Bull. Nov. Comp. Center, Comp. Science, 2 (1993), 25–44

Synchronous-asynchronous transformation of cellular algorithms*

O.L. Bandman

Asynchronous versus synchronous cellular computations are investigated in terms of Parallel Substitution Algorithm (a formal model of fine-grained parallel data processing). A general method for constructing asynchronous cellular computation equivalent to a given cellular algorithm execution is presented. It is shown that more than a twofold increase in cell-complexity should be paid for the abandonment of synchronizing clock in cellular VLSI systems.

1. Introduction

The advancement in VLSI technology as well as the results in optical computing studies enhance the investigations of fine-grained parallel algorithms and their implementation. One of the main problems in this field is to compare synchronous versus asynchronous mode of cellular algorithms.

At the current state of computer technology development a synchronous mode of operation in fine-grained parallel processors is preferred to be used. There are three reasons for that.

1. Any valid synchronous computation is an algorithm, this is in accordance to our way of thinking.
2. Synchronous computation validity is much more easy to provide, than that of asynchronous one.
3. Hardware implementation of synchronous systems is more feasible, and their operation is more reliable. This is confirmed by the long-term experience of VLSI systems exploitation.

Nevertheless, the theory of asynchronous computations is being developed rapidly, and the methods of implementing asynchronous systems are being improved. The most important point in favour of the asynchronous

*This work was supported by Russian Fund of Fundamental Research

mode is the tendency for the decrease of switching time, the latter becoming close to that of clock pulse propagation. Hence, the clock frequency directly depends upon the signal propagation speed, which in its turn is limited by the speed of light. Thus, the common clock becomes the restraining factor for system performance enhancement, which provokes the desire to get rid of it and stimulates the investigation of asynchronous mode of cellular computations.

The main goal of the investigation is to get answer to three following questions:

1. What are the features a cellular algorithm is to possess in order to meet validity conditions.
2. How to transform a given cellular algorithm into an equivalent one, whose asynchronous computations meet the validity conditions.
3. What price in hardware and in executing time is to be paid for the abandonment of synchronizing clock.

The exhausted answer to the first question is given in [1], [2], [3]. The second and the third questions are the subject of this paper. The answers are obtained in terms of an original formal model of fine-grained computations, called *Parallel Substitution Algorithm (PSA)* [4], [5]. The results do not pretend to be perfectly complete. We only succeeded in constructing a formal transition from synchronously executable algorithm to an equivalent one whose features constitute a sufficient but not necessary condition of asynchronous computation validity [3]. It means that we are able to construct only a part of all existing computations which are valid in the asynchronous mode of execution and equivalent to the given algorithm.

The above problems, which are further investigated in terms of PSA theory, have been explored in relation to other computation models, which may be considered as subclasses of PSAs. The most fruitful from the methodological point of view approach is presented in [6], where the asynchronous computations on one-dimentional cellular array referred to as "asynchronous grammars" are studied. The relation between the computation time and the degree of asynchronism is also determined. The latter is measured with the number of times (denoted by D), during which a substitution remains applicable before being executed. The main result of this investigation is as follows: any set of strings, recognized by the synchronous grammar not more than in k times, are recognized by the asynchronous grammar not more than in $3(D+1)k$ times. Very important in [6] is also the proposed strategy for transition from the synchronous mode to the asynchronous one for $2D$ and $3D$ dimensional computation spaces.

In [7] a more general model with no constraints on the computation space dimension is presented. The possibility of simulating a synchronous computation by an asynchronous one has been studied. The main result is the following: any synchronous computation in d-dimensional cellular space may be simulated by an asynchronous one in $(d+1)$-dimensional cellular space. The strict proof of this proposition is anticipated by a method of simulating space construction, which illustrates the idea and is interesting from the methodological point of view.

It is worthwhile to pay attention also to the method of selfsynchronization of the arbitrarily connected identical automata, called "intelligent graphs" [8]. This stage of study showed that the minimal price to be paid for the asynchronism is the threefold amount both of states and of transition functions of each automaton.

In what follows a method for transforming synchronous computations into the equivalent asynchronous ones is presented. The method is in the framework of PSA theory, though some ideas and approaches from the above investigations are also used.

2. Formal statement of the problem

Let A be a finite alphabet and M be a set of names. A pair (am) belonging to $A \times M$ is called *a cell*, where a is *a state* of the cell and m is *a name* of the cell. In our investigation M is *the naming set*, being a set of integer vectors, determining the coordinates of the points in computation space. The $1D$-space is denoted as $M^1 = \{1, 2, \ldots\}$, the $2D$-space is $M^2 = \{\langle i, j\rangle : i = 0, 1, \ldots;\ j = 0, 1, \ldots\}$.

Definition 1. A finite set of cells W none two pairs of which have the same name, is called *a cellular array.* The set of all cellular arrays whose cells have states from A and names from M is denoted by $K(A, M)$.

Definition 2. An expression of the form

$$\Theta : W_1 * W_2 \Rightarrow W_3, \tag{1}$$

where W_1, W_2, W_3, and $W_1 \cup W_2$ are cellular arrays, is called *a substitution*, $W_1 \cup W_2$ is *the left-hand side* of a substitution, W_1 *is the base*, W_2 is *the context*, and W_3 is *the right-hand side* of a substitution.

A substitution is applicable to W if $W_1 \cup W_2 \subseteq W$, and the result of this application is

$$\Theta(W) = (W \setminus W_1) \cup W_3. \tag{2}$$

Example 1. The substitution

$$\Theta : \{(0,\langle 0,1\rangle)(0,\langle 0,2\rangle)\} * \{(1,\langle 0,0\rangle)\} \Rightarrow \{(1,\langle 0,1\rangle)(1,\langle 0,2\rangle)\},$$

where $A = \{0,1\}$ and $M = M^2$, being applied to the array

$$W = \{(1,\langle 0,0\rangle)(0,\langle 0,1\rangle)(0,\langle 0,2\rangle)(0,\langle 1,0\rangle)(0,\langle 1,1\rangle)(0,\langle 1;2\rangle)\},$$

results in

$$\Theta(W) = \{(1,\langle 0,0\rangle)(1,\langle 0,1\rangle)(1,\langle 0,2\rangle)(0,\langle 1,0\rangle)(0,\langle 1,1\rangle)(0,\langle 1,2\rangle)\}.$$

Definition 3. A partial function $\varphi(m)$ determined on the set $M' \subseteq M$ is called *a naming function.* A finite set of naming functions

$$T = \{\varphi_1(m),\ldots,\varphi_t(m)\}, \tag{3}$$

such that for no $m \in M$ $\varphi_i(m) = \varphi_j(m)$, $i,j = 1,\ldots,t$, $i \neq j$ and $\varphi_1(m) = m$, is called *a template.*

A template associates with each name $m_p \in M$ a set of closely allocated names $T(m_p)$ referred to as *a template element*, which may be considered as a pattern in the naming space.

