International Workshop

Automata for Cellular and Molecular Computing

Budapest, Hungary, August 31, 2007

Edited by Gy. Vaszil

MTA SZTAKI, Budapest, 2007

György Vaszil, editor

International Workshop

Automata for Cellular and Molecular Computing

Budapest, Hungary, August, 2007

Proceedings

MTA SZTAKI
Budapest

Foreword

The international workshop *Automata for Cellular and Molecular Computing* was held on August 31, 2007, in Budapest, Hungary, as a co-located event with the *16th International Symposium on Fundamentals of Computational Theory, FCT 2007.* It was organized by the *Theoretical Computer Science Research Group* of MTA SZTAKI, the *Computer and Automation Research Institute* of the *Hungarian Academy of Sciences.*

The aim of the workshop was to present novel ideas connecting cellular and molecular computing with the more classical areas of automata theory. In this framework, the extended notion of "automaton" includes interaction with an environment or other automata, uses resources represented by multisets, and can have new features inspired by "natural computing".

We would like to thank the members of the program committee, Gabriel Ciobanu (Iasi), Erzsébet Csuhaj-Varjú (Budapest), Rudolf Freund (Vienna), Oscar H. Ibarra (Santa Barbara), Maurice Margenstern (Metz), Giancarlo Mauri (Milan), Gheorghe Păun (Bucharest), Mario J. Pérez-Jiménez (Seville), and György Vaszil (Budapest).

We would also like to thank the authors for participating and presenting their work at the workshop.

Budapest, August 2007 György Vaszil

Contents

Contributions

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

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

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 $1, 2, \ldots, n$,
3. $w_i \in O^*$, for each $1 \leq i \leq n$ 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 $1 \leq i \leq n$, is a finite set of rules associated with the region i and which have the following form $u \to v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m$, 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, $1 \leq i \leq n$.

An evolution rule $u \to v_1, tar_1; v_2, tar_2; \ldots; v_m, tar_m$ rewrites u by $v_1, \ldots, v_m$ 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

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

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

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

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:

$$S_1 S_2 \to S_3, out; a, out \qquad S_3 \to S_3, out; a, out \qquad a \to a, out \tag{2}$$

$$S_3 R \to S_5, here \qquad a \to b, here \tag{3}$$

Propagation of the signal:

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

Counting back:

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

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:

$$2 \qquad S_1 \to S_1', mark_+ \qquad\qquad S_1' \to S_1, in_+ \tag{10}$$
$$1 \qquad S_1 \to S_2 \tag{11}$$

Counting algorithm:

$$5 \qquad S_2 \to S_2', mark_+ \qquad\qquad S_2' a \to S_2, in_+ \tag{12}$$
$$4 \qquad a \to a, in_+ \tag{13}$$
$$3 \qquad S_2' \to S_2, in_+ \tag{14}$$
$$2 \qquad S_2 \to S_2, out; a, out \qquad\qquad a \to a, out \tag{15}$$
$$1 \qquad S_2 \to S_3, here \qquad\qquad a \to b, here \tag{16}$$

Distribution:

$$2 \qquad S_3 b \to S_3, in!; S_4, here \qquad\qquad b \to b, in!; b', here \tag{17}$$
$$1 \qquad S_3 b \to S_5, here \tag{18}$$

Counting down:

$$2 \qquad S_4 b' \to S_4, here \qquad\qquad S_5 b \to S_5, here \tag{19}$$
$$1 \qquad S_4 \to F, here \qquad\qquad S_5 \to F, here \tag{20}$$

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. □

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`.

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 23 - 34, 2007.

Tissue Simulator: A Graphical Tool for Tissue P Systems

Rafael Borrego–Ropero, Daniel Díaz–Pernil,
and Mario J. Pérez–Jiménez

Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence. University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
{rborrego,sbdani,marper}@us.es

Abstract

Recently, different new models of tissue-like P systems have received important attention from the scientific community. This paper is focused in a concrete model: recognizing tissue P system with cell division. A software application allowing to understand better this model is presented. A linear-time solution to an **NP**-complete problem from graph theory, the 3–coloring problem is considered as a case study with this tool.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing introduced by Păun in [10], which, considering as computer process the process that takes place into living cells, constructs a new non-deterministic model of computation. In membrane computing, basically, there are two types of framework: P systems with the membranes structure described by a tree, inspired from the cell, and tissue P systems with the membranes placed in the nodes of an arbitrary graph, inspired in the cellular tissue and in the neuron. The second one corresponds to the idea of forming a network of membranes linked in a specific manner and working together, [11]. In both types, the main idea is having multisets of objects placed in compartments evolving according to given rules in a synchronous non-deterministic maximally parallel manner.

It shall be focused here on tissue P systems. This variant has two biological inspirations (see [8]): intercellular communication and cooperation between cells/neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules (this way of communication for P systems was introduced in [12]). Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions.

From the definition of tissue P systems [7, 8], several research lines have been developed and new variants have arisen (see, for example, [1, 2, 6, 17]). One of the most interesting variants of tissue P systems was presented in [13]. In that paper,

the definition of tissue P systems is combined with the one of P systems with active membranes, yielding *tissue P systems with cell division*.

One of the main features of such tissue P systems with cell division is related to their computational efficiency. Several solutions for NP-complete problems have been published recently. In [13] a polynomial-time solution for the SAT problem is presented, and in [4, 5] lineal-time solutions for 3-coloring problem and Subset Sum problem, respectively, are presented. Polynomial-time solutions to NP-complete problems in Membrane Computing framework are done by trading time by space, in a theoretical way by constructing an exponential workspace in polynomial time. Although real implementations of this systems are no possible to be made, because it should be necessary to implement the maximal parallelism in some way, they are very interesting problems to treat in order to study some complexity aspects of Theoretical Computer Science, as for example $P \neq NP$ conjecture [14].

In recent years, this new field has been addressed in different ways: the study of computational properties such as computational power or complexity classes, definition of new variants of membrane systems closer to biological inspiration, working as a new framework of making biological simulations, etc. In P system web page [19] several software applications can be found. Most of them are thought in order to run an experiment which is built using some kind of membrane system as simulation framework; for example, in order to work with P system as a way of simulating biological system ([15]). In another hand, other group of software applications are thought as an easy way of visualize the computations of a P system ([9]) or spiking neural P system ([18]). Finally, in [3] a visual software application to understand the design of solutions for 3-coloring problem in the framework of recognizing tissue P system with cell division, is described.

In this paper a new visual tool called *Tissue Simulator* is presented. It is an application with a friendly graphical user interface that helps to work and understand tissue P systems with cell division.

The tool allows the user to write in an easy way the rules and the elements of a concrete tissue, run the execution of the system, and shows graphically a trace of the simulation with the rules applied in each computation step.

This paper is organized as follows: in Section 2 tissue P systems with cell division are defined and a solution of 3-coloring problem is described. Section 3 is devoted to describe *tissue simulator* application software is presented and some of the most important implementations aspects are commented. Finally, conclusions and future works are formulated.

2 Preliminaries

In this section, we briefly recall some concepts used in the paper.

An *alphabet*, Σ, is a non empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$. A *language* over Σ is a

subset from Σ^*.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \mid f(x) > 0\}$ and its *size* is defined as $\sum_{x \in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over $A = \{a_1, \ldots, a_n\}$, then it will be denoted as $m = a_1^f(a_1) \ldots a_n^f(a_n)$.

An undirected *graph* G is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges, each one of which is a (unordered) pair of (different) vertices. If $\{u, v\} \in E$, we say that u is *adjacent* to v (and also v is *adjacent* to u). The *degree* of $v \in V$ is the number of adjacent vertices to v.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For details, see [11].

3 Tissue P Systems with Cell Division

Gh. Păun et al. presented in [13] a new model of tissue P systems *with cell division*. The biological inspiration is clear: alive tissues are not *static* network of cells, since cells are duplicated via mitotic in a natural way.

The cells obtained by division have the same labels as the original cell and if a cell is divided, its interaction with other cells or with the environment is blocked during the mitotic process. In some sense, this means that while a cell is dividing it closes the communication channels with other cells and with the environment.

Formally, a *tissue P system with cell division* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_o),$$

where:

1. Γ is a finite *alphabet*, whose symbols will be called *objects*.
2. $w_1, \ldots, w_q$ are strings over Γ.
3. $\mathcal{E} \subseteq \Gamma$.
4. $\mathcal{R}$ is a finite set of rules of the following form:
   - **(a)** *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$.
   - **(b)** *Division rules*: $[a]_i \to [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a, b, c \in \Gamma$.
5. $i_o \in \{0, 1, 2, \ldots, q\}$.

A tissue P system with cell division of degree $q \geq 1$ can be seen as a set of q cells (each one consisting of an elementary membrane) labelled by $1, 2, \ldots, q$. We use 0 to refer to the label of the environment, and i_0 denotes the output region (which can be the region inside a membrane or the environment).

The communication rules determine a virtual graph, where the nodes are the cells and the edges indicated if it is possible for pairs of cells to communicate directly.

This is a dynamical graph, as new nodes can appear produced by the application of division rules.

The strings $w_1, \ldots, w_q$ describe the multisets of objects placed in the q cells of the system. We consider that $\mathcal{E} \subseteq \Gamma$ is the set of objects placed in the environment, each one of them in an arbitrary large amount of copies.

The communication rule $(i, u/v, j)$ can be applied over two cells i and j such that u is contained in cell i and v is contained in cell j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells.

The division rule $[a]_i \rightarrow [b]_i[c]_i$ can be applied over a cell i containing object a. The application of this rule divides this cell into two new cells with the same label. All the objects in the original cell are replicated and copied in each of the new cells, with the exception of the object a, which is replaced by the object b in the first new cell and by c in the second one.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way. In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we apply a maximal set of rules. This way of applying rules has only one restriction when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell do not evolve in that step.

4 Recognizing Tissue P Systems with Cell Division

NP-completeness has been usually studied in the framework of *decision problems*. Let us recall that a decision problem is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X.

In order to study the computing efficiency for solving **NP**-complete decision problems, a special class of tissue P systems with cell division is introduced in [13]: *recognizing tissue P systems*. The key idea of such recognizing system is the same one as from recognizing P systems with cell-like structure.

Recognizing cell-like P systems were introduced in [16] and they are the natural framework to study and solve decision problems within Membrane Computing, since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizing cell-like P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by encoding each instance as a multiset placed in an *input membrane*. The output of the computation (`yes` or `no`) is sent to the environment. In this way, cell-like P systems with input and external output are devices which can be seen as black boxes, in the sense that the user provides the data before the computation starts, and then waits *outside* the P system until it sends to the environment the output in the last step of the computation.

A recognizing tissue P system with cell division of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \Sigma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_{in}, i_o)$$

where:

- $(\Gamma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_o)$ is a tissue P system with cell division of degree $q \geq 1$ (as defined in the previous section).
- The working alphabet Γ has two distinguished objects `yes` and `no`, present in at least one copy in some initial multisets w_1, ..., w_q, but not present in $\mathcal{E}$.
- Σ is an (input) alphabet strictly contained in Γ.
- $i_{in} \in \{1, \ldots, q\}$ is the input cell.
- The output region i_o is the environment.
- All computations halt.
- If $\mathcal{C}$ is a computation of Π, then either the object `yes` or the object `no` (but not both) must have been released into the environment, and only in the last step of the computation.

The computations of the system Π with input $w \in \Sigma^*$ start from a configuration of the form $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \mathcal{E})$, that is, after adding the multiset w to the contents of the input cell i_{in}. We say that $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object `yes` (respectively, `no`) appears in the environment associated to the corresponding halting configuration of $\mathcal{C}$.

Definition 4.1 *A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ of recognizing tissue P systems with cell division if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions over I_X such that:*
  - *for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, then $\theta_X(u) = 1$;*

– *the family* $\mathbf{\Pi}$ *is* complete *with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

In the above definition we have imposed to every tissue P system $\Pi(n)$ to be *confluent*, in the following sense: every computation of such system with the *same* input multiset must always give the *same* answer.

It is denoted by $\mathbf{PMC}_{TD}$ the set of all decision problems which can be solved by means of recognizing tissue P systems with cell division in polynomial time.

5 A Solution for the 3–Coloring Problem

A k–coloring ($k \geq 1$) of an undirected graph $G = (V, E)$ is a function $f : V \to \{1, \ldots, k\}$, where the numbers are interpreted as colors. We say that G is k–colorable if there exists a k–coloring, f, such that $f(u) \neq f(v)$ for every edge $\{u, v\} \in E$ (such a k–coloring f is said to be *valid*).

The 3–coloring problem is the following: *given an undirected graph* G*, decide whether or not* G *is* 3*-colorable*; that is, if there exists a valid 3–coloring of G.

In [4] it is proved that the 3–coloring problem can be solved in linear time by a family of recognizing tissue P systems with cell division.

For each $n, m \in \mathbb{N}$, we shall consider the system

$$\Pi(g(n,m)) = (\Gamma(g(n,m)), \Sigma(n), w_1, w_2(n), \mathcal{R}(g(n,m)), \mathcal{E}(g(n,m)), i_{in}, i_o)$$

being $g(n,m) = ((n+m)(n+m+1)/2) + n$, and where:

- $\Gamma(g(n,m))$ is the set
  $\{A_i, R_i, T_i, B_i, G_i, \overline{R}_i, \overline{B}_i, \overline{G}_i : 1 \leq i \leq n\} \cup$
  $\{a_i : 1 \leq i \leq 2n + m + \lceil log\ m \rceil + 11\} \cup \{c_i : 1 \leq i \leq 2n+1\} \cup$
  $\{d_i : 1 \leq i \leq \lceil log\ m \rceil + 1\} \cup \{f_i : 2 \leq i \leq m + \lceil log\ m \rceil + 6\} \cup$
  $\{A_{ij}, P_{ij}, \overline{P}_{ij}, R_{ij}, B_{ij}, G_{ij} : 1 \leq i < j \leq n\} \cup \{b, D, \overline{D}, e, T, S, N, \flat, \texttt{yes}, \texttt{no}\}$
- $\Sigma(n) = \{A_{ij} : 1 \leq i < j \leq n\}$
- $w_1 = \{\{a_1, b, c_1, \texttt{yes}, \texttt{no}\}\}$
- $w_2(n) = \{\{D, A_1, \ldots, A_n\}\}$
- $\mathcal{R}(g(n,m))$ is the set of rules:

  1. **Division rules:**
     $r_{1,i} \equiv [A_i]_2 \to [R_i]_2[T_i]_2$ for $i = 1, \ldots, n$
     $r_{2,i} \equiv [T_i]_2 \to [B_i]_2[G_i]_2$ for $i = 1, \ldots, n$
  2. **Communication rules:**
     $r_{3,i} \equiv (1, a_i/a_{i+1}, 0)$ for $i = 1, \ldots, 2n + m + \lceil log\ m \rceil + 10$
     $r_{4,i} \equiv (1, c_i/c_{i+1}^2, 0)$ for $i = 1, \ldots, 2n$
     $r_5 \equiv (1, c_{2n+1}/D, 2)$

$r_6 \equiv (2, c_{2n+1}/d_1\overline{D}, 0)$
$r_{7,i} \equiv (2, d_i/d_{i+1}^2, 0)$ for $i = 1, \ldots, \lceil log\ m \rceil$
$r_8 \equiv (2, \overline{D}/e\ f_2, 0)$
$r_{9,i} \equiv (2, f_i/f_{i+1}, 0)$ for $i = 2, \ldots, m + \lceil log\ m \rceil + 5$
$r_{10,ij} \equiv (2, d_{\lceil log\ m \rceil + 1} A_{ij}/P_{ij}, 0)$ for $1 \leq i < j \leq n$
$r_{11,ij} \equiv (2, P_{ij}/R_{ij}\overline{P}_{ij}, 0)$ for $1 \leq i < j \leq n$
$r_{12,ij} \equiv (2, \overline{P}_{ij}/B_{ij}G_{ij}, 0)$ for $1 \leq i < j \leq n$
$r_{13,ij} \equiv (2, R_iR_{ij}/R_i\overline{R}_j, 0)$ for $1 \leq i < j \leq n$
$r_{14,ij} \equiv (2, B_iB_{ij}/B_i\overline{B}_j, 0)$ for $1 \leq i < j \leq n$
$r_{15,ij} \equiv (2, G_iG_{ij}/G_i\overline{G}_j, 0)$ for $1 \leq i < j \leq n$
$r_{16,j} \equiv (2, \overline{R}_jR_j/\flat, 0)$ for $1 \leq j \leq n$
$r_{17,j} \equiv (2, \overline{B}_jB_j/\flat, 0)$ for $1 \leq j \leq n$
$r_{18,j} \equiv (2, \overline{G}_jG_j/\flat, 0)$ for $1 \leq j \leq n$
$r_{19} \equiv (2, e\flat/\lambda, 0)$
$r_{20} \equiv (2, e\ f_{m+\lceil log\ m \rceil + 6}/T, 0)$
$r_{21} \equiv (2, T/\lambda, 1)$
$r_{22} \equiv (1, b\,T/S, 0)$
$r_{23} \equiv (1, S\ \texttt{yes}/\lambda, 0)$
$r_{24} \equiv (1, b\ a_{2n+m+\lceil log\ m \rceil + 11}/N, 0)$
$r_{25} \equiv (1, N\ \texttt{no}/\lambda, 0)$

- $\mathcal{E}(g(n,m)) = \Gamma(g(n,m)) - \{\texttt{yes}, \texttt{no}\}$
- $i_{in} = 2$ is the *input cell*.
- $i_o = 0$ is the *output region*.

Given an undirected graph $G = (V, E)$ with $V = \{A_1, \ldots A_n\}$ and $|E| = m$, we consider $s(u) = g(n,m)$ and $cod(u) = \{A_{ij} : \{A_i, A_j\} \in E \wedge 1 \leq i < j \leq n\}$. Then, the recognizing tissue P system $\Pi(s(u) = \Pi(g(n,m))$ with input $cod(u)$ processes the instance G via a brute force algorithm, which consists in the following stages:

- *Generation Stage*: The initial cell labelled by 2 is divided into two new cells; and the divisions are iterated until a cell has been produced for each possible candidate solution. Simultaneously, in the cell labelled by 1 there is a counter that will determine the moment in which the checking stage starts.
- *Pre–checking Stage*: After obtaining all possible 3–colorings encoded in cells labelled by 2, this stage provides objects R_{ij}, G_{ij}, B_{ij} in such cells, for every edge A_{ij}.
- *Checking Stage*: Objects R_{ij}, G_{ij}, B_{ij} will be used in cells labelled by 2 to check if there exists a pair of adjacent vertices with the same color in the corresponding candidate solution.
- *Output Stage*: The system sends to the environment the right answer according to the results of the previous stage.

6 A Look Inside Tissue Simulator

The application has been developed using Java and C♯, which are portable and powerful object-oriented languages. Java has been used to parse the data introduced using a grammar generated with ANTLR (one of the most known tools for this purpose, [20]). C♯ has been used to develop the kernel of the application and the graphical interface. The interconnection between both languages is transparent to the user, that just have to move between windows by clicking on buttons. The data are stored in a XML format after serializing the objects that contain the information

The rules of the tissue P system can be rewritten in a way that is very similar to the one used in papers from computational theory. Because computers can not understand directly those kinds of expressions, it has been necessary to develop a *code generator* that writes and compiles during runtime the source code in C♯ simulating the behavior of the rules of the system.

The software follows the Model-View-Controller (MVC) [21], an architecture model of software development used in interactive systems. Three different parts or layers can be distinguished: data handling layer, algorithmic or business logic layer, and user interface or graphical layer. With this, it is easier to do maintenance of the code.

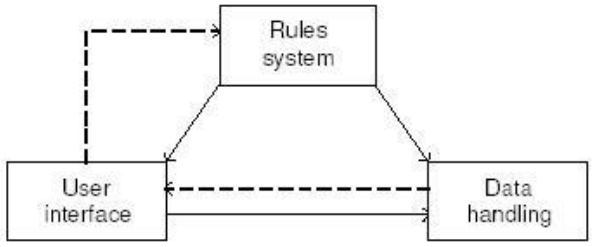


Figure 1: Model-View-Controller architecture software

Algorithmic layer implements a recognizing tissue P system with cell division. All rules are applied in a non deterministic and maximal parallel way. The design of the tissue P system machine for solving the problem is non deterministic until $2n$ step. Every computation ways reach the same configuration in $2n$ step. From this moment, the machine is deterministic. By this way, we have chosen only a computation way in order to implement the tissue P system in the software. The computation selected to make the software simulation is the one determined by the lexicographical order of the rule. Other solutions can be easily added. Graphical layer allows the user a visual and friendly interaction with the application. At the end of the simulation, the result of the problem and the trace of the execution is displayed, and for each step the rules applied and the configurations of the tissue are shown.

7 A User Overview of the Application

This software tool allows to follow step by step the performance of tissue P system with cell division solutions of several problems. The pictures presented in this paper have been captured when a solution of the 3-coloring problem (whose files of codification can be found in the folder of application) is provided to the tool.

7.1 Running a simulation

When it is wanted to introduce a concrete tissue P system it has to be selected in tissue menu. Several options are presented and it must be selected new system button. The corresponding window can be depicted in Figure 2.

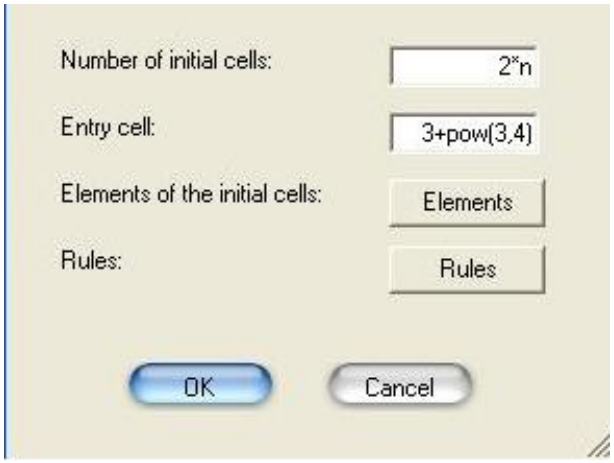


Figure 2: Definition of system window

Then the rules (Figure 3) and the initial multisets can be incorporated in the window.

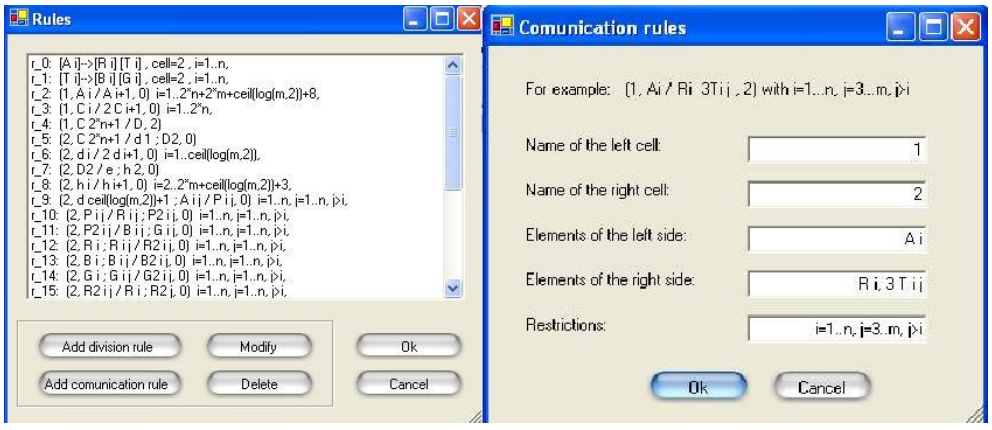


Figure 3: The set of rules for a solution of 3-coloring problem

Next, a concrete input multiset, that depends on the specific instance of the

3-coloring problem, is supplied. For this, the *new instance* option in tissue menu is selected.

If the user needs run a simulation, it is just needed to select the option *Run* in tissue menu. Then the system ask the path of two .xml files, one for the model of the system, and another one for a concrete instance of the problem. After that, it calculates all the steps of the computation, shows the answer, and open the trace window of the system, as can be depicted in Figure 4.

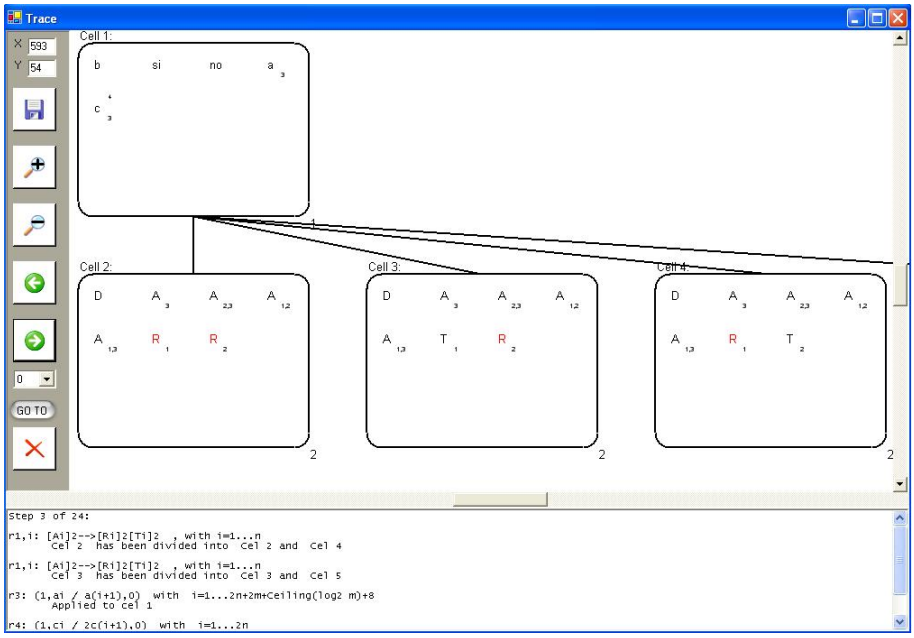


Figure 4: Trace window for a solution of 3-coloring problem

For each step, a situation can be seen in a graphical way, and at the bottom of the screen appear the different rules applied in that step. The picture can be easily handled at any moment, and can be saved in different images format. Moving between the steps can be done in two ways: first, by pressing the buttons with arrows to move from the i-th step to the $(i-1)$-th or the $(i+1)$-th one. Second, choosing the "GO TO" option to go directly to one step of the system.

Software, manuals, examples, a mailing list, useful links, and other resources of *tissue simulator* can be found at: `http://www.tissuesimulator.es.kz`.

8 Future Work

In this paper, a simulator of recognizing tissue P systems has been presented. The simulator has been developed using Java and C♯, which are portable and powerful object-oriented languages. Future works will be focused on different improvements of this software application like introducing a semantic parser, contemplating others kind of rules, such as membrane creation, etc.

One interesting future tasks would be to expand the tool so it can works with others systems like spiking neural P systems.

Acknowledgement

The authors wish to acknowledge the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andaluca.

References

[1] A. Alhazov, R. Freund, M. Oswald. Tissue P Systems with Antiport Rules ans Small Numbers of Symbols and Cells. In volume 3572 of *Lecture Notes in Computer Science*, pages 100–111. 2005.

[2] F. Bernardini, M. Gheorghe. Cell Communication in Tissue P Systems and Cell Division in Population P Systems. *Soft Computing*, 9(9):640–649, 2005.

[3] R. Borrego–Ropero, D. Díaz–Pernil, J.A. Nepomuceno–Chamorro. VisualTissue: a friendly tool to study tissue P systems solutions for graph problems. In M.A. Gutiérrez Naranjo, Gh. Paun, A. Romero–Jiménez, A. Riscos–Núñez, editors, *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, pages 87–96. Fénix Editora, 2007.

[4] D. Díaz–Pernil, M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. A uniform family of tissue P system with cell division solving 3-COL in a linear time. *Theoretical Computer Science*, in press.

[5] D. Díaz–Pernil, M.A. Gutiérrez–Naranjo, M.J. Pérez Jiménez M.J., A. Riscos–Núñez. A Linear Solution for Subset Sum Problem with Tissue P Systems with Cell Division. In M.A. Gutiérrez Naranjo, Gh. Paun, A. Romero–Jiménez, A. Riscos–Núñez, editors, *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, pages 113–130. Fénix Editora, 2007.

[6] R. Freund, Gh. Păun, M.J. Pérez-Jiménez. Tissue P Systems with channel states. *Theoretical Computer Science*, 330:101–116, 2005.

[7] C. Martín–Vide, J. Pazos, Gh. Păun, A. Rodríguez–Patón. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. In volume 2387 of *Lecture Notes in Computer Science*, pages 290–299, 2002.

[8] C. Martín–Vide, J. Pazos, Gh. Păun, A. Rodríguez–Patón. Tissue P systems. *Theoretical Computer Science*, 296:295–326, 2003.

[9] I.A. Nepomuceno–Chamorro. A Java Simulator for Membrane Computing. *Journal of Universal Computer Science*, 10(5):620–629, 2004.

[10] Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[11] Gh. Păun. *Membrane Computing. An Introduction.* Springer–Verlag, Berlin, 2002.

[12] A. Păun, Gh. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20(3):295–305, 2002.

[13] Gh. Păun, M.J. Pérez–Jiménez, A. Riscos-Núñez, A. Tissue P System with cell division. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho–Caparrini, editors, *Second Brainstorming Week on Membrane Computing*, pages 380–386. Sevilla, Report RGNC 01/2004, 2004.

[14] M.J. Pérez–Jiménez. An approach to computational complexity in Membrane Computing. In volume 3365 of *Lecture Notes in Computer Science*, pages 85–109. 2005.

[15] M.J. Pérez–Jiménez, F.J. Romero. P systems, a new computational modelling tool for Systems Biology. In *Transactions on Computational Systems Biology VI.*, volume 4220 of *Lecture Notes in Bioinformatics*, pages 176–197, 2006.

[16] M.J. Pérez–Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, F. A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke and Gy. Vaszil, editors, *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, pages 284–294. 2003.

[17] V.J. Prakash. On the Power of Tissue P Systems Working in the Maximal-One Mode. In A. Alhazov, C. Martín-Vide and Gh. Păun, editors, *Preproceedings of the Workshop on Membrane Computing*, pages 356–364. Report RGML 28/03, Tarragona, 2003.

