Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 11 - 22, 2007.

How to Synchronize the Activity of All Components of a P System?

Francesco Bernardini[1], Marian Gheorghe[2], Maurice Margenstern[3], and Sergey Verlan[4]

[1]Leiden Institute of Advanced Computer Science, Universiteit Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl

[2]Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk

[3]Université Paul Verlaine - Metz, LITA, EA 3097, IUT de Metz
Ile du Saulcy, 57045 Metz Cédex, France
margens@univ-metz.fr

[4]LACL, Département Informatique, Université Paris 12
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract

We consider the problem of synchronizing the activity of all the membranes of a P system. After pointing at the connection with a similar problem dealt with in the field of cellular automata where the problem is called the *firing squad synchronization problem*, *FSSP* for short, we provide two algorithms to solve this problem. One algorithm is non-deterministic, the other is deterministic. Both work in a time which is $3h$, where h is the height of the tree defining the membrane structure of the considered P system. In the conclusion, we suggest various directions to continue this work.

1 Introduction

The synchronization problem can be formulated in general terms with a wide scope of application. We consider a system constituted of explicitly identified elements and we require that starting from an initial configuration where one element is distinguished, after a finite time, all the elements which constitute the system reach a common feature, which we call **state**, all at the same time and the state was never reached before by any element.

This problem is well known for cellular automata, where it was intensively studied under the name of the *firing squad synchronization problem* (FSSP): a line of soldiers have to fire at the same time after the appropriate order of a general which stands at one end of the line, see [2, 7, 6, 11, 12, 13]. The first solution of the problem was found out by Goto, see [2]. It works on any cellular automaton on the line with n cells in the minimal time, $2n-2$ steps, and requiring several thousands of states. A bit later, Minsky found his famous solution which works in $3n$, see [7] with a much smaller number of states, 13 states. Then, a race to a cellular automaton with the smallest number of states which synchronizes in $3n$ started. See the above papers for references and for the best results and for generalizations to the planar case, see [11] for results and references.

The synchronization problem appears in many different contexts, in particular in biology. As P systems modelize the working of a living cell constituted of many micro-organisms, represented by its membranes, it is a natural question to raise the same issue in this context. Take as an example the meiosis phenomenon, it probably starts with a synchronizing process which initiates the division process. Many studies have been dedicated to general synchronization principles occurring in cell cycle; although some results are still controversial, it is widely recognised that these aspects might lead to an understanding of general biological principles used to study the normal cell cycle, see [10].

Apparently, this problem was never studied in the framework of P systems.

Our first idea was to implement the well known solutions for cellular automata on the line. This idea was used in [3, 5] in the context of cellular automata in the hyperbolic plane in order to synchronize sets of cells which are more complex than a line. However, in this context, the sets of cells are trees in which all branches have the same length which allows to immediately implement the algorithms for cellular automata on the line, thanks to the parallelism of computation of the cells of a cellular automaton. It is not that difficult, although not immediately straightforward, to implement an algorithm for linear cellular automata into the membrane structure of a P system, even in the easy case when the tree of the membrane structure is complete, *i.e.* all its branches have the same length. To extend this to any P system, we devised two solutions. The first idea was to complete the tree. The second idea was to use various delay strategies considered in generalized firing squad problems for cellular automaton when the general is at an arbitrary position in the line of soldiers, see [12]. In any case, a direct transcription of a CA solution might use a kind of boundary rules, see [1]. This might be slightly combined with sending appropriate symbols, current states to neighbours such as to make use of them in each cell component, but this will double the synchronization time.

Then, we had a second thought. Why not trying to find a solution, more specific to P systems? In the paper, we give two algorithms to solve the synchronization problem for P systems. One algorithm is non-deterministic while the other is deterministic. However, the working of our deterministic algorithm raises an interesting discussion motivated by the implementation of the solution into a computer program. It is also interesting to notice that both our algorithms work in $3n$.