Definition 4. An expression of the form

$$S = \{(a_1,\varphi_1(m))\ldots(a_t,\varphi_t(m)\}, \tag{4}$$

is called *a configuration.* A pair $(a_i,\varphi_i(m))$ is called *the i-th component of the configuration.* A cellular array, obtained by substituting a certain m_p for m in (4) is referred to as *a configuration element.* The set of naming functions occurring in (4) is called *the underlying template* $T(S)$ for S.

Example 2. Let $M = M^2, A = \{0,1\}$. The configuration

$$S = \{(1,\langle i,j\rangle)(0,\langle i-1,j\rangle)(0,\langle i,j-1\rangle)\}$$

has the underlying template

$$T = \{\langle i,j\rangle,\langle i-1,j\rangle,\langle i,j-1\rangle\}.$$

The cellular arrays:

$$S(\langle 2,2\rangle) = \{(1,\langle 2,2\rangle)(1,\langle 1,2\rangle)(0,\langle 2,1\rangle)\},$$
$$S(\langle 2,3\rangle) = \{(1,\langle 2,3\rangle)(1,\langle 1,3\rangle)(0,\langle 2,2\rangle)\}$$

are the elements of S. The element $S(\langle 0,0\rangle)$ is not defined, because so is the second component of S.

Definition 5. An expression of the form

$$\Theta : S_1 * S_2 \Rightarrow S_3, \tag{5}$$

where S_1, S_2, S_3 and $S_1 * S_2$ are configurations, is called *a parallel substitution* (further, if it does not lead to ambiguity, the adjective "parallel" is omitted for short), S_1 is called *a base*, S_2 – *a context*, $S_1 * S_2$ – *a left-hand side*, S_3 – *a right-hand side.*

A subclass of parallel substitutions, in which the configurations S_1 and S_3 have equal underlying templates is used here. Such substitutions are called *stationary substitutions* or, sometimes, *parallel microprograms* [4].

A parallel substitution is considered to be applicable to the cellular array $W \in K(A,M)$, if there exists a subset $M' \subseteq M$, such that for any $m_p \in M'$

$$S_1(m_p) \cup S_2(m_p) \subseteq W. \tag{6}$$

If $M' = \emptyset$, then Θ is not applicable to W. An applicable substitution may be executed. The execution of Θ in W results in changing states of the cells $(a_i, \varphi_i(m_p)) \in S_1(m_p)$ for the states c_i of the cells $(c_i, \varphi_i(m_p)) \in S_3(m_p)$, which is expressed in term of cellular arrays as

$$\Theta(W) = \Big(W \setminus \bigcup_{m_p \in M'} S_1(m_p)\Big) \bigcup \Big(\bigcup_{m_p \in M'} S_3(m_p)\Big). \tag{7}$$

An execution of a substitution $\Theta \in \Phi$ at a certain cell $m_p \in M$ is referred to as *a microoperation* $\Theta(m_p)$.

Definition 6. A finite set of substitutions $\Phi = \{\Theta_1, \ldots, \Theta_\nu\}$ is called *a Parallel Substitution System* (PSS).

Definition 7. The union of underlying templates of the left-hand side configurations of all substitutions of a PSS is called *the input neighbourhood template* and is denoted as $N'(m)$. Its subset, comprising the templates of the right-hand sides of the substitutions is called *the output neighbourhood template* and is denoted as $N''(m)$, i.e.,

$$N'(m) = \bigcup_{\Theta \in \Phi} T(S_{i1} * S_{i2}), \quad N''(m) = \bigcup_{\Theta \in \Phi} T(S_{i3}). \tag{8}$$

The sets of names obtained by substituting a certain name $m_p \in M$ in $N'(m)$ and in $N''(m)$ are called *input and output neighbourhoods* of the cell named m_p.

Definition 8. The iterative procedure is called *the synchronous mode of execution* of a PSS, if it is accomplished according to the following procedure.

Let W^i be the result of the i-th step, then:

1) if no substitution is applicable to W^i, then W^i is the result of the computation;
2) if there exists a non-empty subset $\Phi' \subseteq \Phi$ applicable to W^i, then W^i is substituted for

$$W^{i+1} = \bigcup_{\Theta_j \in \Phi'} \Theta(W^i). \tag{9}$$

An execution of a PSS Φ with W as the initial cellular array is called *a synchronous computation* and is denoted as $\bar{\Phi}(W)$.

Definition 9. A PSS is called *non-contradictory* on $K(A, M)$ if its application to any $W \in K(A, M)$ results in a cellular array, i.e., the result contains no pair of cells with the same names. If there exists a cellular array $W \in K(A, M)$ such that the result of application of a PSS is not a cellular array (it contains two cells identically named), then the PSS is *contradictory.*

Definition 10. A non-contradictory PSS Φ, which is to be executed on cellular arrays from $K(A, M)$ according to the synchronous mode, is called *a Parallel Substitution Algorithm (PSA)* and denoted as $\Pi = \langle \Phi, K(A, M) \rangle$.

Definition 11. The non-deterministic procedure is called *the asynchronous mode of execution* of a PSS, if it is accomplished according to the following procedure.