[18] D. Ramírez–Martínez,M.A. Gutiérrez–Naranjo. A Software Tool for Dealing with Spiking Neural P Systems. In M.A. Gutiérrez Naranjo, Gh. Paun, A. Romero–Jiménez, A. Riscos–Núñez, editors, *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, pages 299–313. Fénix Editora, 2007.

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

[20] web site `http://www.antlr.org`

[21] web site `http://en.wikipedia.org/wiki/Model-view-controller`

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 35 - 51, 2007.

Colonies of Synchronizing Agents: An Abstract Model of Intracellular and Intercellular Processes

Matteo Cavaliere, Radu Mardare, and Sean Sedwards

Microsoft Research – University of Trento
Centre for Computational & Systems Biology
{cavaliere, mardare, sedwards}@cosbi.eu

Abstract

We present a modelling framework and computational paradigm called Colonies of Synchronizing Agents (CSAs), which abstracts intracellular and intercellular mechanisms of biological tissues. The model is based on a multiset of agents (cells) in a common environment. Each agent has a local contents, stored in the form of a multiset of atomic objects, updated by multiset rewriting rules which may act on individual agents (intracellular action) or synchronize the contents of pairs of agents (intercellular action). Using tools from formal language and temporal logic we investigate dynamic properties of CSAs, including robustness and safety of synchronization. We also identify classes of CSAs where such dynamic properties can be algorithmically decided.

1 Motivations

Inspired by biological tissues and populations of cells, we present and investigate an abstract distributed model of computation which we call Colonies of Synchronizing Agents (in short, CSAs). Our intention is to create a framework to model, analyse and simulate complex biological systems in the context of formal language theory and multiset rewriting.

The model is based on a population of *agents* (e.g., corresponding to *cells* or *molecules*) in a common environment, able to modify their contents and to synchronize with other agents in the same environment. Each agent has a contents represented by a multiset of atomic objects (e.g., corresponding to chemical compounds or the characteristics of individual molecules) with some of the objects classified as terminals (e.g., corresponding to properties or chemicals visible to an external observer). An agent's contents may be modified independently of other agents by means of multiset rewriting rules (called *internal rules*)[1] which can mimic chemistry or other types of *intracellular mechanisms*. Moreover, the agents can influence each other by synchronously changing their contents using pairwise *synchronization rules*.

[1] In [5] internal rules are called evolution rules, adopting a standard terminology from the P systems area. We prefer here a more general term.

This models, in a deliberately abstract way, the various signalling mechanisms and *intercellular mechanisms* present in biological systems. The rules are global, so all agents obey the same rules: the only feature which may distinguish the agents is their contents. Evolutions of CSAs are defined as sequences of transitions obtained by applying the rules to the agents. These transitions thus mark the passage of the system from one configuration to another.

In this paper we search for classes of CSAs where relevant *dynamic properties* can be algorithmically checked. We interpret CSAs as computational devices and can thus study CSAs by applying tools from classical fields of computer science, such as formal language, automata theory and temporal logic. For this reason we define as computations of CSAs the evolutions that reach halting configurations, i.e. configurations where the contents of the agents can no longer be changed because no rules may be applied. This situation can be interpreted as a particular kind of steady state of the system. We are interested in the configuration of the colony when a halting condition is reached and we may take the precise contents of the agents as the output (the result) produced by the CSA. Alternatively, we can use the magnitude of the agents (the total amount of contents irrespective of composition) in the halting configuration as the result produced by a CSA.

We can then investigate the *robustness of CSAs* by considering the ability of a CSA to generate a particular *core result* despite the failure (i.e., removal) of some of the agents or rules. The core result can be seen as a specific configuration in which the colony must be when the system halts. We show that for an arbitrary CSA, robustness cannot be algorithmically decided when the core result is represented by specific contents of agents, while it *can* be algorithmically decided in an efficient way when the core result is represented by agents' magnitudes.

In Section 4 we are interested in dynamic properties concerning the application of the rules. To check these properties we propose a decidable temporal logic. We show that the proposed logic can be used to specify and check whether or not, during any evolution of a CSA, an agent can apply a synchronization whenever it needs (if it can we say that the agent is safe on synchronization). We conclude the present section by comparing our model with other models based on abstractions of cell tissues which use rewriting and multisets.

The introduced model of Colonies of Synchronizing Agents has similarities and significant differences with other models inspired by cell tissues investigated, for instance, in the area of membrane computing (a.k.a. P systems, [15]). Specifically, it can be considered a generalization of P colonies [11], which is also based on interacting agents but has agents with limited contents (two objects) which change by means of restricted rewriting rules. Moreover, in P colonies no direct communication between agents is allowed.

Our model also has similarities with population P systems [3], which is a class of tissue P systems [13] where links may exist between agents and these can be modified by means of a set of bond making rules.

The main differences with population P systems is that in our case agents do not have types; rules are global and only the agents' contents differentiate them. This latter characteristic makes CSAs similar to the model of self-assembly of graphs presented in [2], however in that case; (i) a graph is constructed from an initial

seed using multiset-based aggregation rules to enlarge the structure, (*ii*) there is no internal rewriting of the agent's contents and (*iii*) there is no synchronization between the agents.

Another computational formalism widely used to simulate and model biological tissues is cellular automata (CAs, e.g., see [19]). In particular, CAs have been used to model the immune systems (e.g., [14]). In CAs, cells exist on a regular grid, where each cell has a finite number of possible states and where cells react to or with a defined neighbourhood. In our model, because of the multiset-based contents and because of the arbitrary multiset rewriting rules, the possible different states of a cell may be infinite. Although the initial definition of CSAs does not include an explicit description of space, the extensions we propose include agents located at arbitrary positions and with the potential to interact with any other agent in the colony.

A specific limitation of cellular automata that use synchronous update is that many such models are computational complete (i.e., equivalent to Turing machines [19]), even when employing *simple* rules (e.g., rule 110, [19]). This makes it impossible to algorithmically analyse such systems. Precisely, non-trivial problems are undecidable for Turing machines.

2 Formal Language Preliminaries

This Section is a brief introduction to some basic notions of formal language theory needed in the paper. Further information regarding formal language and automata theory is available from the many monographs in this area, starting with [10, 4] and ending with the handbook [17].

Given the set A we denote by $|A|$ its cardinality and by $\emptyset$ the empty set. We denote by $\mathbb{N}$ the set of natural numbers.

An *alphabet* V is a finite set of symbols. By V^* we denote the set of all strings over V. By V^+ we denote the set of all strings over V excluding the empty string. The empty string is denoted by λ. The *length* of a string v is denoted by $|v|$. The concatenation of two strings $u, v \in V^*$ is written uv. The number of occurrences of the symbol a in the string w is denoted by $|w|_a$.

Each subset of V^* is called a *language*.

The boolean operations (with languages) of union and intersection are denoted $\cup$ and $\cap$, respectively. Concatenation of the languages L_1, L_2 is $L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$.

A generative grammar is a finite device generating in a well-specified sense the strings of a language. Chomsky grammars are particular cases of rewriting systems where the operation used in processing the strings is the rewriting (replacement of a substring of the processed string by another substring). A (Chomsky) grammar is a quadruple $G = (N, T, S, P)$ where N and T are disjoint alphabets, N being a set of non-terminals and T a set of terminals, S is the axiom and P is a finite set of productions (rewriting rules). A production is usually written in the form $r : u \to v$ with $u \in (N \cup T)^*$ with u containing at least a non-terminal (so, it cannot be the empty string).

For $x, y \in (N \cup T)^*$ we write $x \Longrightarrow y$ iff $x = x_1 u x_2, y = x_1 v x_2$ for some $x_1, x_2 \in (N \cup T)^*$ and $u \to v \in P$. One says that x directly derives y. The language generated by G denoted by $L(G)$ is defined by $L(G) = \{x \in T^* \mid S \Longrightarrow^* x\}$, where $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$. So the language $L(G)$ consists of all terminal strings that can be obtained starting from S by applying iteratively the productions in P.

A grammar is called *regular* if each production is of the form $a \to v$ with $a \in N$ and $v \in T \cup TN \cup \{\lambda\}$. A grammar is called *context-free* if each production is of the form $a \to v$ with $a \in N$.

Languages generated by context-free and regular grammars are called context-free and regular languages, respectively. We denote by CF and REG the families of context-free and regular languages, respectively. Regular languages are those accepted by finite state automata.

In general, when we want to specify a terminal alphabet we add a subscript to the name of the family; e.g., REG_A is the family of all regular languages over the alphabet A.

A matrix grammar without appearance checking is a devices with matrices of context-free productions and where productions are applied according to the order given in the chosen matrix (for details see [6]).

Formally, a *matrix grammar without appearance checking* (in short, without a.c.) is a construct $G = (N, T, S, M)$, where N and T are disjoint alphabets of non-terminal and terminal symbols, $S \in N$ is the axiom, M is a finite set of matrices which are sequences of context-free productions of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$ (with $A_i \in N, x_i \in (N \cup T)^*$ in all cases).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots A_n \to x_n)$ in M and strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1, z = w_n + 1$ and, for all $1 \leq i \leq n$, $w_i = w_i' A_i w_i''$, $w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$. The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$. Then the language generated by G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$.

In other words, the language $L(G)$ is composed of all the strings of terminal symbols that can be obtained starting from S and applying iteratively the matrices in M.

For a language $L \subseteq V^*$, the set $length(L) = \{|x| \ |x \in L\}\}$ is called the *length set* of L, denoted by NL.

If FL is an arbitrary family of languages then we denote by NFL the family of length sets of languages in FL (i.e., it is a family of sets of natural numbers). For instance, $NREG$ is the family of length sets of regular languages.

The *Parikh vector* associated with a string $x \in V^*$ with respect to the alphabet $V = \{a_1, a_2, \ldots, a_n\}$ is $Ps_V(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n})$. For $L \subseteq V^*$ we define $Ps_V(L) = \{Ps_V(x) | x \in L\}$. This is called the *Parikh image* of the language L. The null vector is denoted by $\overline{0}$.

If FL is an arbitrary family of languages then we denote by $PsFL$ the family of Parikh images of languages in FL (i.e., it is a family of sets of vectors of natural numbers).

For instance, $PsREG$ is the family of Parikh images of regular languages in REG.

For instance, $V = \{a, b, c\}$ is an alphabet, $x = aaabbbcaa = a^3b^3ca^2$ is a string over V, $L = \{a^n b^n \mid n \geq 1\}$ is a language over V. We have $|x| = 9$, $|x|_a = 5$, $length(L) = \{2n \mid n \geq 1\}$. The Parikh vector of x with respect to V is $Ps_V(x) = (5, 3, 1)$ and for the language L we have $Ps_V(L) = \{(n, n, 0) \mid n \geq 1\}$.

A *multiset* is a set where each element may have a multiplicity. Formally, a multiset over a set V is a map $M : V \to \mathbb{N}$, where $M(a)$ denotes the multiplicity (i.e., number of *occurrences*) of the symbol $a \in V$ in the multiset M. Note that the set V can be infinite.

For instance $M = \{a, b, b, b\}$, also written as $\{(a, 1), (b, 3)\}$, is a multiset with $M(a) = 1$ and $M(b) = 3$.

For multisets M and M' over V, we say that M is *included in* M' ($M \subseteq M'$) if $M(a) \leq M'(a)$ for all $a \in V$. Every multiset includes the *empty multiset*, defined as M where $M(a) = 0$ for all $a \in V$.

The *sum* of multisets M and M' over V is written as the multiset $(M + M')$, defined by $(M + M')(a) = M(a) + M'(a)$ for all $a \in V$. The *difference* between M and M' is written as $(M - M')$ and defined by $(M - M')(a) = max\{0, M(a) - M'(a)\}$ for all $a \in V$. We also say that $(M + M')$ is obtained by *adding* M to M' (or vice versa) while $(M - M')$ is obtained by *removing* M' from M.

For example, given the multisets $M = \{a, b, b, b\}$ and $M' = \{b, b\}$, we can say that M' is included in M, that $(M + M') = \{a, b, b, b, b, b\}$ and that $(M - M') = \{a, b\}$.

The *support* of a multiset M is defined as the set $supp(M) = \{a \in V | M(a) > 0\}$. A multiset with finite support is usually presented as a set of pairs $(x, M(x))$, for $x \in supp(M)$.

The *cardinality* of a multiset M is denoted by $card(M)$ and it indicates the number of objects in the multiset. It is defined in the following way. $card(M)$ is infinite if M has infinite support. If M has finite support then $card(M) = \sum_{a_i \in supp(M)} M(a_i)$, i.e., all the occurrences of the elements in the support are counted.

We denote by $\mathbb{M}(V)$ the set of all possible multisets over V and by $\mathbb{M}_k(V)$ the set of all multisets over V having cardinality k.

For the case that the alphabet V is finite we can use a compact string notation to denote multisets: if $M = \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_n, M(a_n))\}$ then the string $w = a_1^{M(a_1)} a_2^{M(a_2)} \cdots a_n^{M(a_n)}$ (and all its permutations) precisely identify the symbols in M and their multiplicities. Hence, given a string $w \in V^*$, we can say that it identifies the multiset $\{(a, |w|_a) \mid a \in V\}$. For instance, the string bab represents the multiset $\{b, a, b\} = \{(a, 1), (b, 2)\}$ which has cardinality 3. The empty multiset is represented by the empty string, λ.

3 Colonies of Synchronizing Agents

In this section we formalize the notions of colonies discussed in the Introduction. A *Colony of Synchronizing Agents* (a CSA) of degree m is a construct $\Pi = (A, T, C, R)$.
• A is a finite alphabet of symbols (its elements are called *objects*). $T \subseteq A$ is the set of *terminal objects*.
• An *agent* over A is a multiset over the alphabet A (an agent can be represented by a string $w \in A^*$, since A is finite). C is the *initial configuration of* Π and it is a

multiset of agents, with $card(C) = m$. [2]
• R is a finite set of *rules* over A. We have *internal rules* of type $u \to v$, with $u \in A^+$ and $v \in A^*$, and *synchronization rules* of the type $\langle u, v\rangle \to \langle u', v'\rangle$ with $uv \in A^+$ and $u', v' \in A^*$.

An occurrence γ of an internal rule $r : u \to v$ can be applied to an agent w by taking a multiset u from w (hence, $u \subseteq w$) and *assigning* it to γ (i.e., assigning the occurrences of the objects in u to γ). The application of an occurrence of rule r to the agent w consists of removing from w the multiset u and then adding the multiset v to the resulting multiset.

An occurrence γ of a synchronization rule $r : \langle u, v\rangle \to \langle u', v'\rangle$ can be applied to the pair of agents w and w' by: (i) taking from w a multiset u (hence, $u \subseteq w$) and *assigning* it to γ; (ii) taking from w' a multiset v (hence, $v \subseteq w'$) and *assigning* it to γ. The application of an occurrence of rule r to the agents w and w' consists of: (i) removing the multiset u from w and then adding the multiset u' to the resulting multiset; (ii) removing the multiset v from w' and then adding the multiset v' to the resulting multiset.

We assume the existence of a *global clock* which marks the passage of units of time for all agents present in the colony.

A *configuration* of a CSA, Π, consists of the agents present in the colony at a given time. We denote by $\mathbb{C}(\Pi)$ the set of *all possible configurations* of Π. Therefore, using the notation introduced in Section 1, $\mathbb{C}(\Pi)$ is exactly $\mathbb{M}_m(H)$ with $H = \mathbb{M}(A)$.

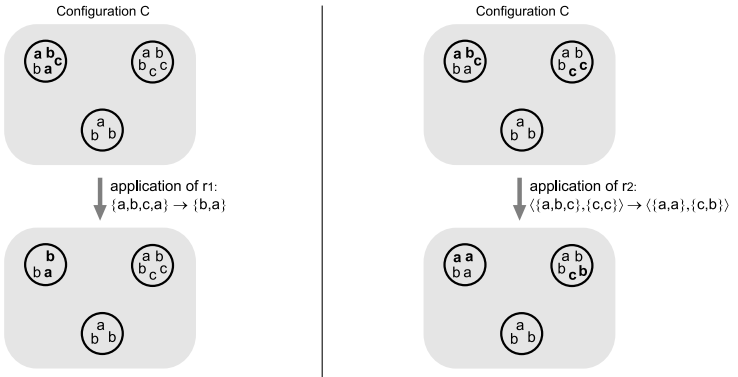
A single *asynchronous transition* (in short, *asyn*-transition) [3] of Π from an arbitrary configuration c of Π to the next one lasts exactly one time unit and is obtained by applying the rules in the set R to the agents present in c in an *asynchronous* way. This means that, for each agent w and each pair of agents w' and w'' present in c, the occurrences of the objects of w, w' and w'' are *either* assigned to occurrences of the rules, with the occurrences of the objects and the occurrences of the rules chosen in a non-deterministic way, *or* left unassigned. A single occurrence of an object may only be assigned to a single occurrence of a rule. In other words, in an *asyn*-transition any number of occurrences of rules (zero, one, or more) can be applied to the agents in the configuration c.

A sequence (possibly infinite) $\langle C_0, C_1, \cdots, C_i, C_{i+1}, \cdots\rangle$ of configurations of Π, where C_{i+1} is obtained from C_i, $i \geq 0$, by an *asyn*-transition is called an *asyn-evolution* of Π. An *asyn*-evolution of Π is said to be *halting* if it halts, that is if it is finite and the last configuration of the sequence is a *halting configuration*, i.e., a configuration containing only agents for which no occurrences of rules from R can be applied.

An *asyn*-evolution of Π that is halting and that *starts with the initial configuration* of Π is called an *asyn-computation* of Π. The *result/output* of an asyn-computation is the *set of vectors of natural numbers*, one vector for each agent w present in the halting configuration with the vector describing the multiplicities of terminal objects present in w. More formally, the result of an asyn-computation

[2] Formally, C is a multiset of degree m over the set of all possible agents over A. Hence, $C \in \mathbb{M}_m(\mathbb{M}(A))$.

[3] We specify *asyn*-transitions to distinguish them from the synchronous maximal parallel transitions often adopted in models coming from P systems and cellular automata.



(*a*) Internal rule r_1 applied to C | (*b*) Synchronization rule r_2 applied to C

Figure 1: Alternative application of rules r_1 and r_2 to configuration C from Example 1.

which stops in the halting configuration C_h is the set of vectors of natural numbers $\{Ps_T(w) \mid w \text{ is an agent present in } C_h\}$.

Because of the non-determinism in applying the rules, several possible asyn-computations of Π may exist. Taking the union of all the results for all possible asyn-computations of Π, we get the *set of vectors generated* by Π, denoted by $Ps_T^{asyn}(\Pi)$.

We may also consider the total number of objects comprising the agent (the agent's *magnitude*), without considering the internal composition. In this case the *result* of an asyn-computation is the *set of natural numbers*, one number for each agent w present in the halting configuration and each number being the length of w. More formally, in this case the result of an asyn-computation that stops in the halting configuration C_h is then the set of numbers $\{|w| \mid w \text{ is an agent present in } C_h\}$. Again, taking the union of all the results for all possible asyn-computations of Π, we get the *set of numbers generated* by Π, denoted by $N^{asyn}(\Pi)$.

In what follows we indicate by C_Π the initial configuration of Π.

Example 1 A CSA with degree 3 is defined by the following.

$\Pi = (A, T, C, R)$ with $A = \{a, b, c\}$, $T = \{a\}$, $C = \{(abcba, 1), (abbcc, 1), (bab, 1)\}$ $= \{abcba, abbcc, bab\}$.

The rules $R = \{r_1 : abca \to ba,\ r_2 : \langle abc, cc\rangle \to \langle aa, cb\rangle\}$.

The application of an occurrence of internal rule r_1 to the agent *abcba* in the configuration C is shown diagrammatically in Figure 1(a).

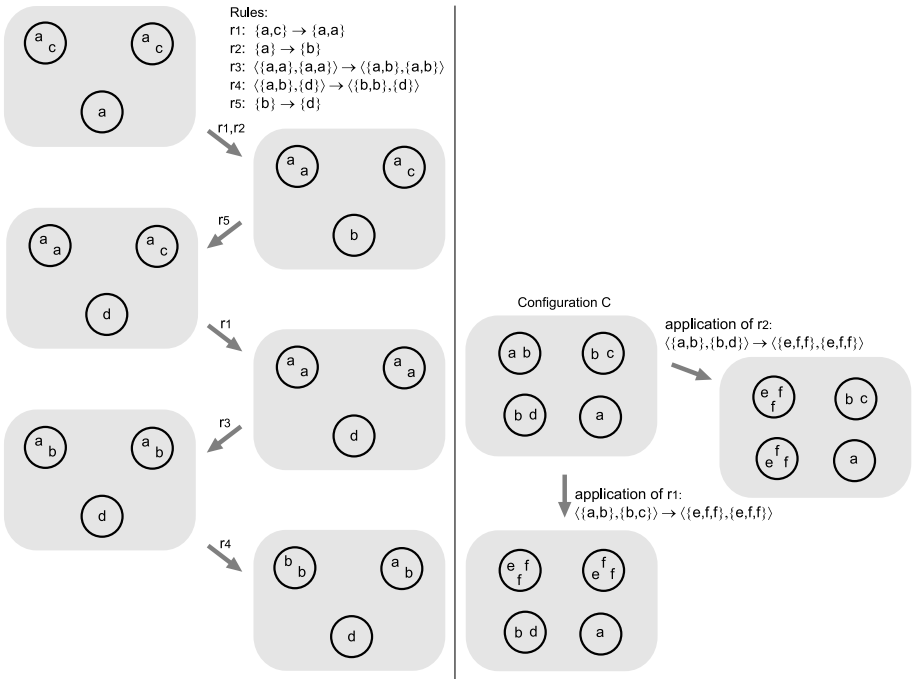
The application of an occurrence of the synchronization rule r_2 to the pair of agents *abcba* and *abbcc* in the configuration C is shown diagrammatically in Figure 1(b).

A more complex example of part of an asynchronous evolution is presented in Figure 2(a): $\Pi' = (A', T', C', R')$ with the initial configuration $C' = \{(ac, 2),\ (a, 1)\}$ and rules $R' = \{ac \to aa,\ a \to b,\ \langle aa, aa\rangle \to \langle ab, ab\rangle,\ \langle ab, d\rangle \to \langle bb, d\rangle,\ b \to d\}$.

In the next Example we show how the output/result produced by a CSA is obtained.

Example 2 Consider a CSA $\Pi = (A, T, C, R)$ with $A = \{a, b, c, d, e, f\}$, $T = \{e, f\}$, $C = \{(ab, 1), (bc, 1), (bd, 1), (a, 1)\}$.

The rules in R are $\{r_1 : \langle ab, bc\rangle \to \langle eff, eff\rangle, r_2 : \langle ab, bd\rangle \to \langle eff, eff\rangle\}$.



(*a*) Asynchronous evolutions of Π' of Example 1

(*b*) The two possible asynchronous computations of Π of Example 2

Figure 2: Asynchronous evolutions and computations.

There are *only two possible asynchronous computations* of Π and these are represented diagrammatically in Figure 2(b).

We have that $Ps_T^{asyn}(\Pi) = \{(1, 2), \overline{0}\}$.

In fact, we have two possible halting configurations (for the two computations). In the first halting configuration we have the agent (in two copies) eff whose associated Parikh vector (with respect to T) is $(1, 2)$ and the agents bd and a, whose associated Parikh vectors (with respect to T) are null vectors $\overline{0}$ (these agents do not contain any terminal object from T). Then the result of this computation is the set of vectors $\{(1, 2)\} \cup \{(1, 2)\} \cup \{\overline{0}\} \cup \{\overline{0}\} = \{(1, 2), \overline{0}\}$ with each vector describing the multiplicities of the terminal objects in the agents in the halting configuration.

In the second halting configuration we have the agent (in two copies) eff whose associated Parikh vector (with respect to T) is $(1, 2)$ and the agents bc and a, whose associated Parikh vectors (with respect to T) are null vectors. Then, also in this case, the result of the computation is the set of vectors $\{(1, 2), \overline{0}\}$

Taking the union of the results for the (two) possible computations we get $Ps_T^{asyn}(\Pi) = \{(1, 2), \overline{0}\} \cup \{(1, 2), \overline{0}\} = \{(1, 2), \overline{0}\}$.

We can also collect the result in terms of magnitude (size) of the agents present in the halting configurations, thus collecting $N^{asyn}(\Pi)$. In this case we obtain

$N^{asyn}(\Pi) = \{3, 2, 1\}$. In fact, in the two halting configurations we have agents of size $3, 2$ and 1 (counting their objects). Then for both computations the result is the set of numbers $\{3, 3, 2, 1\} = \{3, 2, 1\}$ with each number being the magnitude of an agent in the halting configuration.

Taking the union of the results for the (two) possible computations we obtain $N^{asyn}(\Pi) = \{3, 2, 1\} \cup \{3, 2, 1\} = \{3, 2, 1\}$.

4 Dynamic Properties of CSAs

The goal of this Section is to investigate dynamic properties of CSAs, in particular robustness and safety on synchronization. We try to individuate classes of CSAs where such properties can be checked with algorithms and for this we employ tools from formal language theory and from temporal logic. Because of lack of space we omit the proofs, however complete proofs of all the results can be found in the technical report [5].

4.1 Robustness of CSAs

Before investigating robustness of CSAs we state the result that CSAs are equivalent (in terms of Parikh images) to matrix grammars without a.c. (hence to partially blind counter machines, [9]).

In particular, for an arbitrary CSA, $\Pi = (A, T, C, R)$, there exists a matrix grammar without a.c., G, with terminal alphabet T, such that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$, and vice-versa. Matrices can indeed simulate the application of the rules of the CSA because the rules are applied in an asynchronous manner. On the other hand, a CSA with a single agent can simulate a matrix grammar. The detailed proof of the result can be found in [5] (Theorem 8).

Theorem 3 *For an arbitrary CSA, Π, with terminal alphabet T, there exists a matrix grammar without a.c., G, with terminal alphabet T such that $Ps_T^{asyn}(\Pi) = Ps_T(L(G))$ and vice versa.*

We are now ready to define and to investigate robustness of CSAs against perturbations of some of the features of the colony. For this purpose we use a similar idea of robustness as employed in [12] in the framework of grammar systems, adapted here to the proposed CSAs. *We want to investigate situations where either some of the agents (i.e., cells) or some of the rules (i.e., intra or intercellular actions) of the colony do not function.* We would like to know the consequences to the result of the colony. We will investigate CSAs that are robust, e.g. where the produced result does not change critically if one or more agents cease to exist in the colony or if one or more rules stop working. As discussed in the Motivations, this can model the fact that CSAs always stop in a "correct steady state", independently of agents or rules failure.

We can formalize these notions in the following way.

Let $\Pi = (A, T, C, R)$ be an arbitrary CSA.

We say that Π' is an *agent-restriction* of Π if $\Pi' = (A, T, C', R)$ with $C' \subseteq C$. Π' is a CSA where some of the agents originally present in Π no longer work, i.e., as though they are absent from the colony.

We consider a *rule-restriction* of Π obtained by removing some or possibly all of the rules. Then, $\Pi' = (A, T, C, R')$ is a *rule-restriction* of Π if $R' \subseteq R$. In this case some of the rules do not work, as if, once again, they are absent from the colony.

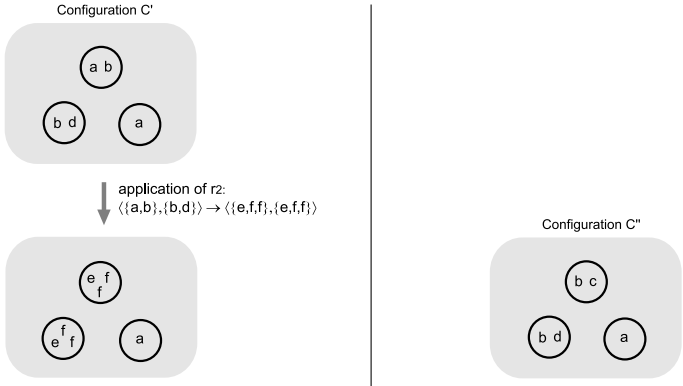
We say that a CSA, Π, is *robust* when a *core result*, i.e., the minimally acceptable result, is preserved when considering proper restrictions of it. Formally, by a core result of Π we mean *part* of the result produced by Π, hence a subset of the set of vectors generated by Π. We define these subsets by making an intersection with a regular set of vectors taken from $PsREG$. The intersection selects the regular property of the core result we are interested in. Note that the core result may be infinite.

Questions about robustness can then be formalized as follows.

Consider an arbitrary CSA, Π, an arbitrary agent- or rule - restriction Π' of Π and an arbitrary set S from $PsREG$. Is it possible to check whether or not $Ps^{asyn}(\Pi) \cap S \subseteq Ps^{asyn}(\Pi')$, i.e., whether Π is robust against the restriction Π' in the sense that Π' will continue to generate at least the core result defined by the intersection of $Ps^{asyn}(\Pi)$ and S)?

Example 4 We produce a small example that clarifies the introduced notion of robustness in the case of agent-restriction, considering as core result specific contents of the agents (the other cases are similar).

Consider Π given in Example 2. Suppose we fix as core result the set of vectors $\{(1, 2)\}$, where it can be clearly obtained by intersection of $Ps_T^{asyn}(\Pi)$ and $\{(1, 2)\}$. Π is robust when an occurrence of agent bc is deleted from its initial configuration. In fact, if we consider $\Pi' = (A, T, C', R)$ with $C' = \{(ab, 1), (bd, 1), (a, 1)\}$ we have that $Ps_T^{asyn}(\Pi') = \{(1, 2), \overline{0}\}$, which still contains the defined core result. The single computation of Π' is represented in Figure 3(a).



(a) Π': agent bc removed from C | (b) Π'': agent ab removed from C

Figure 3: The robustness and lack of robustness of (a) Π' and (b) Π'' from Example 4 when agents bc and ab, respectively, are removed from C.

On the other hand, Π is not robust when an occurrence of ab is deleted from its initial configuration. In fact, if we consider $\Pi'' = (A, T, C'', R)$ with $C'' = \{(bd, 1), (bc, 1), (a, 1)\}$ we have that $Ps_T^{asyn}(\Pi'') = \{\overline{0}\}$, which does not contain the core result. The single computation of Π'', i.e., the one halting in the initial configuration (no rule can be applied), is represented in Figure 3(b).