Before, turning to the algorithms, let us discuss again the setting of the problem in the frame of P systems and how we can recognize the configuration when all the membranes are synchronized.

We shall implement the "fire" state of cellular automaton, traditionally denoted by F as an object which will appear at the time of synchronization, but never before this time. Of course, such an object must occur in at least one rule. But still, this condition is a good implementation of the cellular automaton process. Moreover, if the objects of the membranes are strings, it will not be difficult to adapt the rules of the two algorithms into rules in which F never occurs. We simply decide that F is also a string and it is obtained through the rules by a computing process. This latter process is a complication which we considered as not very informative. This is why we restrict ourselves to the situation where F is an object.

2 Definitions

In the following we briefly recall the basic notions concerning P systems. For more details on these systems and on P systems in general, we refer to [8].

An evolution-communication P system of degree n is a construct

$$\Pi = (O, E, \mu, w_1, \dots, w_n, R_1, \dots, R_n, i_0),$$

where:

1. O is a finite alphabet of symbols called objects,
2. μ is a membrane structure consisting of n membranes that are labelled in a one-to-one manner by $1, 2, \dots, n$,
3. $w_i \in O^*$, for each $1 \le i \le n$ is a multiset of objects associated with the region i (delimited by membrane i),
4. $E \subseteq O$ is the set of objects called environment; each element appears in an infinite number of copies,
5. R_i, for each $1 \le i \le n$, is a finite set of rules associated with the region i and which have the following form $u \to v_1, tar_1; v_2, tar_2; \dots; v_m, tar_m$, where $u \in O^+$, $v_i \in O$ and $tar_i \in \{in, out, here, in!\}$,
6. i_0 is the label of an elementary membrane of μ that identifies the corresponding output region.

An evolution-communication P system is defined as a computational device consisting of a set of n hierarchically nested membranes that identify n distinct regions (the membrane structure μ), where to each region i there are assigned a multiset of objects w_i and a finite set of evolution rules R_i, $1 \le i \le n$.

An evolution rule $u \to v_1, tar_1; v_2, tar_2; \dots; v_m, tar_m$ rewrites u by $v_1, \dots, v_m$ and moves each v_j accordingly to the target tar_j. If the tar_j target is *here*, then v_j remains in membrane i. If the target tar_j is *out*, then v_j is sent to the parent membrane of i. If the target tar_j is *in*, then v_j is sent to any inner membrane of i chosen non-deterministically. If the target tar_j is equal to *in*!, then v_j is sent to all inner membranes of i (a necessary number of copies is made).

A computation of the system is obtained by applying the rules in a non-deterministic maximally parallel manner. Initially, each region i contains the corresponding finite multiset w_i; whereas the environment contains only objects from E that appear in infinitely many copies.

A computation is successful if starting from the initial configuration it reaches a configuration where no rule can be applied. The result of a successful computation is the natural number that is obtained by counting the objects that are presented in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$.

An *evolution-communication P system with polarizations and priorities* of degree n is a construct

$$\Pi = (O, E, \mu, w_1, \dots, w_n, R_1, \dots, R_n, i_0),$$

defined as in the previous definition. However, in addition to that definition each membrane has a label from the set $\{0, +, -\}$ called *polarization*. Initially, all membranes have the polarization 0.

Moreover, the set of rules may contain rules of the form

$u \to v_1, tar_1; v_2, tar_2; \dots; v_m, tar_m$,

where $u \in O^+$, $v_i \in O$ and $tar_i \in \{in_+, mark_+, out, here, in!\}$.

As above, the *here* target means that the object remains in the current membrane and the *in*! target sends the corresponding object to all inner membranes at the same time (making the right number of copies). The *out* target sends the object to the outer membrane and changes at the same time the polarization of membrane to $-$. The in_+ target sends the object to an inner membrane having a $+$ polarization. The $mark_+$ target leaves the objects in the same membrane and, at the same time, it takes one inner membrane having a 0 polarization and changes its polarization to $+$.