Let W^i be the result of the i-th step, then:

1) if no substitution is applicable to W^i, then W^i is the result of the computation;
2) if there exists a non-empty subset $\Phi' \subseteq \Phi$ applicable to W^i, then any but only one substitution $\Theta_j \in \Phi'$ applicable at a cell $m_p \in M$, is executed, so that

$$W^{i+1} = (W^i \setminus S_{j1}(m_p)) \cup S_{j3}(m_p).$$

An execution of a PSS Φ in the asynchronous mode with W as the initial state is called *an asynchronous computation* and is denoted as $\tilde{\Phi}(W)$.

The behavioral properties and validity conditions for both modes have been studied in [1], [2], [3]. They are expressed in terms of computation graphs, which are associated with the computations $\bar{\Phi}(W^0)$ or $\tilde{\Phi}(W^0)$ as follows. The vertices of a computation graph are denoted by the reachable cellular arrays $W^0, \ldots, W^q$, an arc from W^i to W^j existing if W^j is the one-step result of PSS application to W^i. Each path from the initial vertex to the terminal one is referred to as *a computation realization.* The computation is said to be *determinate,* if the associated computation graph has only one termination vertex. If each vertex in the computation graph has only one successor, then the computation is *deterministic.* The computation is *terminating,* if there is no cycles in its computation graph. The computation is called *valid* if it is determinate and terminating.

Definition 12. The synchronous (asynchronous) mode of execution is called *valid* if for any initial cellular array $W \in K(A, M)$ the synchronous (asynchronous) computations are valid.

In [1], [2], [3] necessary and sufficient conditions are obtained and methods for PSS validation are developed. For our aims here the following results from [1], [2], [3] are essential:

1) a PSA is always deterministic (and, hence, determinate);
2) an asynchronous mode of execution of a PSS is determinate if so are asynchronous computations initiated by a set of cellular arrays, referred to as *critical.* A critical cellular array $C_{jh}(m_p)$ is equal to the union of left-hand side configuration elements of a pair of substitutions $\Theta_j, \Theta_h \in \Phi$ of the form

$$C_{jh} = S_{j1}(m_p) \cup S_{j2}(m_p) \cup S_{h1}(m_p + d), \qquad (10)$$

if there exists $m_p \in M$ and an integer vector d, such that $C_{jh} \subset K(A, M)$.

A very useful extension of a parallel substitution is also used here. This extension is associated with a concept of variables and functions as alphabet symbols. The set of variable symbols $X = \{x_1, \ldots, x_p\}$ and functional symbols $Y = \{y_1, \ldots, y_q\}$, $y_i = f(x_1, \ldots, x_p)$ with the domain and the

range in A, are added to the alphabet A, so that the resulting alphabet $A^* = A \cup X \cup Y$. A configuration with variable or functional symbols in its components assign to each its element $S(m_p)$ a number of cellular arrays, each corresponding to a set of values from A standing for $x_i \in X$, $i = 1, \ldots, p$.

Definition 13. A substitution whose configurations contains components with variable and functional symbols is called *functional.*

A functional substitution is considered to be applicable to a cellular array W, if there exists a set of values in A and a name $m_p \in M$ such that after substituting them for the variables the substitution meets the applicability condition (6).

Definition 14. Two computations (of any execution mode) are called to be *equivalent*, if being initialled by equal cellular arrays, they terminate in equal resulting ones.

Definition 15. An asynchronous mode of execution of a PSS Φ' is called to be *equivalent* to a PSA $\Pi = \langle \Phi, K(A,M) \rangle$, if for any $W \in K(A,M)$ the asynchronous computation $\tilde{\Phi}(W)$ is equivalent to a synchronous one $\bar{\Phi}(W)$.

Example 3. A PSS $\Phi = \{\Theta_1, \Theta_2\}$, $A = \{0,1\}$, $M = \{\langle i,j \rangle : i = 1,2,3;$ $j = 1,2,3\}$ represents the logical OR'ing of three Boolean vectors three bit long. The substitutions of Φ are as follows:

$$\Theta_1 : \{(1,\langle i,j \rangle)(0,\langle i+1,j \rangle)\} \Rightarrow \{(0,\langle i,j \rangle)(1,\langle i+1,j \rangle)\};$$
$$\Theta_2 : \{(1,\langle i,j \rangle)\} * \{(1,\langle i+1,j \rangle) \Rightarrow \{(0,\langle i,j \rangle)\}.$$

The same algorithm expressed with using functional configurations contains only one following substitution:

$$\Theta : \{(x_1,\langle i,j \rangle)(x_2,\langle i+1,j \rangle)\} \Rightarrow \{(y,\langle i,j \rangle)(0,\langle i-1,j \rangle)\},$$

where $y = (x_1 \vee x_2)$. The input and output neighbourhoods are as follows:

$$N'(\langle i,j \rangle) = N''(\langle i,j \rangle) = \{\langle i,j \rangle, \langle i+1,j \rangle\}.$$

Figure 1 shows geometrical substitution patterns of a given PSA, Figure 2 shows its functional expression. In Figure 3 a synchronous and in Figure 4 an asynchronous computation, initialled by one and the same cellular array, are displayed. The patterns of the applicable microoperations are enclosed in rectangles. In the asynchronous computation graph an arc