We now analyse the case of agent-restrictions, producing a negative result.

Theorem 5 *It is undecidable whether or not for an arbitrary CSA, Π, with arbitrary terminal alphabet T, arbitrary agent restriction Π' of Π and arbitrary set S from $PsREG_T$, $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$.*

The proof of the above Theorem is based on Theorem 3 and that, given two arbitrary matrix grammars without a.c. M and M', it is undecidable whether or not $L(M) \subseteq L(M')$ (see, e.g., [6], [8] and [9]).

Informally, Theorem 5 says that there is no algorithm to check whether or not a CSA is robust against arbitrary deletion of agents from the initial configuration. This result depends critically on the fact that the core result corresponds to a specific internal contents that the agents must have in the halting configurations. In fact, when we consider *weaker* core results we can get a positive result. For instance, suppose we take as core result a specific *magnitude* that the agents must have in the halting configurations. This means that we collect, for a CSA Π the set of numbers $N^{asyn}(\Pi)$. In this case the robustness problem can be rephrased in the following manner.

Consider an arbitrary CSA, Π, with an arbitrary agent- or rule-restriction Π' of Π and an arbitrary set S from $NREG$. Is it possible to decide whether or not $N^{asyn}(\Pi) \cap S \subseteq N^{asyn}(\Pi')$, i.e., whether Π is robust against the restriction Π' such that Π' can still generate at least the core result defined by the intersection $N^{asyn}(\Pi) \cap S$? Based on the fact that every language over a one letter alphabet produced by a matrix grammar without a.c. is regular (see [6]), on the equality of Theorem 3 we obtain the following corollary.

Corollary 1 *For an arbitrary CSA, Π, there exists a regular language L such that $N^{asyn}(\Pi) = NL$ and vice versa.*

Because containment of regular languages is algorithmically decidable (see, e.g., [10]), we obtain the following result.

Theorem 6 *It is decidable whether or not, for an arbitrary CSA, Π, arbitrary agent restriction Π' of Π and arbitrary set S from $NREG$, $N^{asyn}(\Pi) \cap S \subseteq N^{asyn}(\Pi')$.*

Informally, the above result says that it is possible to check in an efficient way whether or not a CSA is robust against arbitrary deletion of agents, subject to the core result being defined in terms of magnitudes of agents.

We can also investigate the case when rule-restrictions are considered and we obtain similar results. With a similar idea to that of Theorem 5, we obtain the following negative result.

Theorem 7 *It is undecidable whether or not, for an arbitrary CSA,* Π*, with arbitrary terminal alphabet* T*, arbitrary rule restriction* Π' *of* Π *and arbitrary set* S *from* $PsREG_T$*,* $Ps_T^{asyn}(\Pi) \cap S \subseteq Ps_T^{asyn}(\Pi')$*.*

However, using the same ideas as those in Theorem 6 we get a positive result.

Theorem 8 *It is decidable whether or not, for an arbitrary CSA,* Π*, arbitrary rule restriction* Π' *of* Π *and arbitrary set* S *from* $NREG$*,* $N(\Pi) \cap S \subseteq N(\Pi')$*.*

Note, however, that even if robustness against rule absence is in many cases undecidable, it is still possible to decide whether a rule (internal or synchronization) is used or not by a CSA. So, if a rule is not used we can remove it and the colony will be robust against such deletion.

Theorem 9 *It is decidable whether or not, for an arbitrary CSA,* $\Pi = (A, C, T, R)$*, and an arbitrary rule* r *from* R*, there exists at least one asynchronous computation for* Π *containing at least one configuration obtained by applying at least one occurrence of rule* r*.*

The proof is based on the result stated by Theorem 3 and on the fact that membership and emptiness for matrix grammars without a.c. can be algorithmically decided ([6]). The idea of the proof is to reduce the problem to decide if the language produced by a matrix grammar without a.c. is the empty one.

4.2 A computational tree logic for CSAs

In this section we continue the investigation of the dynamic properties of CSAs and for this purpose we introduce a *computational tree logic (CTL temporal logic)* to formally specify, verify and model-check properties of CSAs. An introduction to the basic notions and results of temporal logics can be found in [1, 18].

Temporal logics are the most used logics in model-checking analysis: efficient algorithms and tools having already been developed for them, e.g. NuSMV [20]. They are devised with operators for expressing and quantifying on possible evolutions or configurations of systems. For instance, for an arbitrary system it is possible to specify properties such as *'for any possible evolution,* ϕ *is fulfilled'*, *'there exists an evolution such that* ϕ *is not true'*, *'in the next state* ϕ *will be satisfied'*, *'eventually* ϕ *will be satisfied'* and *'*ϕ *happens until* ψ *is satisfied'*, with ϕ and ψ properties of the system. We show how to use these operators to formally specify and verify complex properties of CSAs, such as *'the agent will always eventually reach a certain configuration'*, or *'rule* r *is not applicable until rule* r' *is used'*, etc.

In what follows we denote by $CSA_m^{A,T,R}$ the class of all CSAs having the alphabet A, terminal alphabet T, set of rules R over A and degree m.

Definition 4.1 (Preconditions) *Let* A *be an arbitrary alphabet and* R *an arbitrary set of rules over* A*. We define the mapping* $prec : R \to 2^{\mathbb{M}(A)}$ *by*

- *if* $r \in R$ *is the evolution rule* $u \to v$ *then* $prec(r) = \{u\}$*.*

- *if $r \in R$ is a synchronization rule $\langle u, v\rangle \rightarrow \langle u', v'\rangle$ then $prec(r) = \{u\} \cup \{v\}$.*

We define $prec(R) = \bigcup_{r\in R} prec(r)$.

We now extend the definition of *asyn*-evolutions for a given CSA by introducing the notion of *asyn-complete evolution* defined for arbitrary classes of CSAs.

In what follows, let $\mathcal{C} = CSA_m^{A,T,R}$ be a class of all the CSAs having alphabet A, terminal alphabet T, set of rules R over A, degree m, with A, T, R and m arbitrarily chosen.

Definition 4.2 (*asyn*-complete evolutions) *A sequence of CSAs $\langle \Pi_0, \Pi_1, \Pi_2, \ldots, \Pi_i, \ldots\rangle$ with $\Pi_i = (A, T, C_i, R) \in \mathcal{C}$, $i \geq 0$, is called* asyn-complete evolution *in $\mathcal{C}$ starting in Π_0 if $\langle C_0, C_1, C_2, \ldots C_i, \ldots\rangle$, $i \geq 0$, is a halting or an infinite asyn-evolution of Π_0.*

We denote by $E_{\mathcal{C}}^{asyn}(\Pi_0)$ the set of all asyn-complete evolutions in $\mathcal{C}$ starting at Π_0.

Let $e = \langle \Pi_0, \Pi_1, \ldots, \Pi_i, \Pi_{i+1} \ldots\rangle$ be an arbitrary asyn-complete evolution in $\mathcal{C}$ starting in Π_0. We call $\langle \Pi_i, \Pi_{i+1}, \ldots\rangle$, $i \geq 0$, an i-suffix evolution[4] *of e and we denote it by e_i.*

Definition 4.3 (Syntax of $\mathcal{L}_{\mathcal{C}}$) *The set $AP(\mathcal{C})$ is defined by:*

- $\top \in AP(\mathcal{C})$.
- $prec(R) \subseteq AP(\mathcal{C})$.
- *if $w_1, w_2, \ldots, w_i \in prec(R) \cup \{\top\}$, $i \leq m$, then $w_1 \oplus \ldots \oplus w_i \in AP(\mathcal{C})$.*

We call the elements of $AP(\mathcal{C})$ atomic formulas *of the logic $\mathcal{L}_{\mathcal{C}}$.*
We define the configuration formulas *of $\mathcal{L}_{\mathcal{C}}$ and the* evolution formulas *of $\mathcal{L}_{\mathcal{C}}$ in the following way.*

- *any atomic formula of $\mathcal{L}_{\mathcal{C}}$ is a configuration formula of $\mathcal{L}_{\mathcal{C}}$.*
- *if ϕ, ψ are configuration formulas of $\mathcal{L}_{\mathcal{C}}$ then $\neg\phi$ and $\phi \wedge \psi$ are configuration formulas of $\mathcal{L}_{\mathcal{C}}$.*
- *if ϕ is an evolution formula of $\mathcal{L}_{\mathcal{C}}$ then $E\phi$ is a configuration formula of $\mathcal{L}_{\mathcal{C}}$.*
- *if ϕ, ψ are configuration formulas of $\mathcal{L}_{\mathcal{C}}$ then $X\phi$ and $\phi U \psi$ are evolution formulas of $\mathcal{L}_{\mathcal{C}}$.*

The configuration formulas and evolution formulas of $\mathcal{L}_{\mathcal{C}}$ form the language of $\mathcal{L}_{\mathcal{C}}$.

The meanings of $\top, \neg, \wedge$ are those from classical logic and we consider the derived operators for implication $\Rightarrow$ and disjunction $\vee$ defined as in classical propositional logic. In addition, we have the temporal operators: $E\phi$ that expresses an existential quantification on evolutions, $X\phi$ which means "at the next configuration ϕ is satisfied" and $\phi U \psi$ which means "ϕ is satisfied until ψ is satisfied". In what follows, the properties we can express by using these operators are checked for some models called temporal structures.

[4]Observe that for an arbitrary *asyn*-complete evolution e in $\mathcal{C}$, for each $i \geq 0$, e_i is also a *asyn*-complete evolution in $\mathcal{C}$.

Definition 4.4 (Temporal structures) *We define the structure* $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ *as follows:*

- $\mathcal{S} \subseteq \mathcal{C}$*, such that if* $\Pi_0 \in \mathcal{S}$ *then* $\{\Pi_1, \Pi_2, \ldots \mid \langle \Pi_0, \Pi_1, \Pi_2, \ldots \rangle \in E_{\mathcal{C}}^{asyn}(\Pi_0)\} \subseteq \mathcal{S}$*.*
- $\mathfrak{R} \subseteq \mathcal{S} \times \mathcal{S}$*, such that* $(\Pi_1, \Pi_2) \in \mathfrak{R}$ *iff there exists* $\langle \Pi_1, \Pi_2, \ldots \rangle \in E_{\mathcal{C}}^{asyn}(\Pi_1)$*.*

We call $\mathcal{T}_{\mathcal{C}}^{asyn}$ *a temporal structure in* $\mathcal{C}$*.*

Definition 4.5 (CSA-Semantics) *Let* $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ *be a temporal structure in* $\mathcal{C}$*. For an arbitrary* $\Pi \in \mathcal{S}$*, an arbitrary* $e \in E_{\mathcal{C}}^{asyn}(\Pi)$ *and an arbitrary formula* ϕ *from the language of* $\mathcal{L}_{\mathcal{C}}$*, we define coinductively the satisfiability relations* $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ *and* $\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$ *by:*

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \top$ *always.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models w$ *for* $w \in prec(R)$ *iff* $C_\Pi = \{(w', 1)\}$ *and* $w \subseteq w'$*.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models w_1 \oplus w_2 \oplus \ldots \oplus w_i$ *for* $w_j \in prec(R) \cup \{\top\}, 1 \leq j \leq i$ *iff* $C_\Pi = C_1 + C_2 + \ldots + C_i$ *s.t. for any* $w_j \neq \top$*,* $1 \leq j \leq i$*,* $C_j = \{(w_j + u_j, 1)\}$ *for some* $u_j \in \mathbb{M}(A)$*.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi \wedge \psi$ *iff* $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ *and* $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \psi$*.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \neg\phi$ *iff* $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \not\models \phi$*.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models E\phi$ *iff there exists* $e \in E_{\mathcal{C}}^{asyn}(\Pi)$ *such that* $\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$*.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi U \psi$ *iff there exists* $i \geq 0$ *such that* $\mathcal{T}_{\mathcal{C}}^{asyn}, e_i \models \psi$ *and for all* $j \leq i$ $\mathcal{T}_{\mathcal{C}}^{asyn}, e_j \models \phi$*.*

$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models X\phi$ *iff* $\mathcal{T}_{\mathcal{C}}^{asyn}, e_1 \models \phi$*.*

Definition 4.6 (Validity and satisfiability) *A configuration formula* ϕ *(evolution formula* ϕ*) from* $\mathcal{L}_{\mathcal{C}}$ *is* valid *iff for every temporal structure* $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ *in* $\mathcal{C}$ *and any* $\Pi \in \mathcal{S}$ *(any* $e \in E_{\mathcal{C}}^{asyn}(\Pi)$*, resp.) we have* $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ *(*$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$*, resp.). A configuration formula* ϕ *(evolution formula* ϕ*) is satisfiable iff there exists a temporal structure* $\mathcal{T}_{\mathcal{C}}^{asyn} = (\mathcal{S}, \mathfrak{R})$ *and a* $\Pi \in \mathcal{S}$ *(an* $e \in E_{\mathcal{C}}^{asyn}(\Pi)$*, resp.) such that* $\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models \phi$ *(*$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$*, resp.).*

Definition 4.7 (Derived formulas) *We define the following derived formulas for* $\mathcal{L}_{\mathcal{C}}$*.*

$A\phi = \neg E \neg \phi$*.*

$F\phi = \top U \phi$*.*

$G\phi = \neg F \neg \phi$*.*

The semantics of the derived formulas are the following.

$\mathcal{T}_{\mathcal{C}}^{asyn}, \Pi \models A\phi$ iff for any $e \in E_{\mathcal{C}}^{asyn}(\Pi)$ we have $\mathcal{T}_{\mathcal{C}}^{asyn}, e \models \phi$.

$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models F\phi$ iff there exists $i \geq 0$ such that $\mathcal{T}_{\mathcal{C}}^{asyn}, e_i \models \phi$.

$\mathcal{T}_{\mathcal{C}}^{asyn}, e \models G\phi$ iff for any $i \geq 0$ we have $\mathcal{T}_{\mathcal{C}}^{asyn}, e_i \models \phi$.

$A\phi$ is a universal quantification on evolutions. $F\phi$ means "eventually ϕ is satisfied" (i.e., $F\phi$ is satisfied by an evolution that contains at least one configuration that has the property ϕ). $G\phi$ means "globally ϕ is satisfied" (i.e., $G\phi$ is satisfied by an evolution that contains only configurations satisfying ϕ).

Theorem 10 (Decidability) *The satisfiability, validity and model-checking problems for $\mathcal{L}_\mathcal{C}$ against the CSA-semantics are decidable.*

Proof. The result derives from the fact that CTL logic is decidable (see, e.g., [18, 1]) and from the fact that $AP(\mathcal{C})$, the set of atomic formulas, is a finite set. □

To show the potential of the introduced logic we give a small example of properties that can be specified. We pose the question whether or not during any evolution the agents can always synchronize when they are *ready* to do so.

In other words, given an arbitrary CSA, Π, and an arbitrary rule $r : \langle u, v\rangle \to \langle u', v'\rangle$, we would like to check whether or not it is true that, whenever during an evolution of Π, a configuration with an agent w_1, where $u \subseteq w_1$, is reached, then in the same configuration there is also an agent w_2 with $v \subseteq w_2$ (so rule r can actually be applied). If this is true we say that Π is *safe on synchronization* of rule r.

This property can be expressed in the proposed temporal logic by the following formula.

$$AG((u \oplus \top) \Rightarrow (u \oplus v \oplus \top)).$$

Taking a CSA, Π_0, from $\mathcal{C}$. If we consider the introduced CSA-semantics we have that:

$\mathcal{T}_\mathcal{C}^{asyn}, \Pi_0 \models AG((u \oplus \top) \Rightarrow (u \oplus v \oplus \top))$
iff for any $e \in E_\mathcal{C}^{asyn}(\Pi_0)$ we have $\mathcal{T}_\mathcal{C}^{asyn}, e \models G((u \oplus \top) \Rightarrow (u \oplus v \oplus \top))$
iff for any $e = \langle \Pi_0, \Pi_1, \ldots, \Pi_i, \ldots\rangle \in E_\mathcal{C}^{asyn}(\Pi_0)$ and any $i \geq 0$ we have
$\mathcal{T}_\mathcal{C}^{asyn}, \Pi_i \models (u \oplus \top) \Rightarrow (u \oplus v \oplus \top)$.

This means that if any configuration present in a *asyn*-evolution of Π_0 satisfies $u \oplus \top$ then it will also satisfy $u \oplus v \oplus \top$.

In fact, we know that $\mathcal{T}_\mathcal{C}^{asyn}, \Pi_i \models u \oplus \top$ iff $C_{\Pi_i} = C_1 + C_2$, $C_1, C_2 \in \mathbb{M}(\mathbb{M}(A))$ and $C_1 = \{(u + u', 1)\}$, i.e., the configuration of Π_i contains an agent w that contains u.

Similarly, $\mathcal{T}_\mathcal{C}^{asyn}, \Pi_i \models u \oplus v \oplus \top$ iff $C_{\Pi_i} = C'_1 + C'_2 + C'_3$, $C'_1, C'_2, C'_3 \in \mathbb{M}(\mathbb{M}(A))$ and $C'_1 = \{(u + u'', 1)\}$, $C'_2 = \{(v + v', 1)\}$, i.e., the configuration of Π_i contains two agents w_1 and w_2 such that $u \subseteq w_1$ and $v \subseteq w_2$, which precisely indicates that Π_0 is safe on synchronization of rule $r : \langle u, v\rangle \to \langle u', v'\rangle$.

5 Prospects

In this paper we have defined a basic model of Colonies of Synchronizing Agents, however several enhancements to this are already in prospect. Primary among these is the addition of *space* to the colony. Precisely, each agent will have a triple of co-ordinates corresponding to its position in Euclidean space and the rules will be similarly endowed with the ability to modify an agent's position. A further extension of this idea is to give each agent an *orientation*, i.e. a rotation relative to the spatial axes, which may also be modified by the application of rules.

The idea is to make the application of a rule dependent on either an absolute position (thus directly simulating a chemical gradient) or on the relative distance between agents in the case of synchronization. Moreover, in the case of the application of a synchronization rule, the ensuing translation and rotation of the two agents may be defined *relative to each other*. In this way it will be possible to simulate reaction-diffusion effects, movement and local environments.

Some additional biologically-inspired primitives are also planned, such as agent *division* (one agent becomes two) and agent *death* (deletion from the colony). These primitives can simulate, for example, the effects of mitosis, apoptosis and morphogenesis. In combination with the existing primitives, it will be possible (and is planned) to model, for example, many aspects of the complex multi-scale behaviour of the immune system.

With the addition of the features just mentioned, it will also be interesting to extend the investigation and proofs given above to identify further classes of CSAs demonstrating robustness and having decidable properties. It is hoped that this approach will then provide insight in challenging areas of systems biology.

References

[1] M. Ben-Ari, A. Pnueli, Z. Manna. The Temporal Logic of Branching Time. *Acta Inf.*, 20, 1983.

[2] F. Bernardini, R. Brijder, G. Rozenberg, C. Zandron. Multiset-Based Self-Assembly of Graphs. *Fundamenta Informaticae*, 75, 2007.

[3] F. Bernardini, M. Gheorghe. Population P Systems. *Journal of Universal Computer Science*, 10(5), 2004.

[4] C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, editors. *Multiset Processing: Mathematical, Computer Science, and Molecular Computing Point of View*, LNCS 2235, Springer-Verlag, 2001.

[5] M. Cavaliere, R. Mardare, S. Sedwards. Colonies of Synchronizing Agents. Technical Report CoSBi 11/2007. Available at `www.cosbi.eu/Rpty_Tech.php`

[6] J. Dassow, Gh. Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin, 1989.

[7] A. Ilachinski. *Cellular Automata - A Discrete Universe.* World Scientific Publishing, 2001.

[8] R. Freund, Gh. Păun, O.H. Ibarra, H.-C.Yen. Matrix Languages, Register Machines, Vector Addition Systems. In *Proc. Third Brainstorming on Membrane Computing.* RGCN Report 01/2005, Sevilla, 2005. Available at `www.gcn.us.es`

[9] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3), 1978.

[10] J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[11] J. Kelemen, A. Kelemenová, Gh. Păun. Preview of P Colonies - A Biochemically Inspired Computing Model. *Proceedings of Workshop on Artificial Chemistry*, ALIFE9, Boston, USA, 2004.

[12] J. Kelemen, Gh. Păun. Robustness of Decentralized Knowledge Systems: A Grammar-Theoretic Point of View. *Journal Expt. Theor. Artificial Intelligence*, 12, 2000.

[13] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón. Tissue P Systems. *Theoretical Computer Science*, 296(2), 2003.

[14] J. Mata, M. Cohn. Cellular Automata-Based Modelling Program: Synthetic Immune Systems. *Immunol Rev*, 207, 2007.

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

[16] Gh. Păun. Introduction to Membrane Computing. In G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, editors, *Applications of Membrane Computing*. Springer-Verlag, Berlin, 2006.

[17] G. Rozenberg, A. Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[18] J. Van Benthem. Temporal logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming: Epistemic and Temporal reasoning*. Oxford University Press, 1995.

[19] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[20] `http://nusmv.irst.itc.it/`

[21] `http://psystems.disco.unimib.it`

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 52 - 63, 2007.

Networks of Mealy Multiset Automata

Gabriel Ciobanu and Mihai Gontineac

"A.I.Cuza" University of Iaşi, Romania
Faculty of Computer Science and Faculty of Mathematics
gabriel@info.uaic.ro, gonti@uaic.ro

Abstract

We introduce the networks of Mealy multiset automata, and study their computational power. The networks of Mealy multiset automata are computationally complete.

1 Learning from Molecular Biology

Systems biology represents a new cross-disciplinary approach in biology which has only recently been made possible by advances in computer science and technology. As it is mentioned in [10], it involves the application of experimental, theoretical, and modelling techniques to the study of biological organisms at all levels. Adding new abstractions, discrete models and methods able to help our understanding of the biological phenomena, systems biology may provide predictive power, useful classifications, new paradigms in computing and new perspectives on the dynamics of various biological systems.

Recent promising work [1] employs automata theory as an efficient tool of describing and controlling gene expression (a small automaton is encoded by DNA strands and then it is used in logical control of gene expression).

In [2], we present a way of interaction between gene machine and protein machine, namely the process of making proteins, in abstract terms of Mealy automata, transformation semigroups and abstract operations.

The Mealy automaton proposed as a formal model of the genetic message translation is a minimal one that accepts the mRNA messages and terminates the translation process (according to [9], there is no appropriate formalism for the process of translation).

However molecular biology "deals" not only with sequences, but also with multisets. The biological cells are "smart" enough to put together at work sequences and multisets of atoms and molecules, so if we try to get models from their functioning, we should not restrict ourselves to dealing only with sequential machines (like classical automata). To deal with multisets, the main approach is given by membrane systems [11]. There is also introduced and studied an automaton-like machine to work with multisets [7], i.e. *multiset automaton*. At a first glance, it seems that they are nothing else but weighted automata with weights in the semiring of positive integers. In [8] it is proven that such automata (weighted automata with

weights in the semiring of positive integers) have the same power as finite automata (accept only regular languages). In fact, a careful reader should remark that a multiset automaton is not a sequential machine, and it is not working with sequences of multisets as in the case of weighted automata. A multiset automaton accepts, together with a sequence of multisets, an entire class, namely the class of that sequence obtained by "abelianization" (as an example, together with the sequence, say *aab*, it accepts *aba* and *baa*). In this manner, multiset automata become very powerful (see [7], for details). Mealy multiset automata, [3], can be viewed as the corresponding Mealy machine. We study some of their (co)algebraic properties in [3] and [4] and we connect these properties with various aspects of their behaviour. In [5], we organize them in a *P machine*, in order to simulate a P system. However, the biological systems are not always organized in a hierarchical manner. This means that we also have to organize sets of Mealy multiset automata in networks. In order to obtain the computing power for networks of such automata, we relate them to neural P systems, proving that networks of Mealy multiset automata are computationally complete.

2 Networks of Mealy Multiset Automata

In order to define networks of Mealy multiset automata, we can connect these automata in many ways, having both parallel and serial connections. In [3] we define the *restricted direct product* of MmA for the parallel case, and the *cascade product* for a serial connection.

2.1 Multisets

The evolution rules performed by membranes are multiset operators; the multiset operators are associative and commutative, and have also an identity.

A *multiset* over an alphabet $A = \{a_1, a_2, ..., a_n\}$ is a mapping $\alpha : A \to \mathbb{N}$. It can be represented by $\{(a_1, \alpha(a_1)), (a_2, \alpha(a_2)), ..., (a_n,\alpha(a_n))\}$. Inspired from formal power polynomials, we denote by $\mathbb{N}\langle A\rangle = \{\alpha : A \to \mathbb{N} \mid \alpha \text{ is a mapping}\}$ the set of all multisets on A . The structure of $\mathbb{N}\langle A\rangle$ is mainly an additive one, since we add multiplicities of appearance (in fact, it is induced by the addition in $\mathbb{N}$). This argument is sustained also by the chemical reactions that are the base of the biological modelling. They provide a notation for defining the way a biological system evolves.

If $\alpha, \beta \in \mathbb{N}\langle A\rangle$, then their *sum* is the multiset $(\alpha + \beta) : A \to \mathbb{N}$ defined by $(\alpha + \beta)(a_i) = \alpha(a_i) + \beta(a_i)$, $i = \overline{1, n}$. Moreover, if we consider the letters from A as multisets, i.e. a_i is given by μ_{a_i}, where $\mu_{a_i} : A \to \mathbb{N}$, $\mu_{a_i}(a_i) = 1$ and $\mu_{a_i}(a_j) = 0$ for all $j \neq i$, then we can express every multiset $\alpha \in \mathbb{N}\langle A\rangle$ as a linear combination of a_i, i.e. $\alpha = \sum_{i=1}^{n} \alpha(a_i) \cdot a_i$ (see also [3]). The *length* of a multiset α, denoted by $|\alpha|$, is defined by $|\alpha| = \sum_{i=1}^{n} \alpha(a_i)$.

We can define an external operation $m\alpha = \sum_{i=1}^{n}(m\alpha(a_i)) \cdot a_i$, for all $m \in \mathbb{N}$ and $\alpha \in \mathbb{N}\langle A\rangle$.

Proposition 1 *$\mathbb{N}\langle A\rangle$ has a structure of $\mathbb{N}$-semimodule (semimodule over the semiring of positive integers).*

2.2 Mealy multiset automata

Roughly speaking, a *Mealy multiset automaton* (MmA) consists of a *storage location* (a *box* for short) in which we place a multiset over an input alphabet and a device to translate the multiset into a multiset over an output alphabet. We have a detection head that detects whether or not a given multiset appears in the multiset available in the box. The multiset is removed from the box whenever it is detected, and the automaton inserts a multiset over the output alphabet. The output alphabet should be different from the input alphabet; if they are the same, we mark the output symbols just to make different the output and input alphabet. In this way the output symbols cannot be viewed by the detection head. This automaton stops when no further move is possible. We say that the sub-multiset read by the head was translated to a multiset over the output alphabet. We give here only the definitions and the properties that we need for networks of MmA. For more informations see [3] or [4].

From the formal point of view, a *Mealy multiset automaton* is a construct $\mathcal{A} = (Q, V, O, f, g, q_0)$ where

1. Q is a finite set, the set of *states*;
2. $q_0 \in Q$ is special state, which is both initial and final;
3. V is a finite set of objects, the *input alphabet*;
4. O is a finite set of objects, the *output alphabet*, such that $O \cap V = \emptyset$;
5. $f : Q \times \mathbb{N}\langle V\rangle \to \mathcal{P}(Q)$ is the *state-transition (partial) mapping*;
6. $g : Q \times \mathbb{N}\langle V\rangle \to \mathcal{P}(\mathbb{N}\langle O\rangle)$ is the *output (partial) mapping.*

If $\mid f(q,a) \mid \leq 1$ we say that $\mathcal{A}$ is Q- *deterministic* and if $\mid g(q,a) \mid \leq 1$ our automaton is O-*deterministic.*

An MmA $\mathcal{A}$ receives a multiset in its box, and processing this multiset it passes through different *configurations.* It starts with a multiset from $\mathbb{N}\langle V\rangle$ and ends with a multiset from $\mathbb{N}\langle V \cup O\rangle$. A *configuration* of $\mathcal{A}$ is a triple $(q, \alpha, \bar{\beta})$ where $q \in Q, \alpha \in \mathbb{N}\langle V\rangle$, $\bar{\beta} \in \mathbb{N}\langle O\rangle$. We say that a configuration $(q, \alpha, \bar{\beta})$ *passes* to $(s, \alpha - a, \bar{\beta} + \bar{b})$ (or, that we have a *transition* between those configurations) if there is $a \subseteq \alpha$ such that $s \in f(q,a), \bar{b} \in g(q,a)$. We denote this by $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$, and by $\vdash^*$ the reflexive and transitive closure of $\vdash$. We can alternatively define a configuration to be a pair (q, α) where $\alpha \in \mathbb{N}\langle V \cup O\rangle$ and the transition relation is $(q, \alpha) \vdash (s, \alpha - a + \bar{b})$, with the same conditions as above.

Behaviour is often appropriately viewed as consisting of both dynamics and observations, which have to do with change of states and partial access to states, respectively. The main advantage of an MmA is that it has an output function that can play the main role in observability, i.e. we do not have to construct an other machine to describe the MmA' s behaviour.

Definition 1 *Let $\mathcal{A} = (Q, V, O, f, g)$ be a Mealy multiset automaton. The* general behaviour *of a state $q \in Q$ is a function* $\mathbf{beh}(q)$ *assigning to every multiset $\alpha \in \mathbb{N}\langle V\rangle$ the output multiset obtained after consuming α starting from q.*

When talking about the behaviour, we consider a specific order of consuming multisets, i.e. in terms of strings of multisets.