Each rule has also a priority which is a natural number. A computational step is obtained by applying the rules in a non-deterministic maximally parallel manner where a rule with a lower priority cannot be applied if a rule of a higher priority is applicable.

In the following we shall restrict our systems to systems where there are at least 2 membranes and all membranes contain the same set of rules and the same set of objects, except the skin. The skin may contain a different set of objects.

We shall also try to satisfy the following goal: starting from the initial configuration where only the skin has some differentiated objects the system halt at some moment. In the halting configuration all membranes contain the same symbol(s) which should not appear before.

3 Non-Deterministic Solution

In this section we discuss a non-deterministic solution to the FSSP using evolution-communication P systems. We shall use the following algorithm (consider the P system as a tree):

Algorithm 1

1. Starting from the root find an arbitrary leaf.

2. Compute the depth of this leaf. Let n be this number.
3. For any node (initially the root) do the following steps:
4. Decrement the counter b (initially equal to n).
5. Make a local copy of b (and call it b').
6. If the current node is not a leaf then send counter b to all inner nodes.
7. At each following step decrement the local copy of b (b').
8. If the local copy of b is equal to zero, then introduce the final symbol F.
9. If at some moment b is equal to zero, while there are inner nodes, then do not stop (perform an infinite computation).

The idea of the algorithm is to guess the longest branch (having the length equal to the height of the tree, *i.e.* to the length of the longest path from the root to a leaf) and after that to propagate this height from the root to the leaves decreasing it at each level. For the synchronization a copy of this height is kept at each visited node and decreased at each step. When all these counters are zero, we may synchronize by introducing the symbol F. If the guess was wrong, then the system will never halt because the symbol # will be introduced.

Now let us present the system in details.

Let $\Pi = (O, E, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ the P system to be synchronized, where i_0 is not mentioned as it is not relevant for the synchronization. To solve the synchronization problem, we make the following assumptions on the objects, the environment, the membranes and the rules. We considerthat:

$O = \{L, Y, R, S_1, S_2, S_2', S_3, S_4, S_5, S_6, S_7, F, a, b, b', \#\}$, $E = \emptyset$,
and that μ is an arbitrary membrane structure, with w_1 as the skin membrane. We also assume that $w_1 = \{L, Y, R, S_1, S_2'\}$ and that all other membranes satisfy $w_i = \{Y, L\}$. The sets of rules, R_1, …, R_n are all equal and they are described below.

The rules:

Finding a leaf:

$$\begin{array}{ll} S_1 \to S_1, in & S_2' \to S_2, here \\ S_2 \to S_2, in & S_2 \to \#, here \end{array} \tag{1}$$

Computing the depth of the leaf:

$$S_1S_2 \to S_3, out; a, out \qquad S_3 \to S_3, out; a, out \qquad a \to a, out \tag{2}$$

$$S_3R \to S_5, here \qquad a \to b, here \tag{3}$$

Propagation of the signal:

$$LYS_5b \to S_5, in!; S_6, here \qquad b \to b, in!; b', here \qquad S_5 \to \#, here \tag{4}$$
$$LYS_5b \to S_7, here \tag{5}$$

Counting back:

$$S_6b' \to S_6, here \qquad S_6 \to F, here \qquad Fb' \to \#, here \tag{6}$$
$$S_7b \to S_7, here \qquad S_7 \to F, here \qquad Fb \to \#, here \tag{7}$$

Traps:

$$bY \to \#, here \qquad b'Y \to \#, here \qquad \# \to \#, here \tag{8}$$

Infinite loop:

$$L \to L, here \tag{9}$$

Rules (1) permit to find an arbitrary leaf. Indeed, two signals S_1 and S_2 descend the tree, but signal S_2 has a one-step delay with respect to S_1. They may meet only at a leaf, where the *in* target is not applicable. If this does not happen because of the non-deterministic descent, then S_2 will be transformed to the trap symbol #. When signals S_1 and S_2 meet, a new signal S_3 is produced. This signal moves up until the root node, where the *out* target is not applicable. When moving up, at each step, a new symbol a is produced. Hence, when S_3 reaches the root, n copies of a will be present, n being the height of the leaf reached by S_1. Rules (2) permit to do this. In this way steps 1 and 2 of the algorithm are implemented.