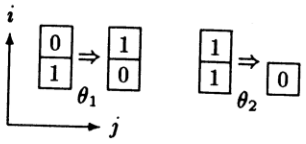

Figure 1

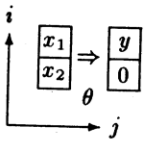

Figure 2

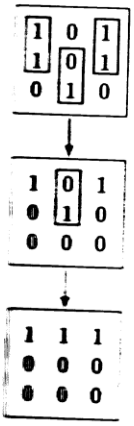

Figure 3

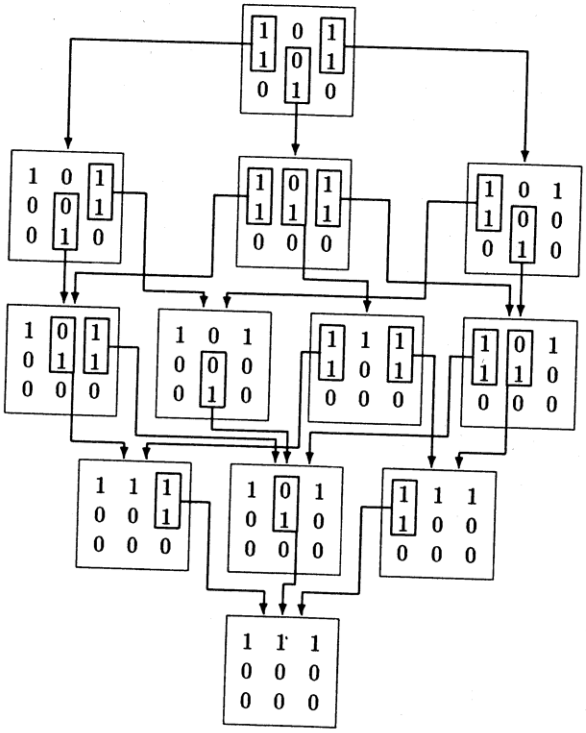

Figure 4

starting from the pattern of applicable microoperation go to the cellular array resulting of this microoperation execution.

Example 3 displays a "nice" PSS, which is valid in both synchronous and asynchronous modes of execution. It is easy to check that for any W computation $\bar{\Phi}(W)$ is equivalent to $\tilde{\Phi}(W)$. Hence, the asynchronous mode of execution of Φ is equivalent to its own synchronous mode. It is not so in general, hence the problem is to obtain a method of constructing a PSS Φ' whose asynchronous mode of execution is equivalent to a given PSA.

3. Simulating a synchronous computation by an asynchronous one

In terms of PSA theory the above problem is formulated as follows. A PSA $\Pi = \langle \Phi, K(A,M) \rangle$ is given, a PSS Φ' is to be found such that its asynchronous mode of execution is equivalent to the given Π. The problem is further solved by means of developing a method of constructing an asynchronous computation (Definition 11), which simulates the synchronous one. The concept of simulation is requested to provide validity and identity of the results.

Definition 16. Let α be a sequence of symbols from A, $\alpha = a_1, \ldots, a_j, \ldots$. Then the sequence $h(\alpha)$, obtained by eliminating from α all but one symbol from each subsequence of identical symbols is called *a history* of α.

Example 4. The history of the sequence $\alpha = 111\lambda0010001\lambda\lambda00$ is as follows: $h(\alpha) = 1\lambda0101\lambda0$.

Definition 17. Let $\alpha' = a_1, \ldots, a_q$ be a sequence of symbols from A' and A is the subset of A'. Then a sequence $\alpha = a_{i1}, \ldots, a_{ip}$, where $a_{ij} \in A$, $p < q$, is called *a projection* of α' on A, if it is obtained by eliminating from α' all symbols not belonging to A.

Definition 18. A sequence $\beta = b_1, \ldots, b_p$, $b_i \in B$, is called a ξ_r-extent of a sequence $\alpha = a_1, \ldots, a_q$, $a_j \in A$, i.e., $\beta = \xi_r(\alpha)$, if there exists a partition on B, $B = B_0 \cup \ldots \cup B_{r-1}$, such that the following holds:

1) $B_i \cap B_j = \emptyset$ for each pair $i, j \in \{0, \ldots, r-1\}$, $i \neq j$,
2) $B_0 = A$,
3) each B_i, $i = 1, \ldots, r-1$, is isomorphic to A, i.e., there exists $r-1$ one-to-one mapping $\delta_i : B_i \Rightarrow A$,

4) the history $h(\beta)$ is of the form

$$h(\beta) = a_1, \delta_1(a_1), \ldots, \delta_{r-1}(a_1), \ldots, a_q, \delta_1(a_q), \ldots, \delta_{r-1}(a_q), \quad (12)$$

5) the projection of $h(\beta)$ on α

$$Pr(h(\beta) \Rightarrow A) = \alpha. \quad (13)$$

The sequence α is referred to as *the initial sequence* for β, i.e., $\alpha = \xi_r^{-1}(\beta)$.

Example 5. Let $A = \{a_1, a_2\}$, $B = \{b_1, b_2\}$, $C = \{c_1, c_2\}$. The sequence $\beta = a_1, a_1, b_1, c_1, a_2, b_2, c_2, a_2, b_2, b_2, c_2, a_1, a_1, b_1, b_1, c_1, a_1, a_1$ is a ξ_3-extension of $\alpha = a_1, a_2, a_2, a_1, a_1$. To make sure of it the conditions of Definition 18 are to be checked as follows:

1) the alphabets B and C are isomorphic to A,
2) the history $h(\beta) = a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3, a_1, b_1, c_1, a_1$ is of the form (12),
3) $Pr(h(\beta)) \Rightarrow A) = a_1, a_2, a_2, a_1, a_1$, which is equal to α.

Lemma 1. *If $\beta = b_1, \ldots, b_p$ and $\gamma = g_1, \ldots, g_q$ are ξ_r-extensions of the one and the same sequence, then their histories are identical, i.e.,*

$$h(\beta) = h(\gamma). \quad (14)$$

Proof. The equality of the histories follows directly from the conditions 4 and 5 of Definition 17. □

Lemma 2. *If $\beta = \xi_0(\alpha)$, then*

$$h(\beta) = \alpha. \quad (15)$$

Proof. According to Definition 18, ξ_0-extension has the trivial partition of B, so that $B = B_0$, $B_0 = A$. The substitution of B for A together with the fact that the projection of a sequence of symbols from B onto B is the sequence itself gives the equality $h(\beta) = \alpha$, which proves the lemma. □

Definition 19. Let $\sigma = W^0, \ldots, W^j, \ldots, W^q$ be a computation realization (a path from the initial vertex to the termination one) in a synchronous or in an asynchronous computation graph, then the sequence

$$\sigma(m_p) = a_{i1}, \ldots, a_{ij}, \ldots, a_{iq},$$

where $(a_{ij}, m_p) \in W^j$, $j = 1, \ldots, q$, is called *a cell-realization.* If the computation is a synchronous one, then cell-realization in it is denoted as $\bar{\sigma}(m_p)$, if the computation is asynchronous – $\tilde{\sigma}(m_p)$.