Definition 2 *Let $\mathcal{A} = (Q, V, O, f, g)$ be a Mealy multiset automaton. The* sequential behaviour *of a state $q \in Q$ is a function* $\mathbf{seqbeh}(q)$ *that assigns to every multiset $\alpha \in \mathbb{N}\langle V\rangle$ all the sequences of the output multisets obtained after consuming α starting from q.*

Example 1 *Suppose that we have the following sequence of transitions $(q, \alpha, \varepsilon) \vdash (q_1, \alpha - a_1, b_1) \vdash (q_2, \alpha - a_1 - a_2, b_1 + b_2) \vdash \ldots \vdash (q_n, \alpha - a_1 - \ldots - a_n, b_1 + \ldots + b_n)$ and MmA stops. Then $\mathbf{beh}(q)(\alpha) = b_1 + \ldots + b_n$ and $\mathbf{seqbeh}(q)(\alpha) \ni b_1 \ldots b_n$. Moreover, $b_1 + \ldots + b_n$ belongs to $\mathbb{N}\langle O\rangle$, while $b_1 \ldots b_n$ belongs to $(\mathbb{N}\langle O\rangle)^*$.*

Consider the canonical inclusion $i : \mathbb{N}\langle O\rangle \to (\mathbb{N}\langle AO\rangle)^*$ and the identity map $id : \mathbb{N}(O) \to \mathbb{N}(O)$. By the universal property of the free monoid, we know that there exists a unique homomorphism of monoids $\mathbf{I}_O : (\mathbb{N}\langle O\rangle)^* \to \mathbb{N}\langle O\rangle$ defined by $\mathbf{I}_O(b_1 \ldots b_n) = b_1 + \ldots + b_n$ such that $\mathbf{I}_O \circ i = id$. Since id is onto, it follows that $\mathbf{I}$ is onto, and so, applying the isomorphism theorem for monoids, we obtain that $(\mathbb{N}\langle O\rangle)^*/ker\mathbf{I}_O \backsimeq \mathbb{N}\langle O\rangle$. Moreover, for all the states q of a Mealy multiset automaton we have

$$\mathbf{I}_O \circ \mathbf{seqbeh}(q) = \mathbf{beh}(q).$$

Let $\mathcal{A}_i = (Q_i, V, O, f_i, g_i)$, and B_i their corresponding boxes, $i = \overline{1,n}$, a finite family of Mealy multiset automata. We can connect them in *parallel* in order to obtain a new MmA defined by $\mathcal{A} = \bigwedge_{i=1}^{n} \mathcal{A}_i = (\times_{i=1}^{n} Q_i, V, O^n, f, g)$, called the **restricted direct product** of $\mathcal{A}_i$, where:

- $f((q_1, q_2, \ldots, q_n), a) = (f_1(q_1, a), f_2(q_2, a), \ldots, f_n(q_n, a))$,
- $g((q_1, q_2, \ldots, q_n), a) = (g_1(q_1, a), g_2(q_2, a), \ldots, g_n(q_n, a))$,
- box of $\mathcal{A}$ is the disjoint union $\bigsqcup_{i=1}^{n} B_i$ of $\{B_i \mid i = \overline{1,n}\}$,
- a *configuration* of $\mathcal{A}$ is a triple $(q, \alpha, \bar{\beta})$, where $q = (q_1, q_2, \ldots, q_n)$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, and $\bar{\beta} = (\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_n)$,
- the *transition relation* of $\mathcal{A}$: $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$ iff $s_i \in f_i(q_i, a_i)$ and $\bar{b}_i \in g_i(q_i, a_i)$ for all $i \in \overline{1,n}$.

The *cascade product* is useful to describe a serial connection, and provide also some results in decompositions of such machines in irreducible ones.

Let $\mathcal{A} = (Q, V, O, f, g)$, $\mathcal{A}' = (Q', V', O', f', g')$ be two Mealy multiset automata. In order to connect them, we need a multiset mapping linking the output of one of them to the input of the other. This can be done using a $\mathbb{N}$-homomorphism from $\mathbb{N}\langle O'\rangle$ to $\mathbb{N}\langle V\rangle$ (this homomorphism can be obtained by using a mapping from O' to V). We denote by $\Lambda : \mathbb{N}\langle O'\rangle \to \mathbb{N}\langle V\rangle$ this homomorphism. Then we can define a mapping $\Omega : Q' \times \mathbb{N}\langle V'\rangle \to \mathbb{N}\langle V\rangle$ by $\Omega(q', a') = \Lambda(g'(q', a'))$.

- This mapping gives us *the cascade product induced by* Ω:

$$\mathcal{A}\Omega\mathcal{A}' = (Q \times Q', V', O, f^{\Omega}, g^{\Omega})$$

  where $f^{\Omega}((q, q'), a') = (f(q, \Omega(q', a')), f'(q', a'))$, $g^{\Omega}((q, q'), a') = g(q, \Omega(q', a'))$, for all $a' \in \mathbb{N}\langle V'\rangle, (q, q') \in Q \times Q'$.

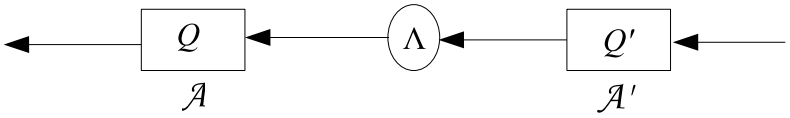
- The transition relation becomes $((q, q'), \alpha', \bar{\beta}) \vdash ((s, s'), \alpha' - a', \bar{\beta} + \bar{b})$ if there is $a' \subseteq \alpha'$ such that $(s, s') = f^{\Omega}((q, q'), a')$ and $\bar{b} = g^{\Omega}((q, q'), a')$, where $a', \alpha' \in \mathbb{N}\langle V'\rangle$, $(q, q') \in Q \times Q'$, and $\bar{\beta} \in \mathbb{N}\langle O\rangle$.

We can alternatively define the transition relation by

$$((q, q'), \alpha', \bar{\beta}) \vdash ((s, s'), \alpha' - a', \bar{\beta} + \bar{b}),$$

if there is $a' \subseteq \alpha'$ such that $s{=}f(q, \Lambda(g'(q', a')))$, $s'{=}f'(q', a')$, $\bar{b}{=}g(q, \Lambda(g'(q', a')))$, where $a', \alpha' \in \mathbb{N}\langle V'\rangle$, $(q, q') \in Q \times Q'$, $\bar{\beta} \in \mathbb{N}\langle O\rangle$.

The graphical representation of the cascade product is given in the following figure:



In order to obtain the behaviour of a network of MmA's, we should also consider the behaviour of the cascade product. Roughly speaking, the two types of behaviour depends mainly on the corresponding behaviours of $\mathcal{A}'$. On the other hand, when we have a cascade product, the observable part is strongly connected with the observations that could be made after we pass through $\mathcal{A}$. We may also emphasize the important role played by the connection homomorphism given by Λ.

Theorem 2 *Let $\mathcal{A} = (Q, V, O, f, g)$, $\mathcal{A}' = (Q', V', O', f', g')$ be two MmA's, $\mathcal{A}\Omega\mathcal{A}'$ their cascade product, and (q, q') a state of this product. The behaviour of (q, q') is* $\mathbf{beh}((q, q')) = \mathbf{beh}(q) \circ \Lambda \circ \mathbf{beh}(q')$.

If we want to get the sequential behaviour starting from $\mathbf{beh}((q, q')) = \mathbf{beh}(q) \circ \Lambda \circ \mathbf{beh}(q')$, then $(\mathbf{I}, \mathbf{I}') \circ \mathbf{seqbeh}((q, q')) = (\mathbf{I} \circ \mathbf{seqbeh}(q)) \circ \Lambda \circ (\mathbf{I}' \circ \mathbf{seqbeh}(q'))$.

Other properties of MmA's, their behaviours and bisimulation relations are presented in [3] and [4].

2.3 Networks of automata

The formal description of a network of Mealy multiset automata is not intuitive. On the other hand, these networks could be very powerful, so we think that they deserve our attention. We can consider several variants of such networks. Some of them can have no inter-communication and, in this case, the network is, in fact, a bigger MmA. The same remark can be done if we have only MmA connected in a serial manner, without any ramifications. The case that we consider in this paper is inspired by the definition of neural P systems (nP systems). Neural P systems are defined in [11] as a computing model inspired by the network of cells. Each cell has a finite state memory, and processes multisets of symbol (impulses); it can send some impulses (called excitations) to the neighbouring cells. It is proved that such networks are rather powerful: they can simulate Turing machines using a small number of cells, every cell having in a small number of states. It is also proved that, in appropriate organization, such a network can solve in linear time the Hamiltonian Path Problem.

We consider a set of MmA that can communicate by means of some *communication channels*. All of them have the same input alphabet V, and their boxes contain an input multiset over V (they can also have an empty multiset ε as input). The output alphabet has a "real" part O of output alphabet, and a "specific" part used for communication. The specific part is, in fact, a Cartesian product between the input alphabet V and the set of targets T (the set of the indexes of the MmA forming the net). We can also have a special MmA to collect in its box the result of the computation (i.e. a multiset over O) for such a network. Alternatively, we can consider as result of the computation the tuple of multisets obtained in the box of every MmA of the net.

Definition 3 *A* network of Mealy multiset automata *(shortly, nMmA) is a construct* $\mathcal{N} = (V, O, \{\mathcal{A}_i\}_{i=\overline{1,n}}, \{\Lambda_i\}_{i=\overline{1,n}}, B)$ *where:*

- $V = \{a_1, a_2, \ldots, a_m\}$ *is a finite set of objects, the* input alphabet*;*
- O *is the* output alphabet *such that* $O \cap V = \emptyset$*;*
- $\mathcal{A}_i = (Q_i, V, \overline{O}, f_i, g_i, s_{0,i})$ *are MmA's connected in the network. Their output alphabets are of the form* $\overline{O} = O \cup (V \times T)$*, where* $T = \{1, 2, \ldots, n\}$*;*
- B *is a box where* $\mathcal{N}$ *"receives" the output multiset. Depending on features that we consider for the net,* B *can be a specific box of a specific MmA in the network, or* B *can be the Cartesian product of all the boxes.*
- $\Lambda_i : \mathbb{N}\langle\overline{O}\rangle \to (\mathbb{N}\langle O\rangle \cup \mathbb{N}\langle V\rangle)^n$ *are the* communication mappings *associated to all the* $\mathcal{A}_i$*,* $i \in T$*.*

A *computation* starts with some input multisets $w_{0,i}$ in the boxes of the MmA's that are in their initial states, $s_{0,i}$; then we have a *big step* given by a translation (made by the MmA's, in fact by their restricted direct product $\bigwedge_{i=1}^{n} \mathcal{A}_i$) and a communication (done by $\{\Lambda_i\}$) - a kind of "parallel cascade product", since every MmA is in cascade with the restricted direct product of itself and the other MmA.

A *configuration* of the network is of the form (s, w), where $s = (s_1, s_2, \ldots, s_n)$ with $s_i \in Q_i$ is the *global state*, and $w = (w_1, \ldots, w_n)$ where $w_i \in \mathbb{N}\langle O\rangle \cup \mathbb{N}\langle V\rangle$.

A *transition between configurations* is denoted by $(s, w) \vdash (s', w')$ and is defined in the following manner:

$s' = (s'_1, s'_2, \ldots, s'_n)$, where $s'_i \in f_i(s_i, a_i)$ with $a_i \in \mathbb{N}\langle V\rangle$; we allow some of the a's to be ε if in the corresponding MmA there is no transition.

$w' = \Lambda_1(b_1) + \Lambda_2(b_2) + \ldots + \Lambda_n(b_n) + (w_1 - a_1, w_2 - a_2, \ldots, w_n - a_n)$, where $b_i \in g_i(s_i, a_i)$.

A network of MmA can be used in various modes. We can use it as a generative system, looking to the number of output objects that we find in the boxes (without considering the final state for the MmA). It can be used also to compute functions from $\mathbb{N}\langle V\rangle$ to $\mathbb{N}\langle O\rangle$.

An example of such a network used as a generative system could clarify these aspects:

Example 2 *Let*

$$\mathcal{N} = (V, O, \{\mathcal{A}_i\}_{i=\overline{1,3}}, \{\Lambda_i\}_{i=\overline{1,3}}, B_1)$$

where:

- $V = \{a\}$ is the input alphabet;
- $O = \{b\}$ is the output alphabet $b \neq a$;
- $\mathcal{A}_i = (\{s_i\}, V, \overline{O}, f_i, g_i, s_i)$, are the MmA's connected in the network. Their output alphabet is $\overline{O} = \{b, (a,1), (a,2), (a,3)\}$
- B_1 is the box where $\mathcal{N}$ "receives" the output multiset.
- $\Lambda_i : \mathbb{N}\langle\overline{O}\rangle \to (\mathbb{N}\langle O\rangle \cup \mathbb{N}\langle V\rangle)^3$ are the communication mappings associated to all the $\mathcal{A}_i$, $i \in T = \{1, 2, 3\}$.

We describe now the mappings. The transition mappings are

- $f_i(s_i, a) = s_i, i = \overline{1,3}$.

The output mappings:

- $g_1(s_1, a) \in \{b, (a,2) + (a,3)\}$, so is a nondeterministic mapping;
- $g_2(s_2, a) = (a, 1)$;
- $g_3(s_3, a) = (a, 1)$.

The communication mappings:

- $\Lambda_1(nb + k_1(a,1) + k_2(a,2) + k_3(a,3)) = (nb + k_1 a,\ k_2 a,\ k_3 a))$;
- $\Lambda_2(nb + k_1(a,1) + k_2(a,2) + k_3(a,3)) = (k_1 a,\ \varepsilon,\ \varepsilon))$;

- $\Lambda_3(nb + k_1(a,1) + k_2(a,2) + k_3(a,3)) = (k_1a,\ \varepsilon,\ \varepsilon)$.

Since $\mathcal{A}_1$ has a nondeterministic output mapping, the behaviour of our network is nondeterministic. We denote the global state by $s = (s_1,\ s_2,\ s_3)$. We start our computation from $(s, (a, \varepsilon, \varepsilon))$ Applying the restricted direct product we can obtain $(s, (b,\ \varepsilon,\ \varepsilon))$ or $(s, (\varepsilon, a, a))$.

In the first case we obtain one b, so we generate 1. In the second case, the computation continues with communication, and we obtain $(s, (a+a,\ \varepsilon,\ \varepsilon) = (s, (2a,\ \varepsilon,\ \varepsilon)$, and, again, we have various possibilities to choose. Anyway, it should be clear now that we can generate any number of b's, so $\mathcal{N}$ can generate every positive integer.

In order to study the computational power of nMmA, we are trying to simulate neural-like P systems. To be more specific, we try to simulate the neural P systems working in minimal mode and replicative manner. To keep the paper self-contained, we remember some facts about neural P systems and adapt the notations from [11].

3 Neural P Systems

The former tissue P systems were called neural-like P systems in [11]. We start with the classical definition, and later we adapt the notation to our needs. We consider a class of networks of membranes inspired by the way the neurons cooperate to process impulses in the complex net established by synapses. A possible model of this symbol processing machinery can be given by a network of membranes, each of them containing a multiset of objects and a state according to which the objects are processed. The membranes can communicate along "axons" channels. We make some minor modifications to the original notations, having in mind that in the Mealy multiset automata we distinguish between multisets and strings that could represent them (since we can deal with two kinds of behaviours, a global one and a sequential one). We also restrict our presentation of neural P systems working in *minimal mode and replicative manner.*

Definition 4 *A* neural P system *(nP system) of degree $m \geq 1$ is a construct*

$$\Pi = (V, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, i_{out}),$$

where

1. *V is a finite non-empty alphabet (of* objects*);*

2. *$syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ (*synapses *among cells);*

3. *$i_{out} \in \{1, 2, \ldots, m\}$ indicates the* output cell*; we can put $i_{out} = 1$;*

4. *$\sigma_1, \sigma_2, \ldots, \sigma_m$ are* cells *of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 \leq i \leq m$,*

where:

- *Q_i is a finite set (of states);*

- *R_i is a finite set of rules of the form $sw \to s'(x + y_{go} + z_{out})$, where $s, s' \in Q_i$, $w, x \in \mathbb{N}\langle V\rangle$, $y_{go} \in \mathbb{N}\langle V \times \{go\}\rangle$, $z_{out} \in \mathbb{N}\langle V \times \{out\}\rangle$, with the restriction that $z_{out} = \varepsilon$ for all i different from 1.*

The objects that appears in the left hand multiset w of the rule $sw \to s'w'$ are called *impulses*, while those from w' are called *excitations*.

Such a system is called to be *cooperative* if it contains at least one rule $sw \to s'w'$ such that $|w| > 1$, and *non-cooperative* in the opposite case.

An m-tuple of the form $(s_1w_1, s_2w_2, \ldots, s_mw_m)$ is called a *configuration* of Π. Using the rules defined above, we can define *transitions* among the configurations of the system. To this end, there are considered three modes of processing the *impulse-objects* and three modes of transmitting *excitation-objects* from one cell to another one. As we already mentioned, we restrict ourselves to the *minimal processing mode.*

Notation: $V_{go} = \{(a; go) \mid a \in V\}$, $V_{out} = \{(a; out) \mid a \in V\}$, and $V_{tot} = V \cup V_{go} \cup V_{out}$. For $s, s' \in Q_i$, $x \in \mathbb{N}\langle V\rangle$ and $y \in \mathbb{N}\langle V_{tot}\rangle$, we write $sx \Rightarrow_{min} s'y$ iff $sw \to s'w' \in R_i$, $w \subseteq x$ and $y = (x - w) \cup w'$. In this case, only one occurrence of the multiset from the left-hand side of a rule is processed, being replaced by the multiset from the right-hand of the rule, and at the same time changing the state of the cell.

We also write $sx \Rightarrow_{min} sx$ for $s \in Q_i$ and $x \in \mathbb{N}\langle V\rangle$ whenever there is no rule $sw \to s'w' \in R_i$ such that $w \subseteq x$. This encodes the case when a cell cannot process the current objects in a given state (it can be "unblocked" after receiving new impulses from the cells which are active and can send objects to it).

Now, recall that the multiset w' from a rule $sw \to s'w'$ contains symbols from V, but also symbols of the form (a, go) (or, in the case of the cell 1, of the form (a, out)). Such symbols are sent to the cells related by synapses to the cell σ_i where the rule $sw \to s'w'$ is applied, according to various manners of communication. As we already mentioned, we choose the *replicative manner*, i.e. each symbol a from (a, go) appearing in w', it is sent to each of the cells σ_j such that $(i; j) \in syn$.

In order to formally define the transition among the configurations of Π, some further notations are needed. For a multiset w over V_{tot}, we consider the projections on V, V_{go} and V_{out}, namely $pr_V(w); pr_{V_{go}}(w)$, and $pr_{V_{out}}(w)$ (see [11] for details). For a node i in the graph defined by syn, the ancestors and the successors of node i are denoted by $anc(i) = \{j \mid (j, i) \in syn\}$ and $succ(i) = \{j \mid (i, j) \in syn\}$, respectively.

Each transition lasts one time unit, and the network is synchronized: a global clock define the passage of time for all the cells.

For two configurations $C_1 = (s_1w_1, \ldots, s_mw_m)$ and $C_2 = (s'_1w''_1, \ldots, s'_mw''_m)$ we write $C_1 \Rightarrow C_2$ if there are $w'_1, \ldots, w'_m$ in $\mathbb{N}\langle V_{tot}\rangle$ such that

$$s_iw_i \Rightarrow s'_iw'_i, 1 \leq i \leq m,$$

and

$$w''_i = pr_V(w'_i) + \sum_{j \in anc(i)} pr_{V_{go}}(w'_j).$$

Obviously, objects are always sent to a cell i only from its ancestors, namely from cells j such that a direct synapse exists from j to i. In the case of the cell 1, we

remove from w_1' all the symbols $a \in V$ which appear in w_1' in the form (a, out). If during a transition a cell does nothing (no rule is applicable to the available multiset of objects in the current state), then the cell waits until new objects are sent to it from its ancestor cells.

A sequence of transitions among the configurations of Π is called a *computation* of Π. A computation ending in a configuration where no rule in no cell can be used is called a halting computation. The result of a halting computation is the number of objects in the output cell 1 (or sent to the environment from the output cell 1). We denote by $N(\Pi)$ the set of all natural numbers computed in this way by a system Π. We denote by $NOnP_{m,r}(coo)$ the family of sets $N(\Pi)$ computed by all cooperative neural-like P systems with at most $m \geq 1$ cells, each of them using at most $r \geq 1$ states. When non-cooperative systems are used, we write $NOnP_{m,r}(ncoo)$ for the corresponding family of sets $N(\Pi)$.

3.1 Computational power

We denote by NRE the family of Turing computable sets of natural numbers.

Following [11], we mention that the minimal mode of using the rules turns out to be computationally universal. If we consider the apparently weak neural-like P systems, then the fact that we obtain universality even in the non-cooperative case when using the mode *min* of applying the rules is rather unexpected. The same result holds true also when using cooperative rules. Among the results presented in [11] we mention here only those for minimal mode and for replicative manner.

Theorem 3 $NOnP_{2,5}(ncoo) = NRE$.

For the cooperative rules, the number of states can be decreased.

Theorem 4 $NOnP_{2,2}(coo) = NRE$.

4 Universality of the Networks

In order to obtain the generative power of a network of MmA, we give the following result.

Theorem 5 *Any nP System working in min mode and replicative manner can be simulated by a network of MmA (possibly nondeterministic).*

Proof. Let $\Pi = (V, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, 1)$ be an nP system with its components described as in the previous section. We remind that $\sigma_1, \sigma_2, \ldots, \sigma_m$ are cells of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i)$ $(1 \leq i \leq m)$, where Q_i is a finite set (of *states*) and R_i is a finite set of rules of the form $sw \to s'(x + y_{go} + z_{out})$ with $s, s' \in Q_i$, $w, x \in \mathbb{N}\langle V \rangle$, $y_{go} \in \mathbb{N}\langle V \times \{go\} \rangle$, $z_{out} \in \mathbb{N}\langle V \times \{out\} \rangle$ (with the restriction that $z_{out} = \varepsilon$ for all i different from 1).

We can build a nMmA $\mathcal{N} = (V, O, \{\mathcal{A}_i\}_{i=\overline{1,m}}, \{\Lambda_i\}_{i=\overline{1,m}}, B_1)$ where:

- the output alphabet O is V_{out};

- $\mathcal{A}_i = (Q_i, V, \overline{O}, f_i, g_i, s_{0,i})$ is the MmA simulating the activity of cell σ_i. The output alphabets are of the form $\overline{O} = O \cup (V \times T)$, where $T = \{1, 2, \ldots, m\}$;
- B_1 is the box where $\mathcal{N}$ collects the output multisets;
- $\Lambda_i : \mathbb{N}\langle\overline{O}\rangle \to (\mathbb{N}\langle O\rangle \cup \mathbb{N}\langle V\rangle)^n$ are the communication mappings associated to $\mathcal{A}_i$, $i \in T$.

Consider a rule $sw \to s'(x + y_{go} + z_{out})$ from R_i. We can simulate this rule with f_i and g_i by defining them in the following manner:

- $f_i(s, w) = s'$;
- $g_i(s, w) = (z_{out} + x + k_1(y, 1) + k_2(y, 2) + \ldots + k_m(y, m))$,

with the following restrictions in g_i:

- if $i \neq 1$, then $z_{out} = \varepsilon$;
- if there is no synapse from σ_i to σ_j, we define $k_j = 0$, else $k_j = 1$.

In this manner we can also simulate the replicative manner of applying the rules, since y is marked to be sent to all the cells having synapse from σ_i. It is easy to see that we have a transition $(s_1w_1, \ldots, s_mw_m) \Rightarrow (s'_1w''_1, \ldots, s'_mw''_m)$ in Π if and only if $((s_1, s_2, \ldots, s_n), (w_1, \ldots, w_n)) \vdash ((s'_1, s'_2, \ldots, s'_n), (w''_1, \ldots, w''_n))$. □

As an immediate consequence of this result we get the following

Theorem 6 *Nondeterministic networks of Mealy multiset automata are universal.*

Proof. We already know that $NOnP_{2,2}(coo) = NRE$. Applying the previous theorem we obtain that the generative power of a nondeterministic network of MmA is NRE.

Moreover, according to the proof for $NOnP_{2,2}(coo) = NRE$ ([11], page 261), the universality is obtained for a network with two Mealy multiset automata, the first one having two states, while the second one has only one state. □

Therefore a network of Mealy multiset automata is able to simulate Turing machines, and so it is computationally complete. The number of cells and states sufficient to characterize the power of Turing machines is rather small.

P systems are simulated on a cluster (network) of computers [6]. It would be interesting to see whether such an implementation can be related to the network of Mealy multiset automata.

References

[1] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, E. Shapiro. An autonomous molecular computer for logical control of gene expression. *Nature* 429:423-429, 2004.

[2] G. Ciobanu, M. Gontineac. An Automata Description of the Genetic Message Translation. *Fundamenta Informaticae*, 64:93-107, 2005.

[3] G. Ciobanu, M. Gontineac. Mealy Multiset Automata. *International Journal of Foundations of Computer Science*, 17(1):111-126, 2006.

[4] G. Ciobanu, M. Gontineac. Algebraic and Coalgebraic Aspects of Membrane Computing. In volume 3850 of *Lecture Notes in Computer Science*, pages 181-198, Springer-Verlag, 2006.

[5] G. Ciobanu, M. Gontineac. P Machines: An Automata Approach to Membrane Computing. In volume 4361 of *Lecture Notes in Computer Science*, pages 314-329, Springer-Verlag, 2006.

[6] G. Ciobanu, W. Guo. P Systems Running on a Cluster of Computers. In volume 2933 of*Lecture Notes in Computer Science*, pages 123-139, Springer-Verlag, 2004.

[7] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana. Multiset Automata. In *Multiset Processing*, volume 2235 of *Lecture Notes in Computer Science*, pages 69-83, Springer-Verlag, 2001.

[8] S. Eilenberg. *Automata, Languages and Machines*. Vol. A, Academic Press, 1976.

[9] H. de Jong. Modelling and Simulation of Genetic Regulatory Systems: A Literature Review. *Journal of Computational Biology*, 9:67-103, 2002.

[10] H. Kitano. Computational Systems Biology. *Nature*, 420:206-210, 2002.

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

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 64 - 75, 2007.

Extended Spiking Neural P Systems with Decaying Spikes and/or Total Spiking

Rudolf Freund[1], Mihai Ionescu[2], and Marion Oswald[1]

[1]Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
{rudi,marion}@emcc.at

[2] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
armandmihai.ionescu@urv.cat

Abstract

We consider extended variants of spiking neural P systems with decaying spikes (i.e., the spikes have a limited lifetime) and/or total spiking (i.e., the whole contents of a neuron is erased when it spikes). Although we use the extended model of spiking neural P systems, these restrictions of decaying spikes and/or total spiking do not allow for the generation or the acceptance of more than regular sets of natural numbers.

1 Introduction

Spiking neural P systems (in short SNP systems) are a growing research direction in the membrane computing community and have as core concept the idea of neural communication through electrical pulses called *spikes*, or *action potential*. It is known that the spikes of a given neuron all look alike, so the form of the action potential does not carry any information. But what matters is the timing and the number of spikes entering a (postsynaptic) neuron (for details on spiking neurons we refer to [5], [8] or [9]).

Initially defined in [7], SNP systems are represented as a graph with the neurons placed in the nodes of the graph. They are sending signals (spikes) along the *synapses* (the edges of the graph) if the *firing rules* inside each neuron can be activated. Hence, the structure is that of a tissue-like P system (e.g., see [4]) where the objects are all of the same form (an introduction to membrane computing can be found at [10] and an up-to-date information on this area is available online at [12]).

The functioning of an SNP system is rather simple. A global clock is assumed, and in each time unit each neuron which can use a rule should use it. The system is synchronized but it works sequentially at the level of the neurons: in every step at most one rule is used in each of them. In a generating system, one of the neurons is the designated *output neuron*, from which spikes are sent to the environment –

eventually only the difference between the first two spikes is considered to be the output number, see [7] – or else are collected as output, whereas in an analyzing (or accepting) system (*spiking neural P automaton*) the designated *input neuron* contains the initial number to be analyzed (accepted) as the number of spikes in the cell or else we take the difference between the first two input spikes as input.

Going back to neural biology, it is worth mentioning that the effect of a spike on the postsynaptic neuron can be measured, and is called the *membrane potential* (the potential difference between the interior of the cell and its surroundings). If the neuron has no electrical activity, its membrane potential is constant. After the arrival of a single spike, the potential changes and finally *decays* back to the resting potential. Hence, if no other spikes arrive in a certain amount of time in the postsynaptic neuron, the initial spike is lost (disappears) and has no further effect on the neuron.

In this paper we incorporate this phenomenon in SNP systems. We also model the *threshold* of the neuron in a different way than it was considered before. More precisely, if the threshold of a neuron is k, then if at any time during the computation the neuron has inside a number of spikes greater or equal to k the neuron must fire, and then erase its whole contents; in a more general way, we here consider *total firing* where a neuron has to empty its whole contents when firing provided its current contents belongs to a given regular set. As the underlying model in which we shall elaborate these ideas of *decaying spikes* and *total firing* we shall take extended spiking neural P systems, see [1], and we shall show that even this extended model does not allow us to go beyond regularity when using *decaying spikes* and/or *total firing.*

2 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [3] and [11]. We just list a few notions and notations: V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element. $\mathbb{N}_+$ denotes the set of positive integers, $\mathbb{N}$ is the set of non-negative integers (natural numbers), i.e., $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$. For $0 \leq k \leq m$, the interval of natural numbers between k and m is denoted by $[k..m]$. Observe that there is a one-to-one correspondence between a set $N \subseteq \mathbb{N}$ and the one-letter language $L(N) = \{a^n \mid n \in N\}$, hence, N is a regular (semilinear) set of non-negative integers if and only if $L(N)$ is a regular language; on the other hand, for a given one-letter language L the corresponding set of natural numbers $\{n \mid a^n \in L\}$ is denoted by $N(L)$. By FIN and REG we denote the families of finite sets and regular sets of natural numbers, respectively. For a finite set N, $|N|$ denotes the cardinality of N.

A deterministic finite automaton M is a construct (Q, T, δ, q_0, F) where Q is a set of states, T is a set of terminal symbols, $\delta : Q \times T \to Q$ is the transition function, q_0 is the initial state, and $F \subseteq Q$ is a set of final states; in the case of a non-deterministic finite automaton, δ is a finite subset of $Q \times T^* \times Q$. The language accepted by M is denoted by $L(M)$. The regular grammar G corresponding to the non-deterministic

finite automaton is $G = (Q, T, P, q_0)$ where P is the set of productions with $P = \{p \to wq \mid (p, w, q) \in \delta\} \cup \{p \to \lambda \mid p \in F\}$. As is well known, the family of languages accepted by (deterministic or non-deterministic) finite automata coincides with the family of regular languages and equals the family of languages generated by regular grammars; hence, the families of one-letter languages accepted by (deterministic or non-deterministic) finite automata or generated by regular grammars coincide with REG.