At the root node, the symbol S_3 is transformed to S_5 and all symbols a are transformed to b by rules (3). If this last transformation happens before the root node, then symbols b or b' which are obtained from them will be trapped by rules (8). The steps 4-6 of the algorithm are implemented by rules (4) and (5). Indeed, symbols b are replicated and sent to all inner membranes and the same number of symbols b' is created. At the same time, one symbol b is used together with S_5, L and Y. In this way the number of b's is decreased at each step. Symbol S_5 is transformed to S_6 or S_7 (if we reached a leaf).

Steps 7 and 8 of the algorithm are implemented by rules (6) and (7) (for the leaf). The number of symbols b' (b for the leaf) is decreased and when it reaches zero, symbol S_6 (S_7 for the leaf) is transformed to F. If this rule is applied before all symbols b' (resp. b) are consumed then the trap symbol # is introduced.

Now we shall present some assertions that guarantee the correctness of the proof, keeping in mind that a correct computation always halts.

- If symbol S_3 does not appear, then the computation never halts.

  Indeed, in this case, symbol S_2 will reach a leaf different from the one reached by S_1 and it will be transformed to the trap symbol.

- Rules (3) may be applied only at the root node and the number of symbols b which is obtained is equal to n, where n is the depth of the leaf visited by S_1.

  Indeed, the first rule uses the symbol R which is present only at the root node. If the second rule is applied at a non-root node, then at least one symbol b is introduced. By the second rule from (4) at least one copy of symbol b' will appear in the same node. Now it suffices to remark that this transformation takes two steps and it is clear that symbol S_5 cannot appear in the meanwhile because if the current node is not the root node, then at least 3 steps are needed to transform symbol S_3 into S_5 and propagate it down. Hence, the symbol Y will be present and the trap symbol will be introduced by the second rule from (8).

  Since at each step the number of a's is increased, at the root node it will be equal to the depth of the starting leaf, *i.e.* the one visited by S_1.

- The first rule from (4) must be applied when at least one b occurs in this node.

  Indeed, otherwise S_5 will be left alone either in the current node, or in the inner nodes. In this case, the third rule from (4) will introduce the trap symbol.

- Rule (5) may be applied only at the leaf.

  Indeed, if it is applied at a node which is not a leaf, then symbols L situated in membranes below cannot be eliminated and the system will always perform an infinite computation because of the rule (9).

- The second rule from (6) (resp. (7)) may be applied if and only if the number of b' (resp. b) at that node is zero.

  It is clear that if these rules are applied before, then the third rule from (6) (resp. (7)) will introduce the trap symbol.

- After the introduction of the symbol S_5, at each step k, $0 \leq k \leq n$ the configuration of nodes having the depth $h < k$ is $\{S_6, b'^{n-k}\}$ ($\{S_7, b^{n-k}\}$ if the node is a leaf) and the configuration of nodes having the depth k is $\{L, Y, S_5, b^{n-k}\}$, where n is the depth of the leaf visited by S_1.

  This assertion may be easily verified by induction. Initially, at the step 0, the root node contains $\{S_5, b^n\}$. Suppose that the assertion holds for $k < n$. Consider all nodes of depth k that are not leaves. In this case rules (4) are applicable and the configuration of these nodes becomes $\{S_6, b'^{n-k-1}\}$. The configuration of the inner nodes, having the depth $k+1$, becomes $\{L, Y, S_5, b^{n-k-1}\}$. Consider now all leaves of depth smaller or equal than k. By the first rule from (7) their configuration becomes $\{S_7, b^{n-k-1}\}$. Now consider all non-leave nodes of depth smaller than k. By the first rule from (6) their configuration becomes $\{S_6, b'^{n-k-1}\}$.