Now a concept of simulating a synchronous computation $\bar{\Phi}(W^0)$ by an asynchronous one $\tilde{\Phi}'(W^0)$ is introduced. It is important to remind that a PSA Φ (a non-contradictory PSS Φ executed synchronously), generates a deterministic computation, whose computation graph is linear, representing a single realization $\bar{\sigma} = W^0, \ldots, W^j, \ldots, W^q$, which is finite, if so is the computation $\bar{\Phi}(W^0)$.

Definition 20. A computation $\bar{\Phi}(W^0)$, $W \in K(A, M)$, generated by a PSA, is said *to be simulated by an asynchronous one* $\tilde{\Phi}'(V^0)$, $V^0 \in K(A', M)$, $A \subseteq A'$, if there is a certain r, such that any cell-realization $\tilde{\sigma}_i(m_p)$ in $\tilde{\Phi}(V^0)$ is a ξ_r-extension of $\bar{\sigma}(m_p)$, i.e.,

$$\tilde{\sigma}(m_p) = \xi_r(\bar{\sigma}(m_p)) \tag{16}$$

for any $m_p \in M$.

Example 6. A computation graph $\bar{\Phi}(W^0)$, where $\Phi = \{\Theta_1, \Theta_2\}$,

$$\Theta_1 : \{(b,i)\} * \{(a,i-1)(a,i+1)\} \Rightarrow \{(c,i)\},$$

$$\Theta_2 : \{(a,i-1)(c,i)(a,i+1)\} \Rightarrow \{(c,i-1)(c,i)(c,i+1)\},$$

and $W^0 = \{(a,1)(b,2)(a,3)(c,4)\}$, is shown in Figure 5. The cellular arrays are represented as strings of symbols. The left-hand sides of applicable substitutions are underlined.

A computation graph $\tilde{\Phi}'(V^0)$, where $V^0 \in A' \times M$, $A' = A \cup A''$, $A'' = \{a', b', c'\}$, $\Phi' = \{\Theta', \ldots, \Theta\}$,

$\Theta'_1 : \{(a,i-1)(b,i)(a,i+1)\} \Rightarrow \{(a',i-1)(b',i)(a',i+1)\};$

$\Theta'_2 : \{(c,i)\} * \{(a',i-1)\} \Rightarrow \{(c',i)\};$

$\Theta'_3 : \{(a',i-1)(b',i)(a',i+1)\} * \{(c',i+2)\} \Rightarrow \{(a,i-1)(c,i)(a,i+1)\};$

$\Theta'_4 : \{(c',i)\} * \{(a,i-1)\} \Rightarrow \{(c,i)\};$

$\Theta'_5 : \{(a,i-1)(c,i)(a,i+1)\} \Rightarrow \{(a',i-1)(c',i)(a',i+1)\};$

$\Theta'_6 : \{(a',i-1)(c',i)(a',i+1)(c',i+2)\} \Rightarrow \{(c,i-1)(c,i)(c,i+1)(c,i+2)\}$

is shown in Figure 6.

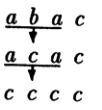

Figure 5

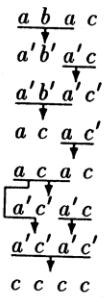

Figure 6

It is easy to make sure that $\bar{\Phi}(W^0)$ is simulated by $\tilde{\Phi}'(V^0)$, checking the condition (16) for any $i = 1,2,3,4$. Indeed, for $i = 1$ and $i = 2$ this is done as follows:

$$\bar{\sigma}(1) = a, a, c; \qquad \tilde{\sigma}_1(1) = a, a', a', a, a, a', a', c;$$

$$\tilde{\sigma}_2(1) = a, a', a', a, a', c; \qquad \xi_2(\tilde{\sigma}_1(1)) = \xi_2(\tilde{\sigma}_2(1)) = a, a, c;$$

$$\bar{\sigma}(2) = b, c, c; \qquad \tilde{\sigma}_1(2) = b, b', b', c, c, c', c', c;$$

$$\tilde{\sigma}(2) = b, b', b', c, c', c; \qquad \xi_2(\sigma_1(2)) = \xi_2(\sigma_2(2)) = b, c, c.$$

Theorem 1. *Let $\bar{\Phi}(W^0)$, $W^0 \in K(A, M)$, be a computation generated by a PSA, $\tilde{\Phi}'(V^0)$, $V^0 \in K(A', M)$, $A \subseteq A'$ is a simulating asynchronous one. Then if $\bar{\Phi}(W^0)$ is valid, then so is $\tilde{\Phi}'(V^0)$.*

Proof. As $\bar{\Phi}(W^0)$ is deterministic and finite, then its computation graph is linear, and for any cell named $m_p \in M$ there exists a single cell realization $\bar{\sigma}(m_p)$. Hence, according to Definition 19 all $\tilde{\sigma}_i(m_p)$ in $\tilde{\Phi}'(V^0)$ are ξ_r-extensions of $\bar{\sigma}(m_p)$. According to Lemma 1, their histories are identical and, hence, (according to Item 5 of Definition 18) so are the projections on the initial alphabet A. Since the projections of the histories of all $\tilde{\sigma}_i(m_p)$ are equal to one and the same sequence $\bar{\sigma}(m_p)$, they terminate in one and the same state, equal to that of the terminating cell in $\bar{\sigma}(m_p)$. As the above holds for all $m_p \in M$, the termination cellular arrays in $\bar{\Phi}(W^0)$ and in $\tilde{\Phi}'(V^0)$ are equal, which proves the theorem. □

4. Synthethis of a simulating PSS

It follows from Theorem 4 that the problem of synchronous-asynchronous transformation may be reduced to the construction of a PSS, which generates asynchronous computations simulating those, generated by the given

PSA. The method of constructing a simulating PSS is based on three propositions, which proceed both from the concept of asynchronous simulation and from the rules of asynchronous mode of execution of a PSS.

For formulating these propositions one more definition is needed.