3 Extended Spiking Neural P Systems

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [10]; comprehensive information can be found on the P systems web page [12]. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [7]. The following definition is mainly taken from [1]:

An *extended spiking neural P system* (of degree $m \geq 1$) (in the following we shall simply speak of an *ESNP system*) is a construct

$$\Pi = (m, S, R)$$

where

- m is the number of *cells* (or *neurons*); the neurons are uniquely identified by a number between 1 and m (obviously, we could instead use an alphabet with m symbols to identify the neurons);
- S describes the *initial configuration* by assigning an initial value (of spikes) to each neuron; for the sake of simplicity, we assume that at the beginning of a computation we have no pending packages along the axons between the neurons;
- R is a finite set of *rules* of the form $\big(i, E/a^k \to P; d\big)$ such that $i \in [1..m]$ (specifying that this rule is assigned to cell i), $E \subseteq REG$ is the *checking set* (the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, d is the *delay* (the "refraction time" when neuron i performs this rule), and P is a (possibly empty) set of *productions* of the form (l, w, t) where $l \in [1..m]$ (thus specifying the target cell), $w \in \mathbb{N}$ is the *weight* of the energy sent along the axon from neuron i to neuron l, and t is the time needed before the information sent from neuron i arrives at neuron l (i.e., the *delay along the axon*).

A *configuration* of the ESNP system is described as follows:

- for each neuron, the actual number of spikes in the neuron is specified;
- in each neuron i, we may find an "activated rule" $\big(i, E/a^k \to P; d'\big)$ waiting to be executed where d' is the remaining time until the neuron spikes;

- in each axon to a neuron l, we may find pending packages of the form (l, w, t') where t' is the remaining time until w spikes have to be added to neuron l provided it is not closed for input at the time this package arrives.

A *transition* from one configuration to another one now works as follows:

- for each neuron i, we first check whether we find an "activated rule" $\big(i, E/a^k \to P; d'\big)$ waiting to be executed; if $d' = 0$, then neuron i "spikes", i.e., for every production (l, w, t) occurring in the set P we put the corresponding package (l, w, t) on the axon from neuron i to neuron l, and after that, we eliminate this "activated rule" $\big(i, E/a^k \to P; d'\big)$;

- for each neuron l, we now consider all packages (l, w, t') on axons leading to neuron l; provided the neuron is not closed, i.e., if it does not carry an activated rule $\big(i, E/a^k \to P; d'\big)$ with $d' > 0$, we then sum up all weights w in such packages where $t' = 0$ and add this sum to the corresponding number of spikes in neuron l; in any case, the packages with $t' = 0$ are eliminated from the axons, whereas for all packages with $t' > 0$, we decrement t' by one;

- for each neuron i, we now again check whether we find an "activated rule" $\big(i, E/a^k \to P; d'\big)$ (with $d' > 0$) or not; if we have not found an "activated rule", we now may apply any rule $\big(i, E/a^k \to P; d\big)$ from R for which the current number of spikes in the neuron is in E and then put a copy of this rule as "activated rule" for this neuron into the description of the current configuration; on the other hand, if there still has been an "activated rule" $\big(i, E/a^k \to P; d'\big)$ in the neuron with $d' > 0$, then we replace d' by $d' - 1$ and keep $\big(i, E/a^k \to P; d' - 1\big)$ as the "activated rule" in neuron i in the description of the configuration for the next step of the computation.

After having executed all the substeps described above in the correct sequence, we obtain the description of the new configuration. A *computation* is a sequence of configurations starting with the initial configuration given by S. A computation is called *successful* if it halts, i.e., if no pending package can be found along any axon, no neuron contains an activated rule, and for no neuron, a rule can be activated.

An ESNP is called *finite* if all the regular checking sets in the rules are finite.

In this paper, however, we will consider the following variants of the above systems:

1. *ESNP systems with decaying spikes:*

   The spikes in the system are decaying, i.e., they only have a limited lifetime before disappearing. In this case, a spike a is now written in the form (a, e), where $e \geq 1$ is the decay that, from the moment a spike (a, e) arrives in a neuron, is decremented by one in each step of the computation. As soon as $e = 0$, the corresponding spike is lost and cannot be used anymore. There could be different strategies with respect to the question which spikes should be consumed when the neuron fires, but these considerations are of no importance for the results elaborated below.

2. *ESNP systems with total spiking:*

   In this case, the whole contents of the neuron is lost as soon as a spiking rule $(i, E/ \to P; d')$ (we omit specifying the number of spikes to be consumed when applying such a rule) can be applied in neuron i as the number of spikes present in the cell is in E.

As a special case of ESNP systems with total spiking we could also consider *ESNP systems with thresholds* where the regular sets E all are of the special form $\{n \in \mathbb{N} \mid n \geq h\}$ with h being the so-called *threshold*. In this case, the rule $(i, E/ \to P; d')$ is also written as $(i, \geq h/ \to P; d')$. We postpone a thorough discussion of these restricted variants for a longer version of this paper, yet we shall use the notation in special examples for ESNP systems with total spiking.

For decaying spikes and total spiking and even a combination of these two, we will consider ESNP systems as generating as well as accepting devices, where the output (input in the case of accepting devices, respectively) is either given in a specified output (input) neuron, or else as the distance between the first two spikes exiting (entering) the system.

4 Results

As throughout this section we do not use delays in the rules and productions, we simply shall omit them to keep the description of the systems concise, e.g., instead of $\left(2, \left\{a^i\right\} / a^i \to \left\{\left(2, a^j, 0\right), (1, a, 0)\right\}; 0\right)$, in the following we shall write $\left(2, \left\{a^i\right\} / a^i \to \left\{\left(2, a^j\right), (1, a)\right\}\right)$.

First we investigate the generative power of extended spiking neural P systems with decaying spikes and/or total spiking; in the following, for generating ESNP systems we shall always assume that the output neuron contains no spiking rules; moreover, the neurons except the output neuron are called *actor neurons*:

When we consider the output to be the number of spikes at the end of a successful computation, then ESNP systems with decaying spikes can only generate finite sets, because the number of spikes that can be added in one step to the contents of a neuron is bounded, but the spikes in a neuron have a limited life-time, hence, at any moment the number of spikes in a neuron is bounded. On the other hand, every finite set of natural numbers can be generated by an extended spiking neural P system with spikes of minimal decay with only two neurons:

Example 1 *Any finite set of natural numbers N can be generated by a finite ESNP system with spikes of minimal decay with only two neurons.*

Let N be a finite set of natural numbers. We now construct the finite ESNP system Π that generates an element of N by the number of spikes contained in the output neuron 1 at the end of a successful computation:

$$\begin{aligned} \Pi &= (2, S, R), \\ S &= \{(1, \lambda), (2, (a, 1))\}, \\ R &= \left\{\left(2, \{(a, 1)\} / (a, 1) \to \left\{\left(1, (a, 1)^j\right)\right\}\right) \mid j \in N\right\}; \end{aligned}$$

after one step, every computation halts, the output neuron having received a number of spikes corresponding to a number from N. We could even add the feature of total spiking or minimal threshold 1 in order to obtain the same result; in this case, R would be written as

$$\left\{\left(2,\{(a,1)\}/\rightarrow\left\{\left(1,(a,1)^j\right)\right\}\right)\mid j\in N\right\}$$

or

$$\left\{\left(2,\geq 1/\rightarrow\left\{\left(1,(a,1)^j\right)\right\}\right)\mid j\in N\right\}.$$

For the sake of completeness, we should like to mention that the empty set is generated by the ESNP system

$$\begin{array}{lcl}\Pi_0 & = & (2,S,R_0)\,,\\ S & = & \{(1,\lambda)\,,(2,(a,1))\}\,,\\ R_0 & = & \{(2,\geq 1/\rightarrow\{(2,(a,1))\})\}\,,\end{array}$$

which only has one infinite computation.

In sum, the ESNP systems with decaying spikes (even together with total spiking) generating the result as the number of spikes in the output neuron at the end of a successful computation characterize the finite sets of natural numbers:

Theorem 2 *Any finite set of natural numbers N can be generated by a ESNP system with spikes of minimal decay with only two neurons (even with total spiking, too). On the other hand, every set of natural numbers generated in the output neuron by an ESNP system with decaying spikes (even with total spiking, too) is finite.*

If we only consider the output to be the difference between the first two spikes arriving in the output neuron during a halting computation, then we obtain a characterization of the regular sets of natural numbers even with ESNP systems with decaying spikes:

Example 3 *Let N be a regular set of natural numbers accepted by the deterministic finite automaton $M=(Q,\{a\},\delta,1,F)$ with $Q=[1..m]$. Then $L(M)$ is generated as the difference between the (first) two spikes arriving in the output neuron by the following ESNP system with decaying spikes and total spiking Π' with the output neuron 1:*

$$\begin{array}{lcl}\Pi' & = & (2,S',R')\,,\\ S' & = & \left\{(1,\lambda)\,,\left(2,(a,1)^{m+1}\right)\right\}\,,\\ R' & = & \left\{\left(2,\left\{(a,1)^i\right\}/\rightarrow\left\{\left(2,(a,1)^j\right)\right\}\right)\mid i,j\in[1..m]\,,\delta(i,a)=j\right\}\\ & \cup & \left\{\left(2,\left\{(a,1)^i\right\}/\rightarrow\{(1,(a,1))\}\right)\mid i\in[1..m]\,,i\in F\right\}\\ & \cup & \left\{\left(2,\left\{(a,1)^{m+1}\right\}/\rightarrow\{(1,(a,1))\,,(2,(a,1))\}\right)\right\}.\end{array}$$

Obviously, this system can also be interpreted as ESNP with having only one of the features decaying spikes and total spiking.

If only the restricted variant of total spiking with thresholds is used, then we need a more complicated ESNP system Π'' *(without or even with decaying spikes) where each state* i *of* M *is represented by the neuron* $i+1$*:*

$$\begin{aligned}
\Pi'' &= (m+2, S'', R''), \\
S'' &= \{(m+2,(a,1))\} \cup \{(i,\lambda) \mid i \in [1..m+1]\}, \\
R'' &= \{(i, \geq 1/ \rightarrow \{(j,(a,1))\}) \\
&\quad \mid i,j \in [2..m+1], \delta(i-1,a) = j-1\} \\
&\cup \{(i, \geq 1/ \rightarrow \{(1,(a,1))\}) \mid i \in [2..m+1], i-1 \in F\} \\
&\cup \{(m+2, \geq 1/ \rightarrow \{(1,(a,1)),(2,(a,1))\})\}.
\end{aligned}$$

In fact, the control set $\{n \geq 1\}$ *could be replaced by the finite set* $\{1\}$*, i.e.,* Π'' *corresponds to a finite system.*

A slight modification of the ESNP system Π'' *yields the ESNP system* Π''' *which generates* $L(M)$ *as the number of spikes in the output neuron even with the minimal threshold, but obviously only without decays:*

$$\begin{aligned}
\Pi''' &= (m+1, S'', R'''), \\
S'' &= \{(2,a)\} \cup \{(i,\lambda) \mid i \in \{1\} \cup [3..m+1]\}, \\
R''' &= \{(i, \geq 1/ \rightarrow \{(1,a),(j,a)\}) \\
&\quad \mid i,j \in [2..m+1], \delta(i-1,a) = j-1\} \\
&\cup \{(i, \geq 1/ \rightarrow \emptyset) \mid i \in [2..m+1], i-1 \in F\}.
\end{aligned}$$

In [1] it was shown that every ESNP system where the number of spikes remains bounded can only generate regular sets. The same arguments used to prove this result immediately show that ESNP systems with decaying spikes can only generate regular sets because the number of spikes is bounded in these cases and therefore the behaviour of the ESNP systems can be modeled by a (non-deterministic) finite automaton. Yet the same also holds true for ESNP systems with total firing:

Theorem 4 *Every language generated by an ESNP system with total firing is regular.*

Proof (sketch). Let Π be an ESNP system with total firing. Then the regular sets used in the rules of Π are of a very special form, i.e., they are a finite union of very simple sets which either are equal to $\{y\}$ or of the form $\{xn+y \mid n \in \mathbb{N}\}$ with $x, y \in \mathbb{N}$ and $x \neq 0$. Hence, it is sufficient to store the actual contents of a neuron as a vector remembering for each of these sets either the value until it exceeds y for $\{y\}$ and the module class after exceeding y for $\{xn+y \mid n \in \mathbb{N}\}$, i.e., in sum we only have a finite number of possible states of each neuron we have to consider instead of the actual values which eventually might go beyond any fixed bound.

Then the number of configurations differing in the actor neurons (i.e., the neurons except the output neuron) and the packages along the axons, but without considering the contents of the output neurons, is finite, hence, we can assign non-terminal symbols A_k to each of these configurations and take all right-regular productions $A_i \rightarrow a^k A_j$ such that k is the number of spikes added the output neuron when going from configuration i to configuration j. The initial configurations is the start

symbol of the regular grammar G constructed in that way, and, finally, for all halting configurations i we add the production $A_i \to \lambda$. In that way we can construct a regular grammar generating a one-letter language corresponding with the set of natural number generated by Π in the output neuron.

These considerations can also be taken over to the case when the output is taken as the difference between the (first) two spikes arriving in the output neuron: here we use productions of the form $A_i \to A_j$ for the periods before the first spike appears in the output neuron; afterwards we take productions $A_i \to aA_j$, i.e., for each time step in Π we generate one symbol a in a derivation in G. After the second spike has appeared in the output neuron we continue again with productions of the form $A_i \to A_j$, and as for the previous case we finish with productions $A_i \to \lambda$ for all halting configurations i. □

Hence, we can summarize these results characterizing *REG* for the generating cases as follows:

Theorem 5 *Any regular set $N \in REG$ can be generated by an ESNP system with decaying spikes and/or total firing and the output taken as the difference between the first two spikes arriving in the output neuron during a successful computation; moreover, N can also be generated by an ESNP system with total firing and the output given as the number of spikes in the output neuron at the end of a successful computation. On the other hand, every language generated by an ESNP system with total firing or by an ESNP system with decaying spikes and/or total firing and the output being the difference between the first two spikes arriving in the output neuron during a successful computation is regular.*

Now we consider the *accepting case* where the case of the input being given as the number of spikes in the input neuron (we always assume that the input neuron gets its input only from the environment) is quite trivial:

Example 6 *Let N be a regular set of natural numbers. Then N is accepted by the ESNP system with decaying spikes and/or total firing*

$$\begin{aligned} \Pi(N) &= (2, \{(1,\lambda),(2,\lambda)\}, R(N)), \\ R(N) &= \{(1, L(\mathbb{N}-N) / \to \{(2,(a,1))\}), (2, \{a\} / \to \{(2,(a,1))\})\}. \end{aligned}$$

The input n is given by $(a,1)^n$ in the first neuron which fires if and only if $n \notin N$, hence the infinite loop in the second neuron is only started in this case, whereas for $n \in N$ the system immediately halts. We should like to mention that the spike consumed by the rule $(2, \{a\} / \to \{(2,(a,1))\})$ will always be a decaying spike $(a,1)$.

Example 7 *Let N be a regular set of natural numbers accepted by the deterministic finite automaton $M = (Q, \{a\}, \delta, 1, F)$ with $Q = [1..m]$. Then $N(L(M))$ is accepted as the input being given as the difference between the first and the second spike arriving in the input neuron by the following finite ESNP system with total spiking*

even when using spikes with minimal decay:

$$
\begin{aligned}
\Pi^t &= \left(m+5, S^t, R^t\right), \\
S^t &= \{(2,(a,1))\} \cup \{(i,\lambda) \mid i \in \{1\} \cup [3..m+5]\}, \\
R^t &= \{(1,\{(a,1)\} / \to \{(m+2,(a,1)), (m+3,(a,1)), (m+4,(a,1))\})\} \\
&\cup \{(m+2,\{(a,1)\} / \to \{(m+2,(a,1)), (m+4,(a,1))\})\} \\
&\cup \left\{\left(m+2,\left\{(a,1)^2\right\} / \to \left\{\left(m+3,(a,1)^2\right),(m+5,(a,1))\right\}\right)\right\} \\
&\cup \{(m+3,\{(a,1)\} / \to \emptyset)\} \\
&\cup \left\{\left(m+3,\left\{(a,1)^2\right\} / \to \left\{\left(m+3,(a,1)^2\right)\right\}\right)\right\} \\
&\cup \left\{\left(m+3,\left\{(a,1)^3\right\} / \to \{(j,(a,1)) \mid j \in [2..m+1]\}\right)\right\} \\
&\cup \{(m+4,\{(a,1)\} / \to \emptyset)\} \\
&\cup \left\{\left(m+4,\left\{(a,1)^2\right\} / \to \{(m+5,(a,1))\}\right)\right\} \\
&\cup \{(m+5,\{(a,1)\} / \to \emptyset)\} \\
&\cup \left\{\left(m+5,\left\{(a,1)^2\right\} / \to \left\{\left(2,(a,1)^2\right)\right\}\right)\right\} \\
&\cup \left\{\left(i,\left\{(a,1)^2\right\} / \to \left\{\left(j,(a,1)^2\right)\right\}\right)\right. \\
&\quad \left. \mid i,j \in [2..m+1], \delta(i-1,a) = j-1\right\} \\
&\cup \left\{\left(i,\left\{(a,1)^3\right\} / \to \emptyset\right) \mid i \in [2..m+1], i-1 \in F\right\}.
\end{aligned}
$$

The neuron $m+2$ keeps the computation alive until the input neuron 1 spikes for the first time. This starting impulse is also propagated through neurons $m+4$ and $m+5$ to neuron 2 (which corresponds to the initial state of M). This delay through neurons $m+4$ and $m+5$ allows for starting the loop in neuron $m+3$ in time, because this loop can only be ceased by the second spike arriving in the input neuron, which then also has to be propagated to the neuron i representing the actual state $i-1$ of M and to halt the computation in Π^t by applying the rule $\left(i,\left\{(a,1)^3\right\} / \to \emptyset\right)$ provided $i-1$ is a final state in M.

Again similar arguments as in Theorem 5 can be applied when considering ESNP systems accepting a number given as this number of spikes in the input neuron or else as the difference between the (first) two spikes introduced in the input neuron from the environment:

Theorem 8 *Every set of natural numbers accepted by an ESNP system with total firing and/or decaying spikes is regular.*

Proof (sketch). Let Π be an ESNP system with total firing and/or decaying spikes. We then construct a (non-deterministic) finite automaton M accepting the regular language corresponding with the set of natural numbers accepted by $\dot{\Pi}$:

First we consider the case where the input is given as the number of spikes in the input neuron. For an ESNP system with total spiking, we first construct a finite automaton M' which analyses the given input according to the rules of Π in the input neuron, which, as already elaborated in the proof of Theorem 5 are of a very special form, i.e., they are a finite union of very simple sets which either are equal to some singleton set $\{y\}$ or of the form $\{xn+y \mid n \in \mathbb{N}\}$ with $x, y \in \mathbb{N}$ and $x \neq 0$.

Hence, it is sufficient to evaluate the contents of the input neuron to a state of M' which represents a vector remembering for each of these sets either the value until it exceeds y for $\{y\}$ and the module class after exceeding y for $\{xn + y \mid n \in \mathbb{N}\}$. According to the state of M' finally reached with the input string, we then know whether the input neuron would spike or not. M then consists of M' and M'' where M'' is constructed to start with the information from the computation in M' and then simulates the transitions in Π without taking into account the input neuron anymore. M'' only makes λ-transitions until it reaches a state corresponding to a halting configuration; exactly these states corresponding to a halting configuration are the final states of M, i.e., M halts in a final state if and only if Π halts. If we add the feature of decaying spikes, this has no influence on M', we only have to take it into account for M''. Observe that in any case, the construction of M'' again relies on the finiteness of the description of the actual contents of the neurons possible for total firing. On the other hand, if we have only decaying spikes (a, e) given in the input neuron, but no total spiking, after e steps no further information is left in the input neuron. Hence, we can apply a similar construction for a finite automaton M where we integrate the possible states of the input neuron in the possible behaviour of the whole system which remains bounded due to the fact that with decaying spikes the maximal number of spikes in the whole system remains bounded after the first e steps.

If the input is given as the difference between the first two spikes in the input neuron, then we have to construct M in three substeps: In the first step, the ESNP system works without taking into consideration the input cell. In all cases, i.e., working with decaying spikes and/or total spiking, we only get a finite set of possible states Q_I and corresponding transitions between them which exactly simulate the behaviour of the ESNP system. For every state $q \in Q_I$ we now take a state q' which only differs from q by having one spike in the input neuron; in that way we get a set Q'_I describing the configurations of Π when the first input spike arrives. Starting with Q'_I we now compute all possible configurations and the transitions between them, which yields the set Q_C. For every state $q \in Q_C$ we now take a state q' which only differs from q by having one spike in the input neuron; in that way we get a set of states Q'_C describing the configurations of Π when the second input spike arrives. The states in Q'_C are the starting point for the third and last step of the simulation, which again can be described by a finite set of states Q_O and the transitions between them. For getting M from Q_I, Q'_I, Q_C, Q'_C, and Q_O and the transitions between these states, we take the union of all these states as the set of states for M; the initial state is the state from Q_I corresponding to the initial configuration; the final states are those states from Q_O that correspond to a halting configuration; the transitions between the states in Q_I, between the states in Q_I and those in Q'_I, the transitions between the states in Q_C and those in Q'_C, as well as the transitions between the states in Q_O are λ-transitions in M, whereas a transition between the states p and q in Q_C corresponds with an a-transition (p, a, q) in M, i.e., each time step between the first and the second spike arriving in the input neuron in Π consumes one symbol a in M. This observation completes the proof. □

Hence, we can summarize the results characterizing REG for the accepting cases as

follows:

Theorem 9 *Any regular set $N \in REG$ can be accepted by an ESNP system with decaying spikes and/or total firing, the input either being given in the input neuron or else being taken as the difference between the first two spikes arriving in the input neuron during a successful computation. On the other hand, every language accepted by an ESNP system with decaying spikes and/or total firing, the input either being given in the input neuron or else being taken as the difference between the first two spikes arriving in the input neuron from the environment during a successful computation, is regular.*

5 Conclusion

In this paper, we have investigated extended spiking neural P systems with decaying spikes and/or total spiking, and we have proved that even when combining decaying spikes and/or total spiking we get a characterization of the regular sets of natural numbers with these systems being considered as generating devices or else as accepting devices (automata), except for the following cases: extended spiking neural P systems with decaying spikes (even with total spiking, too) used as generating devices with the output being given as the number of spikes in the output neuron at the end of a successful computation yield a characterization of the finite sets.

In an extended version we shall also investigate the generating and accepting power of spiking neural P systems incorporating only the original features or even more restricted variants (e.g., see [6]) as well as decaying spikes and/or total spiking. Moreover, ESNP systems with thresholds deserve further investigations. Finally, (E)SNP systems should also be considered as generators or acceptors for sets of vectors of natural numbers as well as even of string languages (e.g., compare [2]).

6 Acknowledgements

The work of Marion Oswald is supported by FWF-project T225-N04.

References

[1] A. Alhazov, R. Freund, M. Oswald, M. Slavkovik. Extended Spiking Neural P Systems Generating Strings and Vectors of Non-Negative Integers. In: H.J. Hoogeboom, Gh. Păun, G. Rozenberg, editors, *Workshop on Membrane Computing, WMC7, Leiden, The Netherlands 2006*, LNCS 4361, pages 123-134. Springer, 2007.

[2] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M. J. Pérez-Jiménez. On String Languages Generated by Spiking Neural P Systems. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F. José Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Vol. I,* RGNC REPORT 02/2006, pages 169-194. Research Group on Natural Computing, Sevilla University, Fénix Editora, Sevilla, 2006.

[3] J. Dassow, Gh. Păun. *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.

[4] R. Freund, Gh. Păun, M.J. Pérez-Jiménez. Tissue-like P systems with channel states. *Theoretical Computer Science*, 330:101–116, 2005.

[5] W. Gerstner, W. Kistler. *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.

[6] O. H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth. Normal Forms for Spiking Neural P Systems. In M. A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F. José Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Vol. II*, RGNC REPORT 02/2006, pages 105-136. Research Group on Natural Computing, Sevilla University, Fénix Editora, Sevilla, 2006.

[7] M. Ionescu, Gh. Păun, T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2–3):279–308, 2006.

[8] W. Maass. Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1):32–36, 2002.

[9] W. Maass, C. Bishop, editors. *Pulsed Neural Networks*. MIT Press, Cambridge, 1999.

[10] Gh. Păun. *Membrane Computing: An Introduction*. Springer-Verlag, Berlin, 2002.

[11] G. Rozenberg, A. Salomaa, editors. *Handbook of Formal Languages* (3 volumes). Springer-Verlag, Berlin, 1997.

[12] The P Systems Web Page, `http://psystems.disco.unimib.it`

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 76 - 89, 2007.

Principles of Transforming Communicating X-Machines to Population P Systems

Petros Kefalas[1], Ioanna Stamatopoulou[2], and Marian Gheorghe[3]

[1]CITY College, Dept. of Computer Science
Tsimiski 13, Thessaloniki 54624, Greece
kefalas@city.academic.gr

[2]South-East European Research Centre
Mitropoleos 17, Thessaloniki 54624, Greece
istamatopoulou@seerc.org

[3]University of Sheffield, Dept. of Computer Science
Regent Court, 211 Portobello Str., Sheffield S1 4DP, UK
m.gheorghe@dcs.shef.ac.uk

Abstract

Population P Systems is a class of P Systems in which cells are arranged in a graph rather than a hierarchical structure. On the other hand, Communicating X-machines are state-based machines, extended with a memory structure and transition functions instead of simple inputs, which communicate via message passing. One could use Communicating X-machines to create system models built out of components in a rather intuitive way. It is worth investigating how existing Communication X-machine models can be transformed to Population P system models so that we could take advantage of the dynamic features of the latter. In this paper, we attempt to define the principles of transforming Communicating X-machines to Population P Systems. We describe the rules that govern such transformation and we present an example in order to demonstrate the feasibility of the transformation and discuss its advantages and shortcomings.

1 Introduction

In the last years, attempts have been made to devise computational models in the form of generative devices, such as P systems [13] and its variants. These new computational paradigms have been used to solve well-known hard problems. Occasionally, some attempts also have been made to use P Systems towards modelling of swarm-based multi-agent systems [14], in order to take advantage of the reconfiguration features of P systems, such as cell death, cell division, reconfiguration of structure etc. The main problem which appears in such modelling activity is that the model resulting for the object interaction within a cell is not always easy to develop. On the other hand, state-based models provide the necessary "intuitiveness"

to model the behaviour of system components or agents. For instance, communicating X-machines have been used as a suitable paradigm of modelling agent based specification [11].

As a natural consequence of the above complementary features is to either try to combine both formalisms [17, 18] or to transform one formalism to another. The current trend in P system community research shows more interest in connecting this model with other computational approaches - Petri nets [12], process algebra [3], cellular automata [4] etc. In the past relationships between some classes of P systems and communicating X-machines have been investigated. Especially transformations of P systems into communicating stream X-machines have been particularly considered [9]. Most of these studies have been interested in translations between these models in order to make use of various strengths offered by different formalisms - model checking, for process algebra, invariants, for Petri nets, or testing methods, for X-machines.

This paper presents some principles for transforming Communicating X-machines to Population P Systems. Section 2 provides the basic background on X-machine modelling and Communicating X-Machines accompanied by an example. The definition and advantages of Population P Systems are presented in Section 3. In Section 4, we demonstrate how a transformation from one model to another is feasible and apply the guidelines to the particular example. We then discuss how the resulting model could be enhanced further to take advantage of the dynamic features of Population P Systems. We conclude by discussing the ideas behind the transformation and the issues that need further consideration.

2 State-Based Modelling with X-Machines

2.1 X-machines

X-machines (XM), a state-based formal method introduced by Eilenberg [5], are considered suitable for the formal specification of a system's components. Stream X-machines, in particular, were found to be well-suited for the modelling of reactive systems. Since then, valuable findings using the X-machines as a formal notation for specification, communication, verification and testing purposes have been reported [6, 7, 10]. An X-machine model consists of a number of states and also has a memory, which accommodates mathematically defined data structures. The transitions between states are labelled by functions. More formally, a steam X-machine is defined as the 8-tuple $(\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ and Γ are the input and output alphabets respectively;
- Q is the finite set of states;
- M is the (possibly) infinite set called memory;
- Φ is a set of partial functions φ that map an input and a memory state to an output and a possibly different memory state, $\varphi : \Sigma \times M \rightarrow \Gamma \times M$;

- F is the next state partial function, $F : Q \times \Phi \rightarrow Q$, which given a state and a function from the type Φ determines the next state. F is often described as a state transition diagram;
- q_0 and m_0 are the initial state and initial memory respectively.

For the purposes of this work we consider that the memory M is of the form $M = (m_1, \ldots, m_n)$, where each m_i is a label that refers to any arbitrary value from a domain set D_i.

2.2 Example of XM

Assume a system that consists of two XM models, i.e. one traffic light and one car. The traffic light XM has a number of states $Q = \{green, yellow, red, off\}$ and a set of inputs $\Sigma = \{tick, power\_on, power\_off\}$ representing a clock tick and the availability of electricity respectively. The output $\Gamma = \{green, red, yellow, black\}$ is the colour that the traffic light displays. The memory structure of this XM holds the display duration (in clock ticks) of each colour and a timer that counts down the ticks on each colour display. Therefore, $M = (time\_left\_to\_change, duration\_green, duration\_yellow, duration\_red)$, where $time\_left\_to\_change \in \mathbb{N}_0$, and $duration\_green, duration\_yellow, duration\_red \in \mathbb{N}$. An instance of the above model, e.g. TL_1, may have $m_0 = (20, 20, 3, 10)$ and $q_0 = green$. The state transition diagram F is depicted in Fig. 1. The set Φ consists of a number of functions, as for example:

$keep\_green(tick, (time\_left, dg, dy, dr)) = (green, (time\_left - 1, dg, dy, dr))$,
if $time\_left > 0$

$change\_yellow(tick, (0, dg, dy, dr)) = (yellow, (dy, dg, dy, dr))$

$switch\_off(power\_off, (tl, dg, dy, dr)) = (black, (tl, dg, dy, dr))$

Similarly the car model (Fig. 1) is defined as follows:

$Q = \{stopped, accelerating, cruising, breaking\}$

$M = (speed, decrease\_rate, position)$ with $speed \in \mathbb{N}_0, decrease\_rate \in \mathbb{N}$ and $position \in \{free\_road, approaching\_light(TL)\}$, where TL is the identifier of the specific traffic light the car is approaching.