From the above assertions it is clear that if a leaf not corresponding to the longest branch of the tree is reached by S_1, then the system will never halt (because some symbols L will be present). If the initial guess corresponds to the longest branch, then it is clear that all nodes will reach the same configuration $\{F\}$ at the same time (because they contain the same number of symbols b or b'). This concludes the proof.

Theorem 1. *The time complexity of the just considered algorithm is $3h$, where h is the height of the tree of μ, the membrane structure.*

Proof. The detection of the longest branch takes $2h$ steps: h steps to go to the farthest leaf and h ones to get the feed back. Then, the synchronizing process takes h steps. □

We recall that the time complexity taken into consideration is the number of steps of the computation. Now, note that $h = \log n$ for a complete tree and that $h = n$ in the worst case.

4 Deterministic Solution

In this section we show a deterministic solution to FSSP. We shall use a different class of P systems, namely evolution-communication P systems with polarizations and priorities.

We use the following algorithm to solve the problem.

1. Find the height of the tree.
2. Descend and distribute symbols like in the non-deterministic case.

In order to find the height of the tree we use the following algorithm:

Algorithm 2

1. Start at the root node. Counting = false, height=0.
2. If there are non-marked inner nodes then go to any of them. If counting= true and height>0, then height=height-1;
3. If at leaf and counting = false, then counting = true.
4. If all inner nodes are marked, then mark the current node, and if the current node is not root go up and height=height+1.
5. If the root is marked then stop.

Theorem 2. *The above algorithm correctly computes the height of a tree.*

Proof. The proof will be done by induction on the height of the tree. Consider a tree of height 1. Obviously, the first chosen node is a leaf and the counting starts. It is easy to see that the value of height will oscillate between 0 and 1, the value 1 appears when we are at the root node.

Now let us suppose that the algorithm returns the height of a tree for any tree having the height $<= n$. Now suppose that we have a tree of height $n + 1$. Let R be the root node and $F_1, \ldots, F_k$ be the children of R. Clearly, each F_t, $1 \leq t \leq k$

is a root node for a tree A_t of height, for instance, h_t. Now let F_i be the first node chosen at step 2 of the algorithm. By induction the algorithm reaches the step 4 with counting=true and height=h_i. After that we move up to the root R and the value of height is h_i+1. Now let F_j be another node chosen at step 2 of the algorithm and h be the current value of height. If $h_j \leq h-1$ then at the deepest node of A_j the value of height is equal to $h-1-h_j$ and when we return to R the value of height is equal to h. If $h_j > h-1$ then at the deepest node of A_j the value of height will be 0 and when we return to R the value of height will become equal to h_j.

Hence, at the last visit of R (we remark that we do not return to A_m that were already visited) the value of height is $1+\max(h_1,\ldots,h_k)$ which is the height of the initial tree. This concludes the proof. □

Before going into the precise description of the P system, we have to focus on what we mean by **deterministic**. In fact, the algorithm which we shall use to determine the height of the tree to be synchronized contains the possibility of an arbitrary choice at some steps of its execution. What happens is, whatever the choice, the result is always the same. Now, if we wish to implement this algorithm in a computer program in order to use it, the implementation must define a rule to fix the choice. Accordingly, the execution of the algorithm by a simulating device is deterministic. This is why we consider the algorithm as deterministic, although its presentation is not.

Let us describe the algorithm to compute the height of the tree in an informal way.

At the beginning, all membranes have the polarity 0. The algorithm repeatedly performs the following sequence of actions:

When the control arrives at a membrane M, it looks whether there is at least one child-membrane with polarity 0. If there is at least one such membrane, the algorithm selects one of them, N, and it changes the polarity of N to $+$. Then, it decreases the height by 1, unless it is already 0, in which case the height remains unchanged. If all child-membrane are with the polarity $-$, the control goes back to the parent membrane of M, changes the polarity to $-$ and the height is increased by 1.