Definition 21. Let a cell realization in an asynchronous computation $\tilde{\Phi}'(V)$, $V \in K(A', M)$, be $\tilde{\sigma}(m_p) = a_1, \ldots, a_q$, $a_i \in A'$, and $A \subseteq A'$ is *an initial alphabet.* Then an ordinal number t of a segment of $\tilde{\sigma}(m_p)$, which begins by a symbol from A and ends just before the next one from A, is called *a cell age.* Any cell state in a segment numbered $t = n$ is referred to as *a n-aged state*, the first one belonging to A being called *the initial n-aged state.*

Example 7. Let $A = \{a, b, c\}$ be an initial alphabet, $A' = \{a, b, c, a', b', c'\}$. Then in a cell realization

$$\sigma(m_p) = \underbrace{a, a', b', c', c', c'}_{t=1} \underbrace{a, b', a', c', b', c'}_{t=2} \underbrace{c, a', a'}_{t=3} \ldots .$$

The cell named m_p is 1-aged during the first six times, A being the initial 1-aged state. Then its age is changed for $t = 2$.

Proposition 1. *A substitution in a simulating asynchronous computation may be executed at a certain cell, if all the cells of its neighbourhood (Definition 7) are in one and the same age.*

In order the proposition might have been realized for a certain cell named $m_p \in M$, there should occur cellular arrays in the simulating computation, which include all the cells of the neighbourhood of m_p being in the same age. This is provided as follows.

Let us require the difference in ages for any two neighbour cells m_j and m_p not to exceed a fixed number τ of times, so that

$$| T(m_p) - T(m_j) | \leq \tau. \tag{17}$$

Hence, two neighbours m_j and m_n of one and the same cell m_p may differ in age by 2τ. So, in order the whole neighbourhood of a cell might be in one and the same age if only one time, it is necessary for each cell to preserve the age at least for $2\tau + 1$ times. Hence, the ξ_r-extension $\tilde{\sigma}(m_p)$ simulating the synchronous realization $\bar{\sigma}(m_p)$ should have the index

$$r \geq 2\tau + 1.$$

The natural wish for fast transition to the next age brings up the choice of minimal $r = 1$ and $r = 3$.

Proposition 2. *The transition of a cell from the age t to the age $t+1$ occurs after two following facts are asserted:*

1) *the cell has already read out the initial t-aged state from all cells of its input neighbourhood (Definition 7);*
2) *all cells to whose input neighbourhoods the cell belongs have already read out its initial t-aged state.*

The above conditions are fulfilled with $r = 3$ in the following way. The alphabet A, used in Φ, is extended to another one, more than three times larger, so that $\tilde{A} = A^0 \cup A' \cup A''$, where A^0 in its turn is the extended A, so that $A^0 = A \cup \{\lambda, -\}$, λ indicating *the idleness* of the cell, and "$-$" being a "don't care" symbol.

Each microoperation $\Theta_i(m_p)$, substituting (a, m_p) for (c, m_p) is simulated by the following sequence of cell-state changings:

$$a \Rightarrow (ac)', \quad (ac)' \Rightarrow c'', \quad c'' \Rightarrow c, \tag{18}$$

where $a, c \in A$, $(ac)' \in A'$, $c'' \in A''$. These changes are performed by the execution of a sequence of microoperations: $\Theta_i^0(m_p)$, $\Theta' - i(m_p)$, $\Theta_i''(m_p)$.

The following restrictions are thus seen to be imposed on the applicability of the simulating PSS.

1. An execution of $\Theta_i^0(m_p)$ is allowed only if the states of all input neighbours of m_p belong to A^0 or to A'. This condition makes sure that the cell has already read out all initial t-aged states from its input neighbours, which allows the cell to make a decision whether Θ_i^0 is applicable. If the decision is positive, then the new state $(ac)'$ should contain both states: a (which is the initial t-aged state) and c (which is an initial $t+1$-aged state). The first is needed because there are input cell-neighbours which have not yet read out the cell-state of the current age. The second is needed for saving the knowledge of the cell-state of the next age. Otherwise, after the applicability test is over the information about the applicable substitution is lost.

2. An execution of $\Theta'(m_p)$ is allowed only if the states of all input neighbours of m_p belong to A' or to A''. The substitution of $((ac)', m_p)$ for (c'', m_p) makes sure that all neighbours have already perceived the initial t-aged state a. Hence, the cell state may lose the information about the state a, preserving only that about the state c.

3. An execution of Θ_i'' is allowed only if all input neighbours of the cell named m_p have the states from A'' or from A^0. The substitution of (c'', m_p) for (c, m_p) indicates that the transition to the age $t+1$ has occured.

Proposition 3. *In the simulating computation a cell undergoes the transition from the age t to the age $t+1$ no matter whether the initial t-aged state is to be changed or not.*

It means that even in the case when at the t-th iteration of a synchronous computation the state of the cell named m_p is not to be changed (the cell is said to be *idle*), nevertheless, in the simulating computation it should undergo three changes

$$a \Rightarrow (\lambda a)', \quad (\lambda a)' \Rightarrow a'', \quad a'' \Rightarrow a.$$

The above three propositions determine the extension of the alphabet A to $\tilde{A}$, as well as the substitutions of a simulating PSS Φ'.

According to Proposition 1 the alphabet $\tilde{A}$ (with $r = 3$) consists of three subsets: $\tilde{A} = A^0 \cup A' \cup A''$, each pair of them having empty intersection, where

$$A^0 = A \cup \{\lambda, -\}, \quad A' = \delta'(A^0 \times A^0), \quad A'' = \delta''(A^0), \tag{19}$$

where δ' and δ'' are one-to-one mapping. Further, to be short, they are denoted as follows:

$$\delta'(a, b) = (a, b)' \quad \text{and} \quad \delta''(a) = a''.$$

In order to represent substitutions of Φ' in a compact form let us use some additional symbols, which are logical constructs of symbols from $\tilde{A}$, if regarded as variables

$$\begin{array}{lllll} \bar{a} & \text{is either} & a & \text{or} & (a-)', \\ (\bar{ab})' & \text{is either} & (ab)' & \text{or} & b'', \\ \bar{a}'' & \text{is either} & a'' & \text{or} & a, \end{array}$$

where a, b are arbitary symbols in A^0.