$\Sigma = \{traffic\_light(TL), passed\_traffic\_light(TL), push\_break\_to\_stop, push\_break, push\_accpedal, leave\_break, leave\_accpedal\}$

and $\Gamma = \mathbb{N}_0$ having each function output the current speed of the car.

An instance of car, e.g. CAR_1, may have $m_0 = (100, 2, free\_road)$ and $q_0 = cruising$.

Indicatively some of the functions in Φ are:

$approaching\_tl(traffic\_light(TL), (speed, decrease\_rate, pos)) =$
$(speed, (speed, decrease\_rate, approaching\_light(TL)))$

$start\_breaking(push\_break, (speed, decrease\_rate, pos)) =$
$(speed/decrease\_rate, (speed/decrease\_rate, decrease\_rate, pos))$

$decrease\_speed(push\_break, (speed, dr, pos)) = (speed/dr, (speed/dr, dr, pos))$

$stop(push\_break\_to\_stop, (speed, dr, pos)) = (0, (0, dr, pos))$

$start(push\_accpedal, (0, dr, pos)) = (10, (10, dr, pos))$

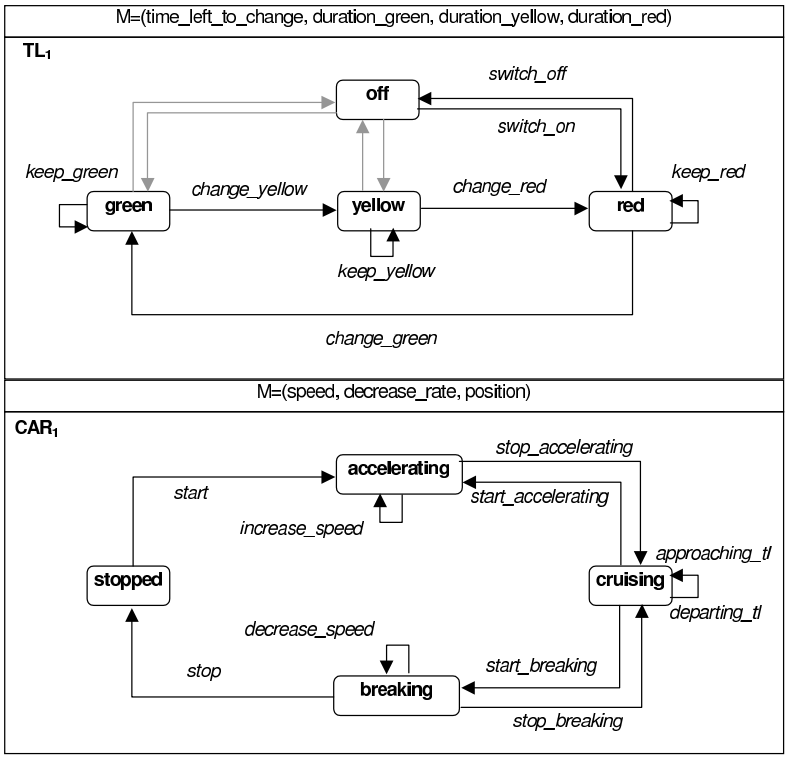


Figure 1: State Transition Diagrams for two XMs: a traffic light and a car.

2.3 Communicating X-machines

In addition to having stand-alone X-Machine models, communication is feasible by redirecting the output of one machine's function to become input to a function of another machine. The system structure of *Communicating X-machines* is defined as the graph whose nodes are the components and edges are the communication channels among them. A *Communicating X-machine System Z* is a tuple:

$$Z = ((C_i^x)_{i=1,\ldots,n},\ R)$$

where:

- C_i^x is the i-th Communicating X-machine Component, and
- R is a relation defining the communication among the components, $R \subseteq C^x \times C^x$ and $C^x = \{C_1^x, \ldots, C_n^x\}$. A tuple $(C_i^x, C_k^x) \in R$ denotes that the X-machine component C_i^x can output a message to a corresponding input stream of X-machine component C_k^x for any $i, k \in \{1, \ldots, n\}$, $i \neq k$.

A Communicating X-machine Component (CXM for short) can be derived by incorporating into an X-machine information about how it is to communicate with other X-machines that participate in the system. Exchange of messages among the components is achieved by redirecting one component's function output to be received as input by a function of another machine. In order to define the communication interface of an X-machine two things have to be stated: (a) which of its

functions receive their inputs from which machines, and (b) which of its functions send their outputs to other machines.

Graphically on the state transition diagram we denote the acceptance of input from another component by a solid circle along with the name C_i^x of the CXM that sends it. Similarly, a solid diamond with the name C_k^x denotes that output is sent to the C_k^x CXM. An abstract example of the communication between two CXMs is depicted in Fig. 2. It has to be noted that though a function φ may only read from one component at a time, it is possible that it sends its output to more than one components. A complete formal definition of Communicating X-Machines can be found in [16].

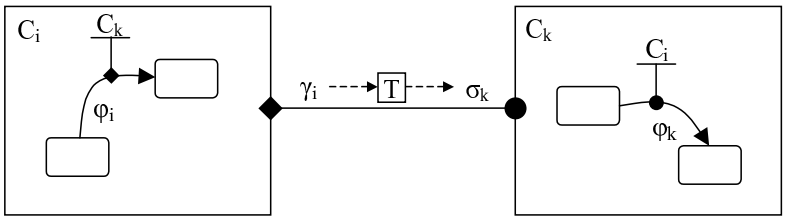


Figure 2: Abstract example of the communication between two Communicating X-machine Components.

2.4 Example of CXM

The two instances TL_1 and CAR_1 may form a communicating system as illustrated in Fig. 3. Functions of TL_1 send messages to CAR_1, through the transformation function T:
$T(change\_yellow) = push\_break$
$T(keep\_yellow) = push\_break$
$T(change\_red) = push\_break\_to\_stop$
$T(change\_green) = push\_accpedal$

Functions in CAR_1 accept those messages as inputs. The rest of the functions not annotated with receive (read) or send (write) obtain their input from the environment and send their output to the environment as normal.

3 Population P-Systems with Active Cells

A *Population P System* (PPS) [2] is a collection of different types of cells evolving according to specific rules and capable of exchanging biological / chemical substances with their neighbouring cells (Fig. 4). More formally, a PPS with active cells [2] is defined as a construct $\mathcal{P} = (V, K, \gamma, \alpha, w_E, C_1, C_2, \ldots, C_n, R)$ where:

- V is a finite alphabet of symbols called objects;
- K is a finite alphabet of symbols, which define different types of cells;

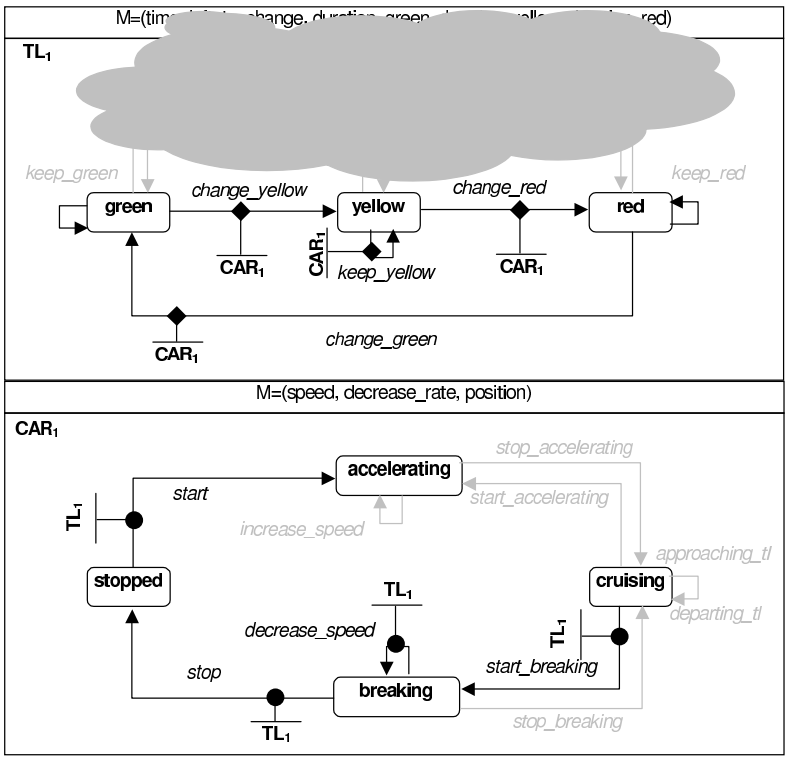


Figure 3: A CXM system consisting of one traffic light and a car.

- $\gamma = (\{1, 2, \ldots n\}, A)$, with $A \subseteq \{\{i, j\} \,|\, 1 \leq i \neq j \leq n\}$, is a finite undirected graph;
- α is a finite set of bond-making rules of the form $(t, x_1; x_2, p)$, with $x_1, x_2 \in V^*$ (multi-sets of objects represented as strings), and $t, p \in K$ meaning that in the presence of x_1 and x_2 inside two cells of type t and p respectively, a bond is created between the two cells;
- $w_E \in V^*$ is a finite multi-set of objects initially assigned to the environment;
- $C_i = (w_i, t_i)$, for each $1 \leq i \leq n$, with $w_i \in V^*$ a finite multi-set of objects, and $t_i \in K$ the type of cell i;
- R is a finite set of rules dealing with communication, object transformation, cell differentiation, cell division and cell death.

All rules present in the PPS are identified by a unique identifier, r. More particularly:

Communication rules are of the form $r : (a; b, in)_t$, $r : (a; b, enter)_t$, $r : (b, exit)_t$, for $a \in V \cup \{\lambda\}$, $b \in V$, $t \in K$, where λ is the empty string, and allow the moving of objects between neighbouring cells or a cell and the environment according to the cell type and the existing bonds among the cells. The first rule means that in the presence of an object a inside a cell of type t an object b can be obtained by

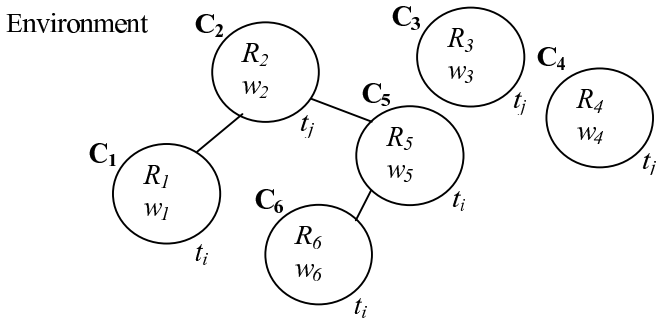


Figure 4: An abstract example of a Population P System; C_i: cells, R_i: sets of rules related to cells; w_i: multi-sets of objects associated to the cells.

a neighbouring cell non-deterministically chosen. The second rule is similar to the first with the exception that object b is not obtained by a neighbouring cell but by the environment. Lastly, the third rule denotes that if object b is present it can be expelled out to the environment.

Transformation rules are of the form $r : (\, a \rightarrow b\,)_t$, for $a \in V$, $b \in V^+$, $t \in K$, where V^+ is the set of non-empty strings over V, meaning that an object a is consumed and replaced by an object b within a cell of type t.

Cell differentiation rules are of the form $r : (\, a\,)_t \rightarrow (\, b\,)_p$, with $a, b \in V$, $t, p \in K$ meaning that consumption of an object a inside a cell of type t changes the cell, making it become of type p. All existing objects remain the same besides a which is replaced by b.

Cell division rules are of the form $r : (\, a\,)_t \rightarrow (\, b\,)_t\, (\, c\,)_t$, with $a, b, c \in V$, $t \in K$. A cell of type t containing an object a is divided into two cells of the same type. One of the new cell has a replaced by b while the other by c. All other objects of the originating cell appear in both new cells.

Cell death rules are of the form $r : (\, a\,)_t \rightarrow \dagger$, with $a \in V$, $t \in K$ meaning that an object a inside a cell of type t causes the removal of the cell from the system.

4 Transformation Principles

The question under investigation is whether some generic guidelines or principles for transforming Communicating X-machines to Population P Systems exist. We are dealing with two different methods that possess different characteristics. CXMs provide a straightforward and rather intuitive way for dealing with a component's behaviour, however, the structure of a communicating system should be known in advance and fixed throughout the computation. Additionally, CXM computation is asynchronous. On the other hand, PPS provide a straightforward way for dealing with the change of a system's structure, however, the rules specifying the behaviour of the individual cells in a PPS are of the simple form of rewrite rules which are not that intuitive to model. Finally, PPS computation is synchronous.

The rationale behind such transformation is to automatically or semi-automatically produce PPS models that can be later on enhanced with dynamic behaviour

features. This will have the advantage of using existing CXM models whose components have been thoroughly verified and tested. The resulting PPS model will have cells, objects, transformation and communication rules. The model can then be enriched with cell differentiation, death, birth and bond making rules (see next section).

4.1 Cells and types

Every CXM component will form a cell with objects, transformation and communication rules. We will refer to these cells as cells of a communicating type.

4.2 Objects in cells

We consider that all objects in the PPS are of the form $(tag : value)$, tag being a description that allows us to identify what each object represents. At least the following objects must be present in a cell to represent:

- the states in Q: objects of the form $(state : q)$, where $q \in Q$
- the memory $M = (m_1, \ldots, m_n)$: objects of the form $(m_1 : d_1) \ldots (m_n : d_n)$ where $d_1, \ldots, d_n \in D_1, \ldots, D_n$, with $D_1, \ldots, D_n$ being the *finite* domains of each memory item
- the inputs in Σ: objects of the form $(input : i)$, where $i \in \Sigma$
- the outputs in Γ: objects of the form $(output : o)$, where $o \in \Gamma$
- the messages sent/received: objects of the form $(message : (m, sender, receiver))$ where m is the actual message being sent.

4.3 Transformation rules

For every function $\varphi : \Sigma \times M \to \Gamma \times M$ such that $\varphi(\sigma, (d_1, ...d_n)) = (\gamma, (d'_1, ...d'_n))$, where $d_i, d'_i \in D_i, \sigma \in \Sigma, \gamma \in \Gamma$, for every $q, q' \in Q$ such that $q' \in F(\varphi, q)$ and for $(m_i : d_i), (m_i : d'_i)$ representing old and new values of the memory used by φ, a rule

$$\begin{aligned} \varphi \quad : \quad & ((state : q)\ (input : \sigma)\ (m_1 : d_1) \ldots (m_n : d_n) \\ \to \quad & (state : q')\ (output : \gamma)\ (m_1 : d'_1) \ldots (m_n : d'_n))_t \end{aligned}$$

is constructed.

4.4 Communication rules

For a function with communication annotations there are three different variations of conformation rules (read only, write only, both read and write). For the latter, it being the most general one, we may consider that for every function $\varphi : \Sigma \times M \to \Gamma \times M$ such that $\varphi(\sigma, (d_1, ...d_n)) = (\gamma, (d'_1, ...d'_n))$, where $d_i, d'_i \in D_i, \sigma \in \Sigma, \gamma \in \Gamma$, for every $q, q' \in Q$ such that $q' \in F(\varphi, q)$, for $(m_i : d_i), (m_i : d'_i)$ representing old and new values of the memory used by φ and for $incoming \in \Sigma$, $T(\varphi) = outgoing$ a rule

$$\begin{array}{rcl} \varphi & : & ((state : q)\ (m_i : d_i) \ldots (m_j : d_j) \\ & & (message : (incoming, sender, *this)) \\ & \rightarrow & (state : q')\ (output : \gamma)\ (m_i : d'_i) \ldots (m_j : d'_j) \\ & & (message : (outgoing', *this, receiver)))_t \end{array}$$

is constructed where $*this$ denotes the identity of the cell containing the rule. *incoming* is a message received from another cell (*sender*) therefore it is of type Σ. *outgoing* is a message to be received by another cell (*receiver*) and thus must be of the input type Σ of the receiver (this being accomplished by $T(\varphi) = outgoing$ which transforms the standard output of the function φ into something understandable for the receiver).

Every transformation rule for a function with communication annotations must also be accompanied by a communication rule that will be responsible for importing the message from a neighbouring cell (receiver) when a bond exists between them. The communication rule resides always on the receiver side and it is of the form: $cr : (\lambda; (message : (incoming, sender, *this)), in)_t$

4.5 Main result

The constructions described by the previous subsections lead to the following:

Theorem 4.1 *For any communicating X-machine working in a synchronous mode, with all sets D_i finite and a given input multi-set there is an equivalent Population P System (produces the same output as the communicating X-machine).*

4.6 Example transformation

The above example of a CXM system consisting of a traffic light TL_1 and a car CAR_1 can be transformed according to the above principles as follows:

So far, we need two types of cells, therefore $K = \{cTL, cCAR\}$ ('c' standing for 'communicating'). There will be two cells, namely $C_{TL_1} = (w_{TL_1}, cTL_{TL_1})$ and $C_{CAR_1} = (w_{CAR_1}, cCAR_{CAR_1})$.

The objects which appear during computation in cell C_{TL_1} will be:

- $(state : q)$, where $q \in \{green, yellow, red, off\}$
- $(time\_left\_to\_change : d_1), (duration\_green : d_2), (duration\_yellow : d_3)$, $(duration\_red : d_4)$ where $d_1 \in \mathbb{N}_0$, and $d_2, d_3, d_4 \in \mathbb{N}$
- $(input : i)$, where $i \in \{tick, power\_on, power\_off\}$
- $(output : o)$, where $o \in \{green, red, yellow, black\}$
- $(message : (push\_break, TL_1, CAR_1))$, $(message : (push\_break\_to\_stop, TL_1, CAR_1))$ and $(message : (push\_accpedal, TL_1, CAR_1))$.

Initially the objects w_{TL_1} are: $(state : green), (time\_left\_to\_change : 20)$, $(duration\_green : 20), (duration\_yellow : 3), (duration\_red : 10)$, which correspond to the initial state and memory values.

The transformation rules for non-communicating functions are indicatively as follows:

$$\begin{array}{lll} keep\_green & : & ((state : green)\ (input : tick)\ (time\_left\_to\_change : tl) \\ & \rightarrow & (state : green)\ (time\_left\_to\_change : tl - 1) \\ & & (output : green))_{cTL}, \text{ if } tl > 0 \\ switch\_off & : & ((state : X)\ (input : power\_off) \\ & \rightarrow & (state : off)\ (output : black))_{cTL} \end{array}$$

The objects which appear during computation in cell C_{CAR_1} will be:

- $(state : q)$, where $q \in \{stopped, accelerating, cruising, breaking\}$
- $(speed : d_1)$, $(decrease\_rate : d_2)$, $(position : d_3)$ where $d_1 \in \mathbb{N}_0$, $d_2 \in \mathbb{N}$ and $d_3 \in \{free\_road, approaching\_light(TL)\}$
- $(input : i)$, where $i \in \{traffic\_light(TL),\ passed\_traffic\_light(TL),\ push\_break,\ push\_break\_to\_stop,\ push\_accpedal,\ leave\_break,\ leave\_accpedal\}$
- $(output : o)$, where $o \in \mathbb{N}_0$ (speed).
- $(message : (push\_break, TL_1, CAR_1))$, $(message : (push\_break\_to\_stop,\ TL_1,\ CAR_1))$ and $(message : (push\_accpedal, TL_1, CAR_1))$.

Initially the objects w_{CAR_1} are: $(state : cruising), (speed : 100), (decrease\_rate : 2), (position : free\_road)$.

Indicatively a transformation rule for the corresponding non-communicating function is:

$$\begin{array}{lll} approaching\_tl & : & ((state : cruising)\ (input : traffic\_light(TL_1)) \\ & & (speed : sp)\ (position : pos) \\ & \rightarrow & (state : cruising)\ (output : sp) \\ & & (speed : sp)\ (position : traffic\_light(TL_1)))_{cCAR} \end{array}$$

As far as communication is concerned, in the cells C_{TL_1} and C_{CAR_1} there will be some transformation rules that correspond to the communicating functions. For example:

$$\begin{array}{lll} change\_yellow & : & ((state : green)\ (input : tick) \\ & & (time\_left\_to\_change : 0)\ (duration\_yellow : dy) \\ & \rightarrow & (state : yellow)\ (message : (push\_break, TL_1, CAR_1)) \\ & & (output : yellow)\ (time\_left\_to\_change : dy))_{cTL} \\ start\_breaking & : & ((state : cruising)\ (decrease\_rate : dr) \\ & & (speed : sp)\ (message : (push\_break, TL_1, CAR_1)) \\ & \rightarrow & (state : breaking)\ (output : sp/dr) \\ & & (decrease\_rate : dr))_{cCAR} \end{array}$$

In addition, cell C_{CAR_1} will have a the communication rules:

$cr_1 : (\lambda; (message : (push\_break, TL_1, CAR_1)), in)_{cCAR}$

$cr_2 : (\lambda; (message : (push\_break\_to\_stop, TL_1, CAR_1)), in)_{cCAR}$

$cr_3 : (\lambda; (message : (push\_accpedal, TL_1, CAR_1)), in)_{cCAR}$

in order to receive messages that appear in C_{TL_1}.

5 Enhancing the model

So far, a set of guidelines have been presented to transform a (static) CXM model to a (static) PPS model. One could enhance the PPS model with features that deal with a potential dynamic structure of the system. For instance:

- if the traffic light malfunctions then it should be removed from the PPS model,
- if the car leaves the traffic light, the bond between the two cells ceases to exist,
- if another car arrives, a new cell should be generated,
- if the new car approaches the traffic light, a bond should be generated, etc.

All the above issues can be dealt with by features of PPS, such as cell death, bond making rules, cell division etc. For the first example, a cell death rule such as $r : ((state : off))_{cTL} \rightarrow \dagger$ will do.

For the rest of the examples, we need to introduce another type of cell which corresponds to the non-communicating counterparts (XMs). This is because two cells that are not connected with a bond should not really have communication rules or transformation rules that correspond to communicating functions. Therefore, it is necessary to introduce two new types in K, namely $genericTL$ and $genericCAR$, which are basically equivalent to the corresponding non-communicating XMs.

So, for example, after a car passes a traffic light, a differentiation rule should change a cell from $cCAR$ type to $genericCAR$ type:

$$\begin{array}{lcl} diffrule_1 & : & ((input : passed\_traffic\_light(TL)))_{cCAR} \\ & \rightarrow & ((input : passed\_traffic\_light(TL)))_{genericCAR} \\ diffrule_2 & : & ((state : green))_{cTL} \\ & \rightarrow & ((state : green))_{genericTL} \end{array}$$

The opposite is also feasible:

$$\begin{array}{lcl} diffrule_3 & : & ((input : traffic\_light(TL)))_{genericCAR} \\ & \rightarrow & ((input : traffic\_light(TL)))_{cCAR} \\ diffrule_2 & : & ((state : yellow))_{genericTL} \\ & \rightarrow & ((state : yellow))_{cTL} \end{array}$$

A bond making rule such as:

$$(cTL, (state : yellow); (input : traffic\_light(TL)), cCAR)$$

will produce a bond between a traffic light and an approaching car.

6 Discussion and Conclusion

We presented a set of principles that guide the transformation of CXM models into PPS models. One of the motives behind this attempt lies in the fact that the resulting PPS model can be further enriched with PPS features that deal with the dynamic nature of the system's structure. There are a few more issues for discussion and further consideration.

Firstly, the objects in the environment w_E in the PPS have not been modelled. In X-machines an environment model per se does not exist. The "environment" provides the inputs in a steam and they are consumed by the functions of the machines

in a timely fashion. In a PPS, we need to consider an equivalent environment. More particularly:

- either the input objects appear in the environment during the computation, or
- input objects are generated by a generator device in an appropriate order.

In both cases, an input object is not as simple as presented in the previous sections. Instead, input objects should be of the form $(input : (\sigma, cell\_identity))$ where *cell_identity* is the cell that the input is for. An additional communication rule in both generic and communicating types of cells is required $r : (\lambda; (input : (\sigma, *this)), enter)_t$ in order for the cells to import the input from the environment. Outputs may be treated in a similar way, i.e. exported to the environment.

Secondly, the direct sending of messages between cells has not been addressed. In a CXM model, a CXM component function sends a message to another CXM component function. In a PPS, a cell cannot directly send a message but instead import a message from another cell as long as they are connected with a bond (due to the bond making rule). For two cells, as presented in the example, this does not appear to be a problem. However, if more than two cells are neighbours, then the transformation rule responsible for producing a messages should be able to produce multiple copies of it. In turn, this would mean that each cell should be aware of the identities of each of its neighbours and therefore it is implied that the identity needs to be communicated once a bond is established.

Finally, we did not deal with the different types of computation, which in CXM is asynchronous whereas in PPS is synchronous. For the specific example, this did not matter much, because the clock ticks and the connectivity of the machines imposes some kind of synchronisation in the CXM model. However, the consequences of the different types of computation in other cases should be further investigated. We anticipate that future work will deal with all these issues.

The current work will facilitate the development of algorithms to automatically translate from a specification to another one. That implies that the tools that have been developed for both methods [1, 8, 15] and their animators, could be linked together to form an integrated environment where transformations are made easy from one model to another and vice-versa.

References

[1] J. Auld, F. Romero-Campero, and M. Gheorghe. P system modelling framework. http://www.dcs.shef.ac.uk/~marian/PSimulatorWeb/P_Systems_applications.htm, November 2006.

[2] F. Bernandini and M. Gheorghe. Population P Systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.

[3] G. Ciobanu and B. Aman. On the relationship between membranes and ambients. *BioSystems*, 2007. To appear.

[4] D. Corne and P. Frisco. Dynamics of HIV infection studied with cellular automata and conformon-P systems. *BioSystems*, 2007. To appear.

[5] S. Eilenberg. *Automata, Languages and Machines*. Academic Press, 1974.

[6] G. Eleftherakis. *Formal Verification of X-machine Models: Towards Formal Development of Computer-based Systems*. PhD thesis, Department of Computer Science, University of Sheffield, 2003.

[7] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer-Verlag, London, 1998.

[8] E. Kapeti and P. Kefalas. A design language and tool for X-machines specification. In D. I. Fotiadis and S. D. Spyropoulos, editors, *Advances in Informatics*, pages 134–145. World Scientific Publishing Company, 2000.

[9] P. Kefalas, G. Eleftherakis, M. Holcombe, and M. Gheorghe. Simulation and verification of P systems through communicating X-machines. *BioSystems*, 70(2):135–148, 2003.

[10] P. Kefalas, G. Eleftherakis, and E. Kehris. Communicating X-machines: A practical approach for formal and modular specification of large systems. *Journal of Information and Software Technology*, 45(5):269–280, 2003.

[11] P. Kefalas, M. Holcombe, G. Eleftherakis, and M. Gheorghe. A formal method for the development of agent-based systems. In V. Plekhanova, editor, *Intelligent Agent Software Engineering*, pages 68–98. Idea Publishing Group Co., 2003.

[12] J. Klein and M. Koutny. Synchrony and asynchrony in membrane systems. In H. J. Hoogeboom, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 7th International Workshop, Leiden, Holland*, number 4361 in Lecture Notes in Computer Science, pages 66–85. Springer, 2007.

[13] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. Also circulated as a TUCS report since 1998.

[14] I. Stamatopoulou, M. Gheorghe, and P. Kefalas. Modelling dynamic configuration of biology-inspired multi-agent systems with Communicating X-machines and Population P Systems. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing: 5th International Workshop*, volume 3365 of *Lecture Notes in Computer Science*, pages 389–401. Springer-Verlag, Berlin, 2005.

[15] I. Stamatopoulou, P. Kefalas, G. Eleftherakis, and M. Gheorghe. A modelling language and tool for Population P Systems. In *Proceedings of the 10th Panhellenic Conference in Informatics*, Volos, Greece, November 11-13, 2005.

[16] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. Modelling the dynamic structure of biological state-based systems. *BioSystems*, 87(2-3):142–149, February 2007.

[17] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. OPERAS for space: Formal modelling of autonomous spacecrafts. In T. Papatheodorou, D. Christodoulakis, and N. Karanikolas, editors, *Current Trends in Informatics*, volume B of *Proceedings of the 11th Panhellenic Conference in Informatics (PCI'07)*, pages 69–78, Patras, Greece, May 18-20, 2007.

[18] I. Stamatopoulou, P. Kefalas, and M. Gheorghe. OPERAS$_{CC}$: An instance of a formal framework for MAS modelling based on Population P Systems. In *The 8th Workshop on Membrane Computing (WMC'07)*, pages 551–566, 2007.

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 90 - 107, 2007.

On a Characterization of Cellular Automata in Tilings of the Hyperbolic Plane

Maurice Margenstern

Laboratoire d'Informatique Théorique et Appliquée, EA 3097
Université de Metz, I.U.T. de Metz, Département d'Informatique
Île du Saulcy, 57045 Metz Cedex, France
margens@univ-metz.fr

Abstract

In this paper, we look at the extension of Hedlund's characterization of cellular automata to the case of cellular automata in the hyperbolic plane. This requires an additional condition. The new theorem is proved with full details in the case of the pentagrid and in the case of the ternary heptagrid and enough indications to show that it holds also on the grids $\{p, q\}$ of the hyperbolic plane.

1 Introduction

Hedlund's theorem, see [4] is a well known characterization of cellular automata in terms of transformation over the space of all possible configurations. The theorem says that the global transition function defined by the local rule of a cellular automaton is a continuous function on the space of all configurations of the cellular automaton and that this global function also commutes with all shifts. The theorem states that the converse is true. As a well known corollary of the theorem, we know that a cellular automaton is reversible if and only if its global transition function is bijective.