The loop is stopped when the control arrives at the skin membrane and all child-membranes are with the polarity $-$.

The choice of the child-membrane whose polarity is turned from 0 to $+$ is the non-deterministic operation. However, the result of the algorithm does not depend on which membrane has been chosen. It is enough to select one of them, whatever the membrane. Now, in the context of a biological environment where some protein would choose a membrane M to perform some operation on M, there are probably additional factors which determine the choice performed by the protein. And so, the choice may be considered as deterministic. This corresponds to the implementation which we above invoked.

This choice is formalized by an operator $mark_+$ which selects one child-membrane with polarity 0 if any, and changes its polarity to $+$. The operator $mark_+$ has the highest priority. Now, note that when the control leaves the membrane M where $mark_+$ was invoked, it changes the polarity $+$ to $-$. Accordingly, when $mark_+$

successfully performed the change on one child-membrane of M, a single membrane has the polarity $+$ among the child-membranes of M.

This can be implemented by the following rules (all nodes are initially empty, except the root containing symbol S_1). The number at the left indicates the priority of the rule (a higher number means a higher priority).

Finding the first leaf:

$$\begin{array}{llll} 2 & S_1 \to S_1', mark_+ & S_1' \to S_1, in_+ & (10) \\ 1 & S_1 \to S_2 & & (11) \end{array}$$

Counting algorithm:

$$\begin{array}{llll} 5 & S_2 \to S_2', mark_+ & S_2'a \to S_2, in_+ & (12) \\ 4 & a \to a, in_+ & & (13) \\ 3 & S_2' \to S_2, in_+ & & (14) \\ 2 & S_2 \to S_2, out; a, out & a \to a, out & (15) \\ 1 & S_2 \to S_3, here & a \to b, here & (16) \end{array}$$

Distribution:

$$\begin{array}{llll} 2 & S_3b \to S_3, in!; S_4, here & b \to b, in!; b', here & (17) \\ 1 & S_3b \to S_5, here & & (18) \end{array}$$

Counting down:

$$\begin{array}{llll} 2 & S_4b' \to S_4, here & S_5b \to S_5, here & (19) \\ 1 & S_4 \to F, here & S_5 \to F, here & (20) \end{array}$$

Let us discuss the functioning of the above system. Rules (10) permit to move the symbol S_1 down until it reaches a leaf. This corresponds to the step 2 of Algorithm 2. When the leaf is reached, the rule (11) becomes applicable and S_1 is transformed to S_2. This corresponds to the step 3 of Algorithm 2. Rules (12)-(15) permit to implement the steps 2-4 of Algorithm 2 when counting is equal to true. Indeed, a membrane having polarization 0 (not yet visited) is chosen and height is decreased (the value of height is represented by the number of symbols a) by rules (12) and (13). If height is equal to zero, then rule (14) is applicable which simply moves symbol S_2 down. Rules (15) are applicable only if we are at a leaf or all inner membranes have the polarization $-$. In this case S_2 is moved up and the number of a's is increased by one. This corresponds to the step 4 of the algorithm. Rules (16) are applicable only at the root node when all children were visited. This corresponds to rule 5 of the algorithm 2 and the number of a's corresponds to the height of the tree. So, rules(16) rename a to b and change S_2 to S_3.

Now a propagation phase, similar to the non-deterministic solution, starts. Rules (17) propagate down the counter b and decrease it at the same time, while rules (19) decrement the local copy of b (consisting of symbols b'). Like in the non-deterministic case, rules (20) will be applicable after h steps from the beginning of the propagation phase, where h is the height of the tree of the P system, and the final symbol F will be synchronously introduced in all membranes.