A substitution containing a configuration component with $\bar{a}$ as a state should be regarded as two substitutions: one of them having a instead of $\bar{a}$, and the other – having $(a, -)'$ instead of $\bar{a}$. Thus, the above additional symbols allow to enclose in a single expression a set of substitutions corresponding to all possible combinations of states in the configuration components instead of symbols with overbars.

Since a functional substitution is also to be used, $\tilde{A}$ should be argumented by a set of variables $\{x_1, \ldots, x_q\}$ and a function $y = F(x_1, \ldots, x_q)$ with the domain in $\tilde{A}$.

Let the substitutions $\Theta_i \in \Phi$ be stationary with the identical first naming function $\varphi_1(m) = m$, and be given in the following form:

$$\Theta_1 : \{(a_1, m) \ldots (a_n, \varphi_n(m))\} * \{(b_1, \psi_1(m)) \ldots (b_p, \psi_p(m))\} \Rightarrow \{(c_1, m) \ldots (c_n, \varphi_n(m)\}.$$

Let the input and output neighbourhoods (Definition 7) be as follows:

$$N'(m) = \{\phi_1, \ldots, \phi_u\} \quad \text{and} \quad N''(m) = \{\phi_1, \ldots, \phi_v\}$$

respectively.

Then according to Proposition 2 each substitution $\Theta_i \in \Phi$ has in the simulating PSS two following corresponding ones:

$$\begin{aligned} \Theta_i^0 : \; & \{(a_1, m)(a_2, \varphi_2(m)) \ldots (a_n, \varphi_n(m))\} * \\ & \{(\bar{b}_1, \psi_1(m)) \ldots (\bar{b}_p, \psi_p(m))\} \Rightarrow \\ & \{((a_1 c_1)', m) \ldots ((a_n c_n)', \varphi_n(m))\}; \\ \Theta_i'' : \; & \{(c_1 \prime\prime, m)(c_2'', \varphi_2(m)) \ldots (c_n'', \varphi_n(m))\} * \\ & \{(\bar{b}_1'', \psi_1(m)) \ldots (\bar{b}_p'', \psi_p(m))\} \Rightarrow \\ & \{(c_1, m) \ldots (c_n, \varphi_n(m))\}. \end{aligned} \tag{20}$$

The substitution Θ_i^0 simulates the applicability recognition of $\Theta_i \in \Phi$, the substitution Θ_i'' simulates the act of replacing states. According to Proposition 2 there should be one substitution more which checks whether all input neighbours have already read out the cell state of the current age. This substitution, let it be Θ', is one and the same for all $\Theta_i \in \Phi$, and hence, it can be expressed in terms of a functional substitution as follows:

$$\Theta' : \{(x_1, z_1)', m\} * \{((\overline{x_2, z_2})', \phi_2^{-1}(m)), \ldots, ((\overline{x_u, z_u})', \phi_u^{-1}(m))\} \Rightarrow \{(z_1'', m)\}, \tag{21}$$

where x_j, z_j are variables in A^0, $\phi_j^{-1}(m)$ are inverses of input neighbourhood components, $j = 1, \ldots, u$.

Three more substitutions should be added for simulating the cell transition to the next age without changing the state. The indication of a cell idleness is obtained by executing the following functional substitution:

$$\Omega^0 : \{(x_1, m)\} * \{(\bar{x}_2, \phi_2(m)) \ldots (\bar{x}_u, \phi_u(m))\} \Rightarrow \{(f(x_1, \ldots, x_u), m)\}, \quad (22)$$

where $x_1, \ldots, x_u$ are variables in $\tilde{A}$; $\phi_1(m), \ldots, \phi_u(m)$ are naming functions determining the set of input neighbourhood components of a cell named m

$$f(x_1, \ldots, x_u) = \begin{cases} (\lambda x_1)', & \text{if no substitution is applicable at } m, \\ — & \text{otherwise.} \end{cases}$$

For preserving the same initial state when entering the next state no microoperations should be applicable not only at the cell itself, but also at all those cells, to whose output neighbourhoods the cell belongs. Hence, the transition of a cell from the state of A' to a state of A'' caused by its output neighbour is performed by the following substitution:

$$\Omega' : \{((\lambda x_1)', m))\} * \{((\overline{\lambda x_2})', \psi_2^{-1}(m)) \ldots ((\overline{\lambda x_v})', \psi_v^{-1}(m))\} \Rightarrow \{((x_1)'', m)\}, \quad (23)$$

where $x_1, \ldots, x_v$ are variables in A and $\psi_1^{-1}(m), \ldots, \psi_v^{-1}(m)$ are the inverse naming functions, determining the output cell neighbourhood.

The transition to the next age is completed by the execution of the substitution

$$\Omega'' : \{(x_1'', m)\} * \{(\bar{x}_2^2 \prime\prime, \phi_2^{-1}(m)) \ldots (\bar{x}_v'', \phi_v^{-1}(m))\} \Rightarrow \{(x_1, m)\}. \quad (24)$$

Theorem 2. *A synchronous computation* $\bar{\Phi}(W)$, $\Phi = \{\Theta_1, \ldots, \Theta_\nu\}$, $W \in K(A, M)$, *is simulated by the asynchronous one* $\tilde{\Phi}'(V)$, $V \in K(\tilde{A}, M)$, $A \subset \tilde{A}$, *if the following conditions hold:*

1) $\tilde{A} = A^0 \cup A' \cup A''$, *constructed according to* (19),
2) $\Phi' = \{\Theta_1^0, \ldots, \Theta_\nu^0, \Theta_1'', \ldots, \Theta_\nu'', \Theta', \Omega^0, \Omega', \Omega''\}$, *all substitutions being obtained according to* (20)–(24).