In the paper, we investigate the status of the theorem in the case of cellular automata in the hyperbolic plane. We shall prove that it is not true, *stricto-sensu*: there are cellular automata in the hyperbolic plane which do not commute with all the shifts which leave invariant the grid of the cellular automaton. In fact, we shall prove that the commutation with shifts entails another property of the cellular automation which we call **rotation invariance**. Then, denoting $\mathcal{C}$ the space of configurations for the considered grid, here the pentagrid or the ternary heptagrid. We can state:

Theorem 1 *A mapping F from $\mathcal{C}$ into $\mathcal{C}$ is the global transition function of a rotation invariant cellular automaton on the pentagrid or the ternary heptagrid if and only if F is continuous and if F commutes with all the shifts leaving the grid invariant.*

Later, we shall extend the theorem to all grids of the form $\{p, q\}$ of the hyperbolic plane. During the proof, we shall prove that the considered shifts are finitely generated: in the case of the pentagrid and of the ternary heptagrid but also, generally, for any grid $\{p, q\}$.

As we shall see, the main concern of the proof is the coordinate system for locating the cells of the cellular automaton.

This problem is obvious in the case of the Euclidean plane: in fact, whatever the grid, we may consider that we are in $\mathbb{Z}^2$ and the proof is almost word by word the same as in the unidimensional case.

In the case of the hyperbolic plane, things are very different. First, there are infinitely many tilings defined by tessellation, *i.e.* generated by the reflection of a regular polygon in its edges and, recursively, of the images in their edges. Second, there is no as general pattern as in the Euclidean plane to locate the cells of the grid. In [6], a new tool was introduced which allows to better handle the problem. It gives a general frame to locate the cells in any grid $\{p, q\}$, but the realization of the frame for each tiling $\{p, q\}$ generally depends of the tiling. There are a few exceptions. Among them we have the case of the pentagrid and of the ternary heptagrid which, up to a point, can be handled in the same way.

Just after this introduction, in the second section, we remind the reader with the system of coordinates introduced in [6], also explained in [7]. Then, in the third section, we look at the continuity part of the theorem. In the fourth section, we prove that the shifts are finitely generated, extending the result to any grid $\{p, q\}$. In the fifth section, we prove that the commutation with the shifts is equivalent to the rotation invariance. In the sixth section, we prove the theorem and its corollary about reversible cellular automata in the hyperbolic plane.

The reader is referred to [7] for an introduction to hyperbolic geometry which is aimed at the implementation of cellular automata in the corresponding spaces.

2 Coordinates in the Pentagrid and in the Heptagrid of the Hyperbolic Plane

As recalled in the introduction, the hyperbolic plane admits infinitely many tilings defined by tessellation. This is a corollary of a famous theorem proved by Henri Poincaré in the late 19^{th} century, see [7], for instance.

Figure 1 sketchily remembers that the tiling is spanned by a generating tree. Now, as indicated in figure 2, five quarters around a central tile allows us to exactly cover the hyperbolic plane with the **pentagrid** which is the tessellation obtained from the regular pentagon with right angles.

In the right-hand side picture of figure 2, we remember the basic process which defines the coordinates in a quarter of the pentagrid, see [7]. We number the nodes of the tree, starting from the root and going on, level by level and, on each level, from the left to the right. Then, we represent each number in the basis defined by the Fibonacci sequence with $f_1 = 1$, $f_2 = 2$, taking the maximal representation, see[6, 7].

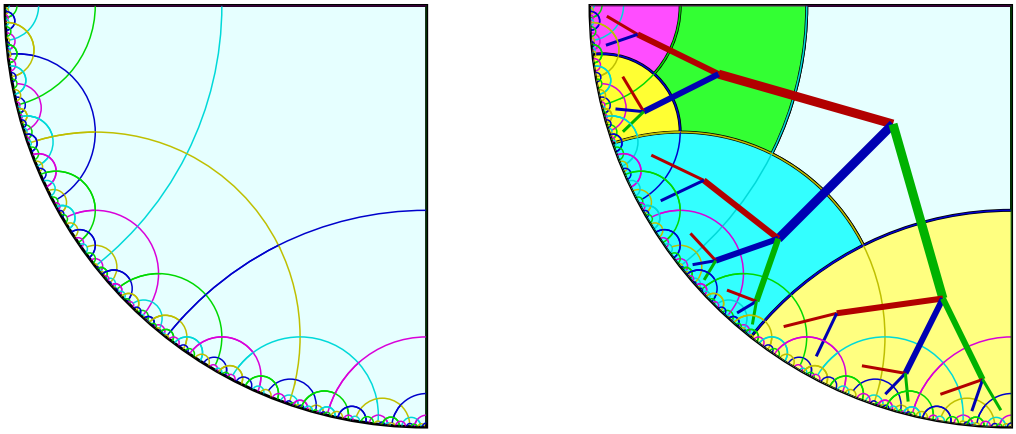


Figure 1 *On the left: the tiling; on the right: the underlying tree which spans the tiling.*

From the left-hand side picture of figure 2, we can see that any tile can be located by the indication of two numbers (i, ν), where $i \in \{1..5\}$ numbers a quarter around the central tile and ν is the number of the tile in the corresponding tree which we call a **Fibonacci tree** as the number of tiles at distance n from the root of the tree is f_{2n+1}, see [8, 6, 7].

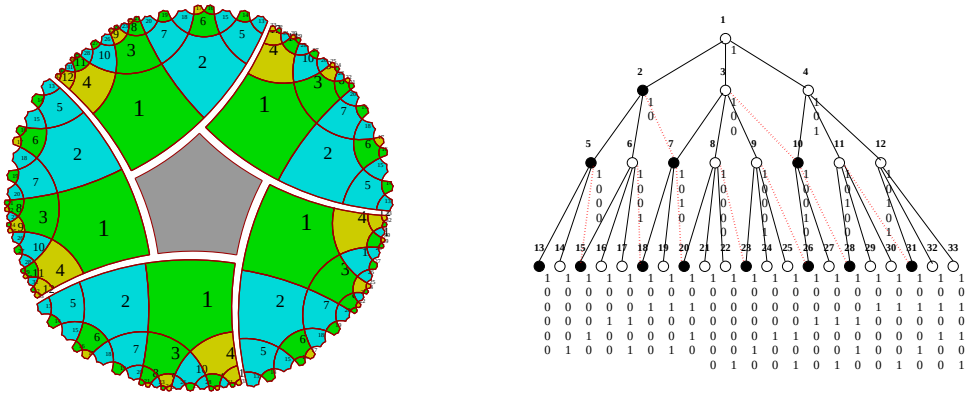


Figure 2 *On the left: five quarters around a central tile; on the right: the representations of the numbers attached to the nodes of the Fibonacci tree.*

Almost the same system of coordinates can be defined for the **ternary heptagrid** which is obtained by tessellation from a regular heptagon with the interior angle of $\frac{2\pi}{3}$, see figure 3.

Remind that the main reason of this system of coordinates is that from any cell, we can find out the coordinates of its neighbours in linear time with respect to the coordinate of the cell. Also in linear time from the coordinate of the cell, we can compute the path which goes from the central cell to the cell.

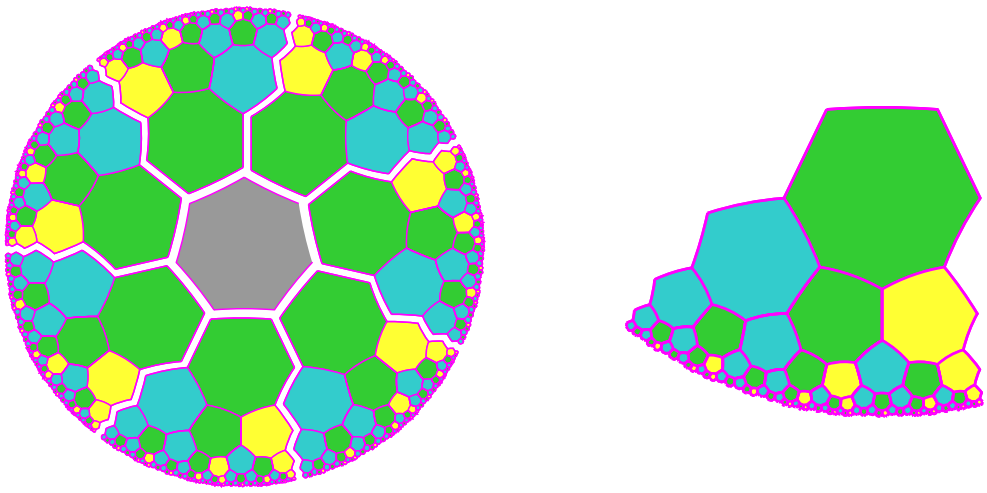


Figure 3 *On the left: seven sectors around a central tile; on the right: the structure of a sector, where a Fibonacci tree can easily be recognized.*

Now, as the system coordinate is fixed, we can turn to the space of configurations.

3 Topology on the Space of All Possible Configurations

In the proof of Hedlund's theorem, the space of configurations a cellular automaton with Q as a set of states is represented by $Q^{\mathbb{Z}^2}$. Accordingly, each configuration is viewed as a mapping from $\mathbb{Z}^2$ into Q. Now, as Q is a finite set, it is naturally endowed with the discrete topology which can be defined by a distance: $\mathrm{dist}(q_1, q_2) = 1$ if $q_1 \neq q_2$ and $\mathrm{dist}(q_1, q_2) = 0$ if $q_1 = q_2$. The space $Q^{\mathbb{Z}^2}$ is endowed with the product topology. It is the topology of the simple convergence, and it can also be defined by a distance:

$$\mathrm{dist}(x, y) = \sum_{i \in \mathbb{Z}^2} \frac{\mathrm{dist}(x(i), y(i))}{4(2|i| + 1)} 2^{-|i|},$$

where $|(\alpha, \beta)| = \max(|\alpha|, |\beta|)$. Note that $4(2n+1)$ is the length of a square centred at (0,0), exactly containing the points (α, β) with $|(\alpha, \beta)| = n$.

The translation to the case of the pentagrid or the heptagrid is immediate. Again, let Q be the set of states of the cellular automaton. We define dist on Q as previously. Now, we denote by $\mathcal{F}_5$ the set of five Fibonacci trees dispatched around a central node. Similarly, we define $\mathcal{F}_7$ for the set of seven Fibonacci trees dispatched in a similar way.

Then the distance on the set of all configurations is defined by

$$\mathrm{dist}(x, y) = \sum_{i \in \mathcal{F}_\alpha} \frac{\mathrm{dist}(x(i), y(i))}{\alpha(f_{2|i|+1})} 2^{-|i|},$$

where $\alpha \in \{5, 7\}$ and $|i|$ is defined by the distance of i to the central cell. In other terms, $|i|$ is the index of the level of the tree on which i is. We note that αf_{2n+1} is the number of nodes which are at distance n from the central cell.

It is not difficult to see that if $x(i) = y(i)$ on a ball of radius n around the central

cell, $\mathrm{dist}(x,y) \leq 2^{-n}$. Conversely, if $\mathrm{dist}(x,y) \leq \dfrac{1}{f_{2n+1}2^{-n}}$, then $x(i) = y(i)$ on a ball of radius $n-1$ around the central cell.

As well known, the set of all configurations $Q^{\mathcal{F}_\alpha}$ endowed with the just defined topology is a compact metric space.

It is plain that we have the following property:

Lemma 1 *A cellular automaton on the pentagrid or on the heptagrid is continuous on the set of all configurations with respect to the product topology.*

Indeed, as long as two configurations are equal on the neighbourhood of a cell c which corresponds to the local function of transition, the values given by the cellular automaton at c are the same for both configurations.

It is possible to extend this result to any grid $\{p,q\}$.

Remind that the restriction of the tiling to an angular sector of angle $\dfrac{2\pi}{q}$ can be spanned by a tree $\mathcal{F}_{pq}$, see [9]. Accordingly, the whole tiling can be generated by $p.(h-1)$ trees dispatched around a central tile, where $h = \lfloor \dfrac{q}{2} \rfloor$. Then, there is a bijection between the copies of the spanning trees and this tile with the tiling. Let $\mathcal{FF}_{pq}$ denote the new tree obtained by the central cell surrounded by the $p.(h-1)$ copies of $\mathcal{F}_{pq}$. We can then consider that the set of configurations of a cellular automaton A in the grid $\{p,q\}$ is $Q^{\mathcal{FF}_{pq}}$, where Q is the set of states of A.

Then, the metric of this compact metric space is defined by:

$$\mathrm{dist}(x,y) = \sum_{i \in \mathcal{FF}_{pq}} \frac{\mathrm{dist}(x(i),y(i))}{\alpha(u_i)} 2^{-|i|},$$

where u_i is the number of nodes at distance i from the root of $\mathcal{F}_{pq}$, and where $\alpha = p(h-1)$, as there are $p(h-1)$ copies of $\mathcal{F}_{pq}$ around the considered central cell. Note that the case $q = 3$ has an exceptional status, see [7].

Now, the same arguments as above for the pentagrid and for the ternary heptagrid allows us to reformulate lemma 1 as:

Lemma 2 *For all positive integers p and q with $\dfrac{1}{p} + \dfrac{1}{q} < \dfrac{1}{2}$, a cellular automaton on the grid $\{p,q\}$ of the hyperbolic plane is continuous on the set of all configurations with respect to the product topology.*

4 Generating the Shifts

First, if we analyze the proof of Hedlund's theorem, we only need the commutation with shifts to prove that a continuous mapping on the set of configurations is a cellular automaton. It is not required that the shifts constitute a group. What is needed is that for any cell c, there is a shift which transforms the origin $(0,0)$ into c. Next, if the shifts we need can be generated by finitely many fixed in advance shits, we are done, whether the shifts commute or not between themselves. If they do not commute, the representation will be more complicate, but this aspect is not relevant for our question.

The second good news is that we can find two shifts for the generation of all the shifts, both in the case of the pentagrid and of the ternary heptagrid. The proof is rather simple for the pentagrid. It is a bit more complex for the ternary heptagrid. It is a bit more difficult, also in the case of the grids $\{p, q\}$, when q is even. At last, it requires some effort in the case of the grids $\{p, q\}$, when q is odd.

In all these studies, we shall make use of the following general property:

Lemma 3 *Let τ_1 and τ_2 be two shifts along the lines ℓ_1 and ℓ_2 respectively. Then, $\tau_1 \circ \tau_2 \circ \tau_1^{-1}$ is a shift along the line $\tau_1(\ell_2)$, with the same amplitude as τ_2 and in the same direction.*

Although it is well known in the specialized literature, we provide the reader with a proof of the lemma. It relies on the following well known features on shifts in the hyperbolic plane:

(i) a shift has no fixed point in the hyperbolic plane,

(ii) there is a unique line of the hyperbolic plane, called the **axis** of the shift which is globally invariant under the action of the shift,

(iii) a shift also leaves each half-plane, defined by the complement of its axis in the plane, globally invariant,

(iv) a shift is an isometry, in particular it preserves lengths and it transforms lines into lines.

A transformation of the hyperbolic plane into itself which satisfies these three properties is a shift along its axis.

Proof of lemma 3. Consider two shifts τ_1 and τ_2, and let $\tau = \tau_1 \circ \tau_2 \circ \tau_1^{-1}$. Let δ be the axis of τ_2 and let $\delta_1 = \tau_1(\delta)$. Take a point A on the line δ and define $A_1 = \tau_1^{-1}(A)$. Clearly, if $\tau_2(A) = B$, we have $\tau(A_1) = \tau_1(B)$. Define $B_1 = \tau_1(B)$. Now, as δ is the axis of τ_2, $B \in \delta$ and so, $A_1, B_1 \in \delta_1$. Now, $\tau_1(B_1) = B$, so that $\tau(B_1) = \tau_1(C)$, where $C = \tau_2(B)$. As δ is the axis of τ_2 and as $B \in \delta$, we have also that $C \in \delta$, so that $\tau_1(C) \in \delta_1$. Now, $\tau_1(C) = \tau(B_1)$, so that $\tau(B_1) \in \delta_1$. Accordingly, A_1 and B_1 belong to δ_1 and $A_1 \neq B_1$ as $A \neq B = \tau_2(B)$ as τ_2 has no fixed point. Consequently, as τ is an isometry as a finite product of isometries, $\tau(\delta_1) \subseteq \delta_1$. And so, δ_1 is the axis of τ. Also, τ has no fixed point. Indeed, if P were a fixed point of τ, $\tau_1^{-1}(P)$ would be a fixed point of τ_2. Impossible, as τ_2 is a shift. Accordingly, as τ is a product of shifts which are positive isometries, τ is also a positive isometry: it necessarily leaves the half-planes defined by δ_1 globally invariant. And so, τ is a shift along δ_1. Now, $A_1B_1 = \tau_1^{-1}(AB) = AB$, as τ_1 is an isometry. And so the amplitude of τ, which is $A_1B_1 = A_1\tau(A_1)$, is $AB = A\tau_2(A)$, the amplitude of τ_2. □

Now, it is possible to state:

Lemma 4 *The shifts leaving the pentagrid globally invariant are generated by two shifts and their inverses. The same property holds for the ternary heptagrid.*

We shall consider the cases of the pentagrid and of the heptagrid separately. We shall make use of the traditional notation of $\tau_1 \circ \tau_2 \circ \tau_1^{-1}$ by $\tau_2^{\tau_1}$.

First, consider the case of the pentagrid, it is illustrated by the left-hand side picture of figure 4.

Fix a tile of the pentagrid, say Π_0. Fix an edge of Π_0 and let ℓ_1 be the line which supports this edge. Consider a contiguous edge, supported by the line ℓ_2. Both lines are lines of the pentagrid. Let A be the common point of ℓ_1 and ℓ_2: it is a vertex of Π_0. Let B be the other vertex of Π_0 on ℓ_1 and let C be the other vertex of Π_0 on ℓ_2. Then, define τ_1 to be the shift along ℓ_1 which transforms A into B and define τ_2 to be the shift along ℓ_2 which transforms A into C. Now, let us show that τ_1, τ_2, τ_1^{-1} and τ_2^{-1} generate all the shifts which leave the pentagrid globally invariant. It will be enough to show that if we take a tile P, there is a product of τ_1, τ_2, τ_1^{-1} and τ_2^{-1} which is a shift and which transforms Π_0 into P.

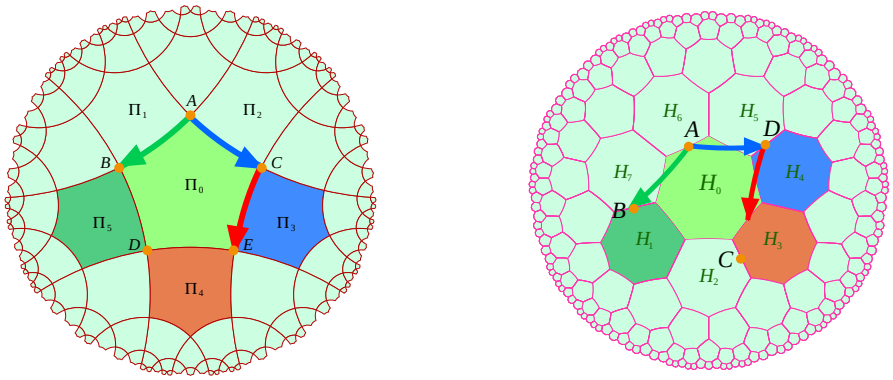


Figure 4 *Action on the shifts τ_1, green, and τ_2, blue. On the left-hand side, Π_i, $i \in \{1..5\}$ denote the neighbour of Π_0 sharing with it the edge i. Similarly, on the right-hand side, the neighbours of H_0 are denoted by H_i, $i \in \{1..7\}$. Also, note the mid-points A, B, C and D which are used by table* 1.

Number the edges of Π_0 by 1 up to 5 and assume that the edge 1 is AB and that the edge 2 is AC. Then, from lemma 3, $\tau_2^{\tau_1}$ is a shift along the edge 5, transforming B into the other end of this edge. Similarly, $\tau_1^{\tau_2}$ is the shift along the edge 3 which transforms C into the other end of this edge. Now, it is not difficult to see that $\tau_2^{\tau_1^{\tau_2}}$ is a shift along the edge 4 transforming $\tau_2^{\tau_1}(B)$ into $\tau_1^{\tau_2}(C)$. Taking these shifts and the inverses, we get shifts which transform Π_0 in all its neighbouring tiles in the sense of Moore. Now, it is not difficult to repeat this construction with any neighbour of Π_0: it shares an edge with Π_0 and it has two other edges which are supported by a line which also supports another edge of Π_0. Accordingly, we get all the tiles within a ball of radius 2 around Π_0. Now, by an easy induction, we get all the tiles of the pentagrid. Note, that for a given shift of the pentagrid, there is no unique representation of this shift as a product of powers of τ_1, τ_2 and their inverses.

Now, let us look at the case of the ternary heptagrid which is illustrated by the right-hand side picture of figure 4.

This time, we cannot take the lines which support the edges of a heptagon: due to the angle $\frac{2\pi}{3}$, such a line supports edges but it also cuts heptagons for which they are an axis of reflection. In [2, 7], I have indicated that **mid-point lines** play the rôle of the expected shifts. This is what is performed in the right-hand side picture of figure 4.

Consider again $\tau_2^{\tau_1}$, where τ_1 and τ_2 are shifts along two mid-point lines which meet on an edge of the heptagon. By construction of the mid-point lines, the definition of τ_1 and τ_2 involves the neighbours of H_0, the heptagon which we fix in order to define the generators of the shifts. As shown in the right-hand side of figure 4, τ_1 transforms H_0, say into H_1 while τ_2 transforms H_0 into H_4: we number the edges of H_0 clockwise. Now $\tau_2^{\tau_1}$ transforms H_1 into H_2, and so, it transforms H_0 into H_3. Similarly, we find that $\tau_1^{\tau_2}$ transforms H_0 into H_2.

For the convenience of the reader, we indicate the next shifts which transform H_0 into the remaining neighbours. Using the previous transformations, let us set $\xi_1 = (\tau_2^{\tau_1})^{-1}$ and $\xi_2 = (\tau_1^{\tau_2})^{-1}$. Then, ξ_1 transforms H_0 into H_7 while ξ_2 transforms H_0 into H_5. At last, $\xi_1^{\xi_2}$ transforms H_0 into H_5.

H_i	point	shift_1	shift_2
H_1	B	τ_1	$\tau_2^{\tau_1}$
H_2	C	$\tau_1^{\tau_2}$	ξ_1
H_3	C	ξ_1	ξ_2
H_4	D	ξ_2	τ_2
H_5	D	τ_2^{-1}	ξ_2
H_6	B	τ_1^{-1}	τ_2^{-1}
H_7	B	τ_1^{-1}	ξ_1

Table 1 *The shifts which, for each neighbour of H_0 generate the transformations of H_i into its neighbours. Note that there is no order in the pair of generating shifts.*

In order to reproduce the same actions for the neighbours of H_0, we just need to define mid-points of edges which will allow us to define the shifts which will play the rôle of τ_1 and τ_2 for each neighbour. The considered mid-points are indicated in the right-hand side picture of figure 4. Table 1 indicates for each neighbour the mid-point which is used and the shifts denoted in terms of the shifts which we already defined.

This allows us to prove the statement of lemma 4 for the ternary heptagrid. □

Before proving the same property of finite generation for any grid $\{p, q\}$ of the hyperbolic plane, the reader may wonder why we need two different techniques for the pentagrid and for the heptagrid? The mid-point lines can also be defined in the pentagrid and the same type of shifts, defined for the ternary heptagrid, can be defined for the pentagrid. This is true but such shifts would not be interesting for our purpose in the pentagrid. In the pentagrid, it is possible to colour the tiles with black and white in order to get something which looks like a chessboard: any white tile is surrounded by black ones and any black one is surrounded by white ones. Now, it is not difficult to remark that the shifts based on mid-point lines transform a tile of one colour into a tile of the same colour. Accordingly, we cannot get the immediate neighbours of a cell with such shifts.

As announced in our introduction, now, we prove the same property of finite

generation for any grid $\{p, q\}$ of the hyperbolic plane. From the previous remark, we may expect that the parity of q is important.

Indeed, the argument which we considered can be extended to any grid $\{p, q\}$ but, roughly speaking, the argument for the pentagrid extends to all grid $\{p, q\}$, when q is even. Similarly, the argument for the ternary heptagrid extends to all grid $\{p, q\}$, when q is odd.

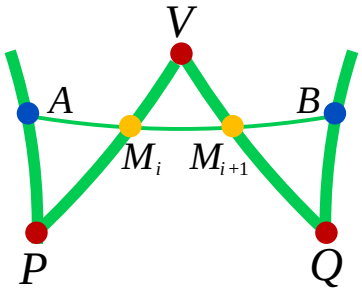


Figure 5 *The mid-point figure around a vertex, when q is odd.*

This is obvious for the grids $\{p, 4\}$ and $\{p, 3\}$. For the other grids, it follows from the following consideration. When q is bigger, number the p edges of the basic polygon P_0, $e_1, \ldots, e_p$, by turning around P_0, clockwise. Also number the vertices $V_1, \ldots, V_p$ with $V_{i+1} \in e_i \cap e_{i+1}$ for $i \in \{1..p-1\}$ and $V_1 \in e_1 \cap e_p$. Denote by τ_i the shift along the axis of e_i which transforms V_i into V_{i+1}, considering that $V_{p+1} = V_1$. Then, if we perform successively the shifts $\tau_1, \ldots, e_p$, the image of e_1 is not e_1 but its image under a rotation of $p.\frac{2\pi}{q}$ around V_1. Repeating this *tour*, we get all the tiles which are around V_1. Now, from τ_1, we go to a polygon P which is around V_2. With an appropriate number of rounds around P, we get the neighbour of P_0 which shares e_2 with it. And then, we can repeat the construction with the other edges, which provides us with all the shifts transforming P_0 into its immediate neighbours. Now, we notice that, for this construction, we need all the shifts defined by the edges of P_0. They are enough as the shifts around the sides of P are given by τ_1 and $\tau_2^{\tau_1}, \ldots, \tau_p^{\tau_1}$.

For the case when q is odd, the situation is a bit more complex. In fact, we take this time the mid-points of the edges of Q_0, the basic polygon, into consideration. Now, we consider also the mid-points of all edges of polygons which share a vertex with Q_0. Now, fix a vertex V_1 of Q_0. We consider all the mid-point of the edges which have a vertex in common with Q_0. All such mid-points which are around V_1 constitute the **mid-point figure** around V_1, see figure 5, where a partial view is given.

Let us focus on this figure. M_i and M_{i+1} are consecutive mid-points of edges which share V. The mid-point line which joins M_i and M_{i+1} also meets the line AP in P and the line BQ in Q. The line AP is an edge of a copy Q_b of Q_0 which shares V with Q_0 and which is also determined by its other edge VP. Similarly, the line BQ is also an edge of another copy Q_a of Q_0 which shares V with Q_0 and which is determined by its edge VQ. Now, the shift σ_i along the line M_iM_{i+1} which

transforms A into M_{i+1} transforms Q_b into Q_a. The opposite shift, along the same line, transforms Q_a into Q_b and, for instance, B into M_i.

By rotation around V, we determine the other shifts, constructed from two consecutive mid-point edges around V. It is not difficult to note that by applying these shifts consecutively in turning twice around the vertex, we obtain all the copies of Q_0 which are around V. Now, one of these shifts, say τ, transforms Q_0 in another neighbouring polygon Q. Note that all shifts, constructed around vertices in the above indicated way, but corresponding to Q, are obtained from those, say t, which are attached to Q_0 as t^τ. Accordingly, the shifts attached to Q_0 by the above process generate all the shifts which leave the tiling $\{p, q\}$ invariant.

And so, we proved the following extension of lemma 4:

Lemma 5 *For all positive integers p and q such that $\frac{1}{p} + \frac{1}{q} < \frac{1}{2}$, the shifts leaving the grid $\{p, q\}$ globally invariant are finitely generated. The number of generators is at most p when q is even, and at most $p.q$ when q is odd.*

5 Commutation with Shits and Rotation Invariance

First of all, we have to define what is rotation invariance and then, we shall prove that it is characterized by the commutation with shifts.

5.1 Rotation invariance

In the Euclidean plane, the definition of rotation invariant rules, a well known notion in cellular automata, can easily be defined.

Consider the case of von Neumann neighbourhood. It is not difficult to see that the rules of a cellular automaton can be represented as follows:

$$(r) \qquad s_N, s_E, s_S, s_W, s_c \to s'_c,$$

s_N, s_E, s_S and s_W are the states of the neighbours of c which are on the North, the East, the South and the West respectively. The state of c itself is s_c at the moment when the ruled is applied, and it becomes s'_c after that, which is indicated by the arrow in formula (r).

In the Euclidean case, a rotation invariant cellular automaton A is **rotation invariant** if for all rules of A written in the form of (r), the rules obtained from (r) by a circular permutation on the terms which are on the left-hand side of the arrow are also rules of A and they all give the same new state s'_c as in (r).

It is not difficult to see that such a syntactic rule can easily be transported to the case of any grid $\{p, q\}$ of the hyperbolic plane.

If we transpose the definition of the Euclidean plane to the hyperbolic one, we can see that the notion of direction plays a key rôle. As mentioned in the introduction, there is no such notion on the hyperbolic plane. The tools introduced in [6] provide us with something which plays the rôle of the North pole in the hyperbolic plane. As the basic structure of a tiling $\{p, q\}$ of the hyperbolic plane is the existence of a generating tree, for each cell, the central one excepted, the direction to the father is a way to define a direction in a meaningful way. In the case of cellular automata in

the Euclidean plane, the coordinates seems to be so an evident feature that almost nobody pays attention to that. However, if we want to **actually** implement cellular automaton for some simulation purpose, we are faced to the problem, even in this trivial case. And we can see that there is a price to pay, although the coordinate system seems to be for free. In a concrete implementation, cells have coordinates which are numbers, and numbers take some room which cannot be neglected. It could be answered that this is a hardware matter and that in a theoretical study, we may ignore this constraint. OK, let us take that granted. In this case, we can assume the same for the hyperbolic plane: fixing a central cell, the generating trees and from that the coordinates of any cell is a hardware feature.

In the next section, we shall go back to this question. We shall see that the question of direction can be, *theoretically* be handled in a pure *cellular automata* approach.

