. Thus, the automaton network interpreting \(\psi\) may be stratified into two networks: the main network and the synchronizing network. The cells of the main network contain \(\Phi \cup \Phi'\), and the cells of the synchronizing network contain the \text{mod}_3 counters.

If the microprogram is realized as an automaton network, the complexity of the synchronizing network will be substantially less than the complexity of the main network. If, however, a parallel machine realization is used [5], the complexity of the "data memory" will be approximately doubled.

LITERATURE CITED


PRACTICAL METHODS OF PROGRAM VERIFICATION

V. A. Nepomnyashchii

Current methods of program correctness testing (program verification) based on axiomatic semantics of programming languages are reviewed. Verification methods based on operational, denotative semantics or on induction on data structure are, as a rule, not discussed.

The following stages can be indicated in the development of the discussed method.

1. 1947-1966. J. Neumann and A. Turing discussed the usefulness of program verification and certain ideas concerning the proof of their correctness. P. Naura proposed a (non-formalized) method of program verification. V. M. Glushkov introduced systems of algorithmic algebras oriented on proving the equivalence and other properties of programs.


3. 1969-1974. T. Hoare proposed a modification of the Floyd method which makes it possible to describe the axiomatic semantics of programming languages and considerably to simplify the determination of correctness conditions. The first automatic program verification systems were developed.


Problems associated with program verification were discussed in books [1-5] and reviews [6-11]. The present review covers some of their ground in discussing practical verification methods developed principally at the fourth stage.

1. THEORETICAL PRINCIPLES

A program A with specifications (input condition P and output condition Q) is called partially correct if, when P is true before A starts operating and after A completes operation, Q is found to be true for output values of variables. A partially correct program A is called (totally) correct if it terminates operation with P being true for input variable values.


0011-4235/84/0193(80)50 © 1984 Plenum Publishing Corporation 193