Theorem 3. *The time complexity of the just considered algorithm is $3h$, where h is the height of the tree of μ, the membrane structure.*

Proof. It is exactly the same as for the non-deterministic algorithm. To detect the longest branch, we have at most to go down to the farthest leaf and then to go back to the root. $\square$

We have the same remarks on the time complexity and on h as for theorem 3.1.

5 Conclusions

In this paper, we indicated two algorithms to perform the synchronization of all the membranes of a given P system. It is also interesting to notice that the algorithms are not only linear, they are in $3h$, h being the height of the tree defined by the membrane structure of the P system. In the case of a linear structure, $h = n$. Now, the time of $3n$ is the time of many algorithms for the $FSSP$ in the case of cellular automata on the line with n cells. In this domain, the optimal time is $2n$. And so, a natural question is: why not trying to do reach $2h$ for a P system?

Other directions are in a possible reduction in the number of rules used by the algorithms. Another direction is to look at the possibility to define a deterministic synchronization algorithm without using priority rules. As an example, it would be interesting to see whether there is specific solution with symport/antiport P systems which are considered as very close to the real activity of the cell membranes. Another direction is the extension of the result to tissue P systems. It is not difficult to adapt our algorithms to the case a convex graph in which it is possible to implement a tree structure. Taking an exploring algorithm, it is possible to cut the possible cycles in order to get a covering tree of the graph which would be in bijection with the vertices of the graph, see [4, 9], and then we can apply the algorithms described in the previous sections to solve the synchronization problem.

For sure, new results in any of the above mentioned direction would have direct consequences of the scope of application of the problem. It seems to us that the possibility to synchronize the membranes of a P system in a rather reasonable time is a serious argument in favour of the suitability of the model for biology.

We would not be surprised to see the possibility to closer mimic real biological phenomena at the level of a cell with P systems in a near future.

Acknowledgements

The research of Francesco Bernardini is supported by NWO, Organization for Scientific Research of the Netherlands, project 635.100.006 VIEWS.

References

[1] F. Bernardini, V. Manca. P systems with boundary rules. In Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 02, Curteă de Arge s, Romania, August, 19-23, 2002, Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 107-118. 2003.

[2] E. Goto. A Minimum Time Solution of the Firing Squad Problem. *Course Notes for Applied Mathematics*, 298. Harvard University, 1962.

[3] Ch. Iwamoto, M. Margenstern. Time and Space Complexity Classes of Hyperbolic Cellular Automata. *IEICE Transactions on Information and Systems*, 387-D(3):700–707, 2004.

[4] J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48-50, 1956.

[5] M. Margenstern. An algorithm for building intrinsically universal cellular automata in hyperbolic spaces. *FCS 2006, Las Vegas, June, 2006.*

[6] J. Mazoyer. A Six-State Minimal Time Solution to the Firing Squad Synchronization Problem. *Theoretical Computer Science*, 50:183-238, 1987.

[7] M. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[8] Gh. Păun. *Membrane Computing. An Introduction.* Springer-Verlag, 2002.

[9] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389-1401, 1957.

[10] P.T. Spellman, G. Sherlock. Reply: whole-cell synchronization - effective tools for cell cycle studies. *Trends in Biotechnology*, 22(6):270-273, 2004.

[11] H. Umeo, M. Maeda, N. Fujiwara. An Efficient Mapping Scheme for Embedding Any One-Dimensional Firing Squad Synchronization Algorithm onto Two-Dimensional Arrays. In *ACRI 2002*, volume 2493 of *Lecture Notes in Computer Science*, pages 69-81. 2002.

[12] H. Schmid, T. Worsch. The Firing Squad Synchronization Problem with Many Generals For One-Dimensional CA. In *IFIP TCS 2004*, pages 111-124. 2004.

[13] J.-B. Yunès. Seven-state solution to the firing squad synchronization problem. *Theoretical Computer Science*, 127(2):313-332, 1994.

[14] The P systems web page. `http://psystems.disco.unimib.it`.