Proof. According to Definition 19 it is sufficient to show that for any $m_p \in M$ all $\tilde{\sigma}_i(m_p)$ in $\tilde{\Phi}'(V)$ are ξ_3-extensions of $\bar{\sigma}(m_p)$ in $\bar{\Phi}(W)$. It means that any $\tilde{\sigma}_i(m_p)$ meets five conditions of Definition 18. The first three conditions, concerning the properties of the alphabet follow immediately from the construction of $\tilde{A}$ according to (19). The forth condition is evident from the construction of the substitutions. As for the fifth condition, it looks as follows: $\Pr(h(\tilde{\sigma}_i(m_p) \Rightarrow A) = \bar{\sigma}(m_p)$ for all $\tilde{\sigma}_i(m_p)$ in $\Phi'(W)$. According to Definitions 16 and 17 the left-hand side represents the sequence of initial states in $\tilde{\sigma}_i(m_p)$. This sequence is one and the same for all $\tilde{\sigma}_i(m_p)$ in $\tilde{\Phi}'(V)$, which follows from the determinacy of each transition from a t-aged

from the fact that the substitutions in Φ' have no critical intersections by construction, which is clearly seen from (20)–(24). Besides, from Φ' it is evident that each replacement of the t-aged initial state of a cell named m_p results in the same state that the cell with the same name has after the t-th iteration in $\bar{\Phi}(W)$. Since all above is true for any $m_p \in M$, all conditions of Definition 18 are fulfilled, which completes the proof. □

Example 7. An asynchronous PSS simulating the PSA $= \langle \Phi, K(A, M) \rangle$, where $\Phi = \{\Theta_1, \Theta_2\}$, $A = \{a, b, c\}$, $M = \mathbf{N}$,

$$\Theta_1 : \{(c, i)\} * \{(a, i-1)\} \Rightarrow \{(b, i)\}; \quad \Theta_2 : \{(a, i)\} \Rightarrow \{(b, i)\};$$

is to be constructed.

Using the methods from [4] it is easy to check, that the asynchronous mode of execution of Φ is not determinate. It is clearly seen from the computation graphs $\bar{\Phi}(W)$ and $\tilde{\Phi}(W)$ with $W = \{(a, 1)(c, 2)\}$ which are shown in Figure 7 and Figure 8 respectively.

The PSS Φ' simulating the given PSA is constructed as follows.

The alphabet A is expanded to $\tilde{A} = A^0 \cup A' \cup A''$, where $A^0 = \{a, b, c, \lambda\}$, $A' = \{(aa)', (ab)', (cc)', (cb)'\}$, $A'' = \{a'', b'', c''\}$. For the sake of the simulating PSS compactness the following additional symbols are also used.

$\bar{z}$	which stands for either z or$(z-)'$,
$(\overline{zy})'$	which stands for either $(zy)'$ or y'', and
z''	which stands for either z'' or z,

where z, y are any symbols from A^0.

The substitutions simulating Θ_1 and Θ_2 are obtained according to (20) and (21):

$$\Theta_1^0 : \{(c, i,)\} * \{(\bar{a}, i-1)\} \Rightarrow \{((cb)', i)\};$$

$$\Theta_1'' : \{(b'', i)\} * \{(\bar{a}'', i-1)\} \Rightarrow \{(b, i)\};$$

$$\Theta_2^0 : \{(a, i)\} \Rightarrow \{((ab)', i)\};$$

$$\Theta_2'' : \{(b'', i)\} \Rightarrow \{(b, i)\};$$

$$\Theta' : \{((x_1 z_1)', i))\} * \{((\overline{x_2, z_2})', i-1))\} \Rightarrow \{(x_1'', i)\}.$$

The computation graph of $\tilde{\Phi}(W)$ is shown in Figure 9. It is seen from the graph that there is no idle cells in the cellular arrays of this

Figure 7

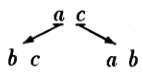

Figure 8

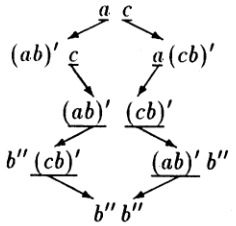

Figure 9

computation. Hence, the substitutions of the type Ω^0, Ω' and Ω'' are not needed. So $\Phi' = \{\Theta_1^0, \Theta_1'', \Theta_2^0, \Theta_2'', \Theta'\}$.

The complexity of a simulating PSA is easily assessed from the above method of its construction. Let μ be the amount of bits needed for storing the state from A, $\nu = |\Phi|$. Then the amount of bits for storing a symbol from $\tilde{A}$ and the number of substitutions of Φ' are respectively the following:

$$\mu' \geq 2\mu + 3, \quad |\Phi'| \geq 2\nu + 4. \tag{25}$$

As for the amount of steps, it is three times larger, but this increase may be compensated by the absence of common clock, which yields shorter steps. Moreover, this assessment is made for the general case. Very often the good behavioral properties allow to find the PSS with much better complexity, whose asynchronous mode of execution is equivalent to a given PSA.

References

[1] S.M. Achasova, Correctness of interpretations of parallel substitution systems, J. New Gener. Comput. Syst., 1, 1991, 19–27.

[2] S.M. Achasova, O.L. Bandman, Correctness of Parallel Computation Processes, Nauka, Novosibirsk, 1990 (in Russian).

[3] S.M. Achasova, Correctness of synchronous cellular computations, Parallel Computing Technologies, Proceedings of the International Conference, August 30, September 4, 1993, Ed. V.E. Malyshkin, Computer Center, Novosibirsk, Russia, Vol. 3.

[4] O.L. Bandman, S.V. Piskunov, Parallel substitution algorithms as a model for distributed computations, J. New Gener. Comput. Syst., 4, 1991, 1, 3–18.

[5] Methods of Parallel Microprogramming, Ed. O.L. Bandman, Nauka, Novosibirsk, 1981 (in Russian).

[6] R.J. Lipton, R.E. Miller, L. Snyderr, Synchronization and Computing Capability of Linear Asynchronous Structures, J. Comput. and Syst. Sci., 14, 1977, 49–72.

[7] U. Golze, (A)synchronous (Non)deterministic Cell spaces simulating each other, J. Comp. and Syst. Sci., 17, 1978, 176–193.

[8] P. Rosenstiel, J.R. Fiskel, A. Holliger, Intelligent graphs: network of finite automata capable of solving graphs problems, Graph Theory and Computers, Ed. R. Reed, Acad. Press, N–Y, 1972, 219–265.