Remember that the neighbourhood of a cell c is a part of a ball around c which contains c itself. We require that the neighbourhoods N_c and N_d of two cells c and d could be put into a one-to-one correspondence by a positive displacement δ from N_c onto N_d such that $\delta(c) = d$ and $\delta(d) = f(d)$, where $f(x)$ is the father of the cell x. As we shall consider the question of rotation invariance, we assume that the neighbourhood around a cell c is a ball around c of a fixed radius k depending only on the cellular automaton. Now, as the father is known, we can number the neighbours of c by associating 1 to the father and then, clockwise turning around the cell, by associating the next numbers to the next cells at distance 1, then, in the same rotation motion, to the cells at distance 2, and then, going on in this way until we reach the last cell which is at the distance k of c. This allows us to define the **format** of a rule as follows:

$$(R) \qquad \Big(\{(\eta_i)\}_i \in \{1..\alpha u_k\}, \eta\Big) \to \eta'$$

where η_i is the state of the neighbour i of c, u_k is the number of cells in $\mathcal{F}_{pq}$ which are at distance $k-1$ from the root of $\mathcal{F}_{pq}$, and α is the number of such trees around the central cell. Note that, in particular, η_1 is the state of the father of c. Now, we remark that $1, \dots, p$ are exactly the numbers of the neighbours which are distance 1 and that a rotation on the neighbourhood of c defines a circular permutation on $\{1, ..p\}$.

Now, it is easy to notice that, conversely, a circular permutation on the numbers of the cells which are at distance 1 of c can be extended into an isometry which is nothing else than a rotation around c. If we consider a circular permutation π on $\{1, ..p\}$, this defines a rotation on the neighbourhood of c. Now, this induces a new numbering of the cells of the neighbourhood by applying the same algorithm to number the cells at a greater distance than 1 as the one we have above described. This new numbering will also be denoted by π, $\pi(i)$ being the value defined by the just defined algorithm when $i > p$. Accordingly, we can give the following definition:

Definition 1 *Consider a cellular automaton A on a grid $\{p, q\}$ of the hyperbolic plane, and assume that the neighbourhood of any cell c is a ball around c of radius k, where k is a constant. Say that A is* **rotation invariant** *if and only if for any rule of its table which can be written in the form (R), all the rules:*

$$(R') \qquad \Big(\{(\eta_{\pi(i)})\}_i \in \{1..\alpha u_k\}, \eta\Big) \to \eta'$$

where π is a circular permutation on $\{1..p\}$, extended to $\{1..\alpha u_k\}$ by the rotation induce by π, also belong to the table of A.

5.2 Commutation with shifts

Consider a cellular automaton A on the grid $\{p, q\}$ of the hyperbolic plane. Let us denote by $\mathcal{C}$ the set of configurations on the grid. We define the **global function** F_A from $\mathcal{C}$ into $\mathcal{C}$ as usual: if $x \in \mathcal{C}$, then for any cell c, we have $F_A(x)(c) = f(x(N_c), x(c))$, where N_c is the set of the neighbours of c, listed as $\{c_i\}_{i \in \{1..\alpha u_k\}}$, according to the numbering which we have above defined, and f is the table of the rules of A.

Definition 2 *Let A be a cellular automaton on the grid $\{p, q\}$ of the hyperbolic plane. Let F_A denote its global transition function. Then A is said to* **commute with the shifts** *if and only if $F_A \circ \sigma = \sigma \circ F_A$ for all shifts σ leaving the grid $\{p, q\}$ globally invariant.*

The main result of this section is:

Theorem 2 *A cellular automaton on the grid $\{p, q\}$ of the hyperbolic plane commutes with the shifts if and only if it is rotation invariant.*

Before proving the theorem, let us remark that most cellular automata which are devised for various theoretical computations are rotation invariant. This is the case for many of them in the Euclidean plane. It is also the case of several of them, among the few ones devised in the hyperbolic plane or in the $3D$ space.

Let us go back to the definition of the commutation of F_A with a shift. This means that: $F_A(\sigma(x)) = \sigma(F_A(x))$. Let $d = \sigma(c)$, where c is a cell. Then, by definition, $F_A(\sigma(x))(d) = f(\sigma(x(N_c)), s_c)$, where f is the table of A, as σ gives in d the state s_c which we have in c. Now, $\sigma(x(N_c))$ clearly transports the states of the cells in N_c onto a set of states on a rotated image of N_d with respect to the father of d. And, a priori, the father of d is **not** the image of the father of c under σ. In the next sub-section, we shall see that indeed, the shifts need not commute with the operation which, to a cell, assigns its father.

Accordingly, if the cellular automaton commutes with the shifts, it is invariant under this rotation, and conversely. Now, we know that all these rotations are generated by shifts, as it easily follows from the proof of lemma 5. Consequently, this gives us the result. □

5.3 Rotation invariant cellular automata

In this section, we shall first see that a cellular automaton on a grid $\{p, q\}$ need not commute with shifts. Then, we shall prove the following result:

Theorem 3 *For any cellular automaton A on the pentagrid or the ternary heptagrid, there is a cellular automaton B and a projection ξ of the states of B on state of A such that B is rotation-invariant and, for any cell c, $A(c) = \xi(B(c))$. There is also another cellular automaton C with a projection χ of its states on those of A satisfying $A(c) = \chi(C(c))$ and which is not rotation invariant.*

The proof of the theorem is obtained by constructing a **product** automaton with a cellular automaton which we shall define. Then, from this product, we shall construct a set of rules which is not rotation invariant and another one which is so.

The special factor of this product is a cellular automaton which propagates the tree structure inside the grid, here the pentagrid or the ternary heptagrid.

For this purpose, we assign an **extended status** to each cell which is an extension of the notion of status of this cell as a node of the Fibonacci tree where it stands. Remember that a node is **black** if it has two sons and that it is **white** if it has three sons. Black and white defines the **status** of the node, see [6]. Now, we define the **extended status** as follows, indicating them by **symbols** at the same time. First, we proceed with black nodes and then with white ones.

Bb, Bw : black node, black, white father respectively; in figure 6, below, they are represented by the colours dark and light blue, respectively.

Wwm, Wwr : white node, white father, middle, right-hand son, respectively; in figure 6, they are represented by the colours yellow and green, respectively.

Wb : white node, black father, represented in orange in figure 6.

For each node, its immediate neighbours are given by the following tables, first listing the father f of a cell c and then, clockwise turning around c, its neighbours $n_2, \ldots, n_\alpha$, with $\alpha \in \{5, 7\}$.

We can see that black nodes are always identified by the pattern Bb, Wb, Bw in their immediate neighbourhood, while white nodes are identified by the pattern Bw, Wwm, Wwr.

Now, the extended status can always be inferred from such a neighbourhood. In nodes of extended status Bb and Bw, the identification comes from the neighbour n_1 : it is white for Bb-nodes but Wwm nether occurs. For white nodes, the distinction between the extended status Wwm and the others comes from the neighbour n_4: it is Bw for Wwm nodes and Bb for the others. Between Wmr and Wb nodes, the difference comes from the father, of course.

Now, the rows of these tables can easily be transformed into conservation rules: a row $c, f, n_2, \ldots, n_\alpha$ induces the rule $f, n_2, \ldots, n_\alpha, c \to c$.

It remains to see that we can define **propagation rules** for a cellular automaton. Indeed, the initial configuration would assign a special state to the central cell and the quiescent state to all the other cells. Then, the propagation rules would define the extended status of the neighbouring cells, and defining the extended status of all cells, ring by ring, where a ring is a set of cells at the same distance from the central cell.

We give the propagation rules for such an automaton in the case of the pentagrid in figure 6, where the explanation of the rules is shortly given in the caption of the figure. We leave the propagating rules for the case of the ternary heptagrid as an exercise for the reader.

ν	f	n_1	n_2	n_3	n_4
Bb:	Bb	Wmr	Bb	Wb	Bw
	Bw	Wb	Bb	Wb	Bw
	Bw	Wmr	Bb	Wb	Bw
Bw:	Wwm	Bw	Bb	Wb	Bw
	Wwr	Wwm	Bb	Wb	Bw
	Wb	Bb	Bb	Wb	Bw
Wwm:	Wwm	Bw	Wwm	Wwr	Bw
	Wwr	Bw	Wwm	Wwr	Bw
	Wb	Bw	Wwm	Wwr	Bw
Wwr:	Wwm	Bw	Wwm	Wwr	Bb
	Wwr	Bw	Wwm	Wwr	Bb
	Wb	Bw	Wwm	Wwr	Bb
Wb:	Bb	Bw	Wwm	Wwr	Bb
	Bw	Bw	Wwm	Wwr	Bb

Table 2 *Rules for the conservation of the structure of the Fibonacci tree, case of the pentagrid.*

Now, we are in the position to prove theorem 3.

Consider the automaton P whose table is defined by the rules of figure 6 and table 2 in the case of the pentagrid. In the case of the ternary heptagrid, the propagation rules are adapted from figure 6 and also taken from table 3.

Let A a cellular automaton. We first define the product $A \times P$ by the states (η, π), where η runs over the states of A and π over those of P. We shall also say that η is the A-state of (η, π) and that π is its P-states.

Before going further, let us note that the function which associates its father to a cell does not necessarily commute with shifts.

This can easily be seen on figure 4. Consider its left-hand side picture, the case of the pentagrid. Imagine that Π_0 is a black node whose father is Π_1. Then the shift ED, which transforms E into D along the line passing through these points transforms Π_0 into its black son Π_5. Now, the same shift does not transform Π_1 into Π_0, but in the reflection of Π_1 in the line BD. On another hand, the shift BD transforms Π_1 into Π_0 and Π_0 into P_4 whose father is indeed Π_0. The same figure shows that for each kind of node and each kind of son there is a shift which maps the father onto the father in this situation and a shift which does not.

This allows us to prove the theorem. First, we notice that we can consider cells at a time when their P-state is stable. Then, we note that the rules of $A \times B$ are of the form:

(R_1) $\quad \{(\eta_i, \pi_i)\}_{i \in \{1..\alpha\}}, (\eta, \pi) \to (\eta', \pi)$

From the table 2 and 3, it is clear that rotating a rule does not entail contradictions with already established rules: the distinction between the actual father and the *rotated* one is always clear.

ν	f	n_1	n_2	n_3	n_4	n_5	n_6
Bb:	Bb	Wwr	Wwr	Bb	Wb	Bw	Wb
	Bw	Wb	Wwr	Bb	Wb	Bw	Wb
	Bw	Wwr	Wwr	Bb	Wb	Bw	Wb
Bw:	Wwm	Bw	Wb	Bb	Wb	Bw	Wwm
	Wwr	Wwm	Wwr	Bb	Wb	Bw	Wwm
	Wb	Bb	Wb	Bb	Wb	Bw	Wwm
Wwm:	Wwm	Bw	Bw	Wwm	Wwr	Bw	Wwr
	Wwr	Bw	Bw	Wwm	Wwr	Bw	Wmr
	Wb	Bw	Bw	Wwm	Wwr	Bw	Wmr
Wwr:	Wwm	Wwm	Bw	Wwm	Wwr	Bb	Bw
	Wwr	Wwm	Bw	Wwm	Wwr	Bb	Bb
	Wb	Wwm	Bw	Wwm	Wwr	Bb	Bb
Wb:	Bb	Bb	Bw	Wwm	Wwr	Bb	Bw
	Bw	Bb	Bw	Wwm	Wwr	Bb	Bw

Table 3 *Rules for the conservation of the structure of the Fibonacci tree, case of the ternary heptagrid.*

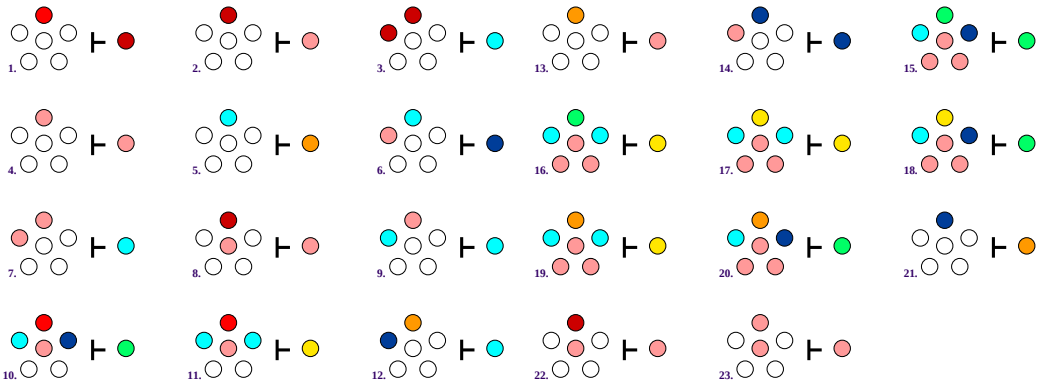


Figure 6 *Rules for the propagation of the Fibonacci tree structure in the case of the pentagrid.*

Initially, the central cell O contains a red state. By the rule 1*, it sends a dark red state to each root of a Fibonacci tree. The rules* 2 *and* 3 *allow to determine the black and white nodes of the first level of a tree which consists of the sons of the root. The rule* 3 *defines a*

black node and the rule 2 *defines a white one. The same difference later occurs on the next levels: a black node, up to now in a quiescent state, takes the state of its status when it sees two non quiescent nodes on the previous levels, namely its father and its neighbour* 2*. This is provided by the rules* 6*,* 7*,* 9*,* 12 *and* 14*. Note that when a quiescent node sees two non quiescent nodes, it recognizes its father as the right-hand side one which allows to also fix its extended status. In the other cases, the node is white, which is provided by the other rules.*

The colours of the nodes represent their extended status which indicates the status of the node **and** *the status of its father. For white nodes, it also indicates the position of their position in the list of the white sons when the father is white.*

For white nodes, they know there status at the same speed as the black nodes: a node knows as it is white as it can see only one neighbour, n_1*, in a non quiescent state. Now, the propagation of the* **extended** *status requires an additional step for the white nodes. The rules* 22 *and* 23 *introduce this delay. And so, the node remains pink while its future white sons become pink. This is why in the rules* 15 *up to* 20 *the future white sons are pink while the black sons are already installed.*

Accordingly, we can decide, either to introduce all the following rotated rules:

(R') $\quad \{(\eta_{\sigma(i)}, \pi_{\sigma(i)})\}_{i \in \{1..\alpha\}}, (\eta, \pi) \to (\eta', 0_P),$

where 0_P is the quiescent state of P and σ does perform a rotation, or all the following ones:

(R') $\quad \{(\eta_{\sigma(i)}, \pi_{\sigma(i)})\}_{i \in \{1..\alpha\}}, (\eta, \pi) \to (\eta', \pi).$

In the first case, the new automaton is not rotation invariant. In the second case, it is rotation invariant. □

As a matter of case, for the cellular automaton P itself, the rules given by figure 6 are rotation invariant, while those given by tables 2 and 3 are not. The just produced argument for the proof of theorem 3 allows us to extend the rules of tables 2 and 3 either to rotation invariant ones or to non rotation invariant ones.

6 Proving Hedlund's Theorem

Now, the proof of the theorem goes as it does classically.

From lemmas 1 and 2, we know that cellular automata on grids $\{p, q\}$ are continuous on the space of configurations. From lemma 3, we know that they commute with any shift if and only if they are rotation invariant.

For the converse, we consider a mapping F on the space of configurations. We assume that it is continuous with respect to the topology defined in section 3 and that it commutes with the shifts. Then, again, the standard argument applies. The compacity of the space with respect to the topology allows to consider the distance between two sets $\{x \mid F(x)(c) = p\}$ for different states p, as the configurations are defined on $Q^{\mathcal{F}_{pq}}$, Q being called the set of states which we assume to be finite, c being a fixed cell. This minimal distance is positive and it allows to define a ball B_n for some n such that the value of $F(x)$ at c depends only on the values of x on the ball B_n around c.

Next, as in the classical proofs, we transport this property to any cell thanks to the commutation property of F with the shifts. □

And so, we proved theorem 1. From this, we immediately get, as classically:

Theorem 4 *A rotation invariant cellular automaton on a grid* $\{p, q\}$ *of the hyperbolic plane is reversible if and only if it is bijective.*

At this point, let us note that the proof of theorem 1 is non-constructive. Mainly, the proof that a continuous mapping which commutes with the shifts is a cellular automaton is non-effective. The compactness argument indicating that the distance between the two sets of configurations giving rise to the same state is not effective. This does not allow to directly give an estimate on the size of the neighbourhood of the inverse cellular automaton. However, in the one dimensional case, the converse is obtained effectively, see [1]. Recent results, with a tight bound on the size of the inverse neighbourhood, can be found in [3].

7 Conclusion

The question arises whether other classical theorems on cellular automata are also true for hyperbolic cellular automata. As an example, we can take the theorems of Moore and Myhill, see [10, 11], characterizing surjective global transition functions as injective global transition functions restricted to finite configurations. In fact, it seems difficult to adapt the classical proof in a straightforward way.

The reason is that the classical argument relies on the fact that the surface of a big square in a square tiling of the Euclidean plane becomes negligible with respect to its all area when the size of the square tends to infinity. In the hyperbolic plane, this is no more true for a closed ball: the number of tiles on the border is more than the half of the total of number of all the tiles in the ball.

And so, there is still some work ahead: either to find another argument, or to find that Moore's or Myhill's theorem is no more true in the hyperbolic space.

Another example is the theorem about whether the reversibility of cellular automata in the hyperbolic plane is undecidable as it is in the case for the Euclidean plane, see [5].

Accordingly, there is still much work to do in these directions.

References

[1] S. Amoroso, Y. Patt. Decision Procedures for Surjectivity and Injectivity of Parallel Maps for Tessellations Structures. *Journal of Computer and System Sciences*, 6:448-464, 1972.

[2] K. Chelghoum, M. Margenstern, B. Martin, L. Pecci. Cellular automata in the hyperbolic plane: proposal for a new environment. In *Proceedings of ACRI'2004, Amsterdam, October, 25-27, 2004*, volume 3305 of *Lecture Notes in Computer Sciences*, pages 678-687. 2004.

[3] E. Czeizler, J. Kari. A tight linear bound on the synchronization delay of bijective automata. *Theoretical Computer Science*, 380(1-2):23-36, 2007.

[4] G. Hedlund. Endomorphisms and automorphisms of shift dynamical systems. *Math. Systems Theory*, 3:320-375, 1969.

[5] J. Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48:149-182, 1994.

[6] M. Margenstern. New Tools for Cellular Automata of the Hyperbolic Plane. *Journal of Universal Computer Science* 6(12):1226–1252, 2000.

[7] M. Margenstern. Cellular Automata in Hyperbolic Spaces, Volume 1, Theory, *OCP*, Philadelphia, 2007, to appear.

[8] M. Margenstern, K. Morita. NP problems are tractable in the space of cellular automata in the hyperbolic plane. *Theoretical Computer Science*, 259:99–128, 2001.

[9] M. Margenstern. About an Algorithmic Approach to Tilings $\{p, q\}$ of the Hyperbolic Plane. *Journal of Universal Computer Science*, 12(5):512-550, 2006.

[10] E.F. Moore. Machine Models of Self-reproduction. *Proceedings of the Symposium in Applied Mathematics*, 14:17-33, 1962.

[11] J. Myhill. The Converse to Moore's Garden-of-Eden Theorem. *Proceedings of the American Mathematical Society*, 14:685-686, 1963.

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 108 - 119, 2007.

Non-Determinism in Peptide Computer

M. Sakthi Balan

Department of Computer Science, The University of Western Ontario
London, Ontario, Canada, N6A 5B7
sakthi@csd.uwo.ca

Abstract

Peptide computer is a formal model for peptide computing. It involve reactions between various multiset of symbols and sequences. We study three kinds of non-determinism in peptide computer – global, locally-global and local. We show that (local) locally-global is a restrictive version of (locally-global) global. We also characterize conditions for a (locally-global) global system to be a system which is not (local) locally-global system.

1 Introduction

Peptide computing introduced by H. Hug and R. Schuler [6], takes interaction between peptides and antibodies as the basic frame work for computing. A formal model for this computing, called peptide computer, was proposed in [2]. This paper continues that study with an investigation of various kinds of non-determinism present in a peptide computer.

Peptide, a sequence of amino acids attached by covalent bonds called peptide bonds, consists of recognition sites, called *epitopes,* for the antibodies. A peptide can contain more than one epitope for the same or different antibodies. With each antibody, which attaches to a specific epitope, a binding power is associated, called its *affinity.* When antibodies compete for recognition sites – which may overlap in the given peptide – then the antibodies with greater affinity have higher priority. For further information regarding the bio-chemical processes themselves we refer to, for example, [5]. Dynamic global computing models for the immune system are presented in [7, 9].

Peptide computing refers to computational processes based on the elementary operations such as binding of antibodies to peptide sequences and removal of antibodies from peptide sequences.

In [6] it was shown how to solve the satisfiability problem using peptide computing and in the subsequent paper [3] it was shown to solve two further NP-complete problems – *Hamiltonian circuit* and *exact cover by 3-set*. Moreover in [3], a simulation of Turing machine by peptide computing is presented to show peptide computing is computationally complete. Towards formalizing the peptide computing model in a rigorous way a formal model called as peptide computer was proposed in [1, 2]. Peptide computer defines the notion of a step and it was also shown in [2] that it

can be simulated by a Turing machine under some conditions. A survey on peptide computing depicting the state-of-the-art at its time is presented in [4].

A peptide computer, informally, consists of some symbols and sequences and the processing take place through reactions between symbols and sequences or between sequences. The presence of many symbols and sequences together with their multiple occurrences (multiset) brings in non-determinism to the system. Hence non-determinism can be studied in two ways: one, due to interactions between multiple occurrence of a sequence or a symbol and the other one where non-determinism happens when interactions occur between different sequences and different symbols. Non-determinism is an essential one for unconventional computing like peptide computing. Hence a study of non-determinism existing in a peptide computer would help us to understand how processing take place.

Our paper is organized as follows: in the following section we give some preliminary on peptide computer and introduce the basic notations that are necessary for the paper. In Section 3 we define three kinds of non-determinism in peptide computer and in Section 4 we study some of the properties of non-determinism. The paper concludes with some remarks in Section 5.

2 Preliminaries

For a set S, $|S|$ denotes the cardinality of S. When S is a singleton set, $S = \{x\}$ say, we often omit the set brackets, that is, we write x instead of $\{x\}$. For sets S and T, consider a relation $\varrho \subseteq S \times T$. Then ϱ^{-1} is the relation $\varrho^{-1} = \{(t, s) \mid (s, t) \in \varrho\}$ and, for $s \in S$, $\varrho(s) = \{t \mid (s, t) \in \varrho\}$. We use the notation $\varrho : S \xrightarrow{\circ} T$ to denote a partial mapping of S into T. In that case $\mathsf{dom}\,\varrho$ is the subset of S on which ϱ is defined. The notation $\varrho : S \to T$ means that ϱ is a total mapping of S into T, hence $\mathsf{dom}\,\varrho = S$ in this case. In addition to the standard symbols for operations on sets, we use the symbol $\uplus$ to denote disjoint union.

Let S be a non-empty set. A *multiset* on S is a pair $M = (I, \iota)$ where I is a set, the *index set,* and ι is a mapping of I into S, the *index mapping.* A multiset M is non-empty, if I is non-empty; it is finite if I is finite. For $s \in S$, the number $\mathsf{mult}(s) = |\{i \mid i \in I, \iota(i) = s\}|$ is the *multiplicity* of s. When I is countable, we write $M = \{m_i \mid i \in I\}$ where $m_i = \iota(i)$ is implied. With this notation, it is possible that $m_i = m_j$ while $i \neq j$ for $i, j \in I$. We use the standard symbols for set theoretic operations also for multisets. However, on multisets, union is disjoint union and both intersection and difference take multiplicities into account. Formally this can be handled by appropriate operations on the index sets.

Multisets as defined above are also called *families* in the literature. The usual definition of a multiset as a set $\{(s, \mathsf{mult}(s)) \mid s \in S\}$ of pairs is adequate only when all multiplicities are finite.

By $\mathbb{N}$ and $\mathbb{N}_0$ we denote the sets of positive integers and of non-negative integers, respectively. The set $\mathbb{B} = \{0, 1\}$ represents the set of Boolean values. For $n \in \mathbb{N}_0$, the ordinal number $\mathfrak{n}$ is represented by the set $\mathfrak{n} = \{i \mid i \in \mathbb{N}_0, i < n\}$. Thus, for example, $\mathfrak{o} = \emptyset$, $\mathfrak{1} = \{0\}$ and, in general, $\mathfrak{n} = \{0, 1, \ldots, n-1\}$. By $\mathbb{R}$ we denote the set of real numbers, and $\mathbb{R}_+ = \{r \mid r \in \mathbb{R}, r \geq 0\}$.

An alphabet is a non-empty set. Let X be an alphabet. Then X^* is the set of all words over X including the empty word λ, and $X^+ = X^* \setminus \{\lambda\}$. For a word $w \in X^*$, $|w|$ is its length. Any word $u \in X^*$ with $w \in uX^*$ is a prefix of w; let $\mathsf{Pref}(w)$ be the set of prefixes of w; the words in $\mathsf{Pref}_+(w) = \{u \mid u \in X^+, w \in uX^+\}$ are the proper prefixes of w. Similarly, a word $u \in X^*$ with $w \in X^*uX^*$ is an infix of w, $\mathsf{Inf}(w)$ is the set of infixes of w and $\mathsf{Inf}_+(w) = \{u \mid u \in X^+, u \in \mathsf{Inf}(w), u \neq w\}$ is the set of proper infixes of w. A language over X is a subset of X^*. For a language L over X and $Y \in \{\mathsf{Pref}, \mathsf{Pref}_+, \mathsf{Inf}, \mathsf{Inf}_+\}$, $Y(L) = \bigcup_{w \in L} Y(w)$.

Let L be a language over X and $w \in X^*$. An *L-decomposition* of w is a pair of sequences $(u_0, u_1, \ldots, u_k)$, $(v_0, v_1, \ldots, v_{k-1})$ of words in X^* such that $u_0v_0u_1v_1 \cdots v_{k-1}u_k = w$, $v_0, v_1, \ldots, v_{k-1} \in L$ and $u_0, u_1, \ldots, u_k \notin X^*LX^*$. A language in X^+ such that every word has a unique L-decomposition is called a *solid code* [8]. Consider $w \in X^+$ of length n, say $w = x_0x_1 \cdots x_{n-1}$ with $x_i \in X$ for $i = 0, 1, \ldots, n-1$. An L-decomposition of w as above can be specified by a set of pairs $\{(i_l, j_l) \mid l = 0, 1, \ldots, k-1\}$ such that, for $l = 0, 1, \ldots, k-1$, $v_l = x_{i_l}x_{i_l+1} \cdots x_{j_l}$. Let $\partial_L(w)$ be the set of L-decompositions when represented in this way. Let $\mathcal{D}(L) = \{(w, d) \mid w \in X^*, d \in \partial_L(w)\}$ be the set of words together with all their L-decompositions.

Now we give a brief description of peptide computer presented in [2].

Definition 1 *A* peptide computer *is a quintuple $\mathcal{P} = (X, E, A, \alpha, \beta)$ where X is a finite alphabet (to represent basic building units like molecules), $E \subseteq X^+$ is a language (to represent epitopes), A is a countable alphabet with $A \cap X^* = \emptyset$ (to represent antibodies), $\alpha \subseteq E \times A$ is a relation (such that $a \in \alpha(e)$ means that antibody a can be attached to epitope e), $\beta : E \times A \to \mathbb{R}_+$ is a mapping such that $\beta(e, a) > 0$ if and only if $(e, a) \in \alpha$ (denoting the affinity between epitope e and antibody a).*

Consider a word $w \in X^+$ and $d \in \partial_E(w)$. An *A-attachment* is a partial mapping $\tau : d \overset{\circ}{\to} A$. Suppose $w = x_0x_1 \cdots x_n$ and $d = \{(i_l, j_l) \mid l = 0, 1, \ldots, k-1\}$. Then τ defines a word $w_\tau \in (X \cup (E \times A))^*$ as follows: For all $l = 0, 1, \ldots, k-1$, if $(i_l, j_l) \in \mathsf{dom}\,\tau$ replace $e = x_{i_l}x_{i_l+1} \cdots x_{j_l}$ by $(e, \tau(i_l, j_l))$ in w. Such a mapping τ is *legal* if $(e, \tau(i_l, j_l)) \in \alpha$ for all l. When τ is legal then $w_\tau \in (X \cup \alpha)^*$ and τ is called an *A-attachment to w*. For a language $L \subseteq X^+$, let $\mathcal{T}(L)$ be the set of A-attachments to words in L. Conversely, a word $z \in (X \cup \alpha)^*$ defines a word $w \in X^*$ and a set of A-attachments τ, such that $w_\tau = z$. Note that w is uniquely defined, but that τ may apply to several $d \in \partial_E w$.

Consider a word $z \in (X \cup \alpha)^+$ and a symbol $a \in A$. Let w and τ be such that $w_\tau = z$. Moreover, let $w = x_0x_1 \cdots x_n$ with $x_0, x_1, \ldots, x_n \in X$. Consider any $d \in \partial_E w$ with $\mathsf{dom}\,\tau \subseteq d$ and any $d' \in \partial_E w$. For $(i, j) \in d'$ let $e_{i,j} = x_ix_{i+1} \cdots x_j$. We say that *$a$ dominates (i, j)* in z when the following condition is satisfied: For all $(i', j') \in d$ such that $\{i', i'+1, \ldots, j'\} \cap \{i, i+1, \ldots, j\} \neq \emptyset$ and $(i', j') \in \mathsf{dom}\,\tau$,

$$\beta(e_{i,j}, a) > \beta(x_{i'}x_{i'+1} \cdots x_{j'}, \tau(i', j')).$$

In such a case, all pairs $(i', j') \in d$ with $\{i', i'+1, \ldots, j'\} \cap \{i, i+1, \ldots, j\} \neq \emptyset$ are said to be *affected*. If a dominates (i, j) in z, the following *basic reaction will happen* forming a multiset $R(z, a)$: For each affected pair (i', j'), a copy of $\tau(i'j')$ is

put into $R(z,a)$; let $Y \subseteq \mathsf{dom}\,\tau$ be the set of pairs which are not affected and let $d'' \in \partial_E w$ be such that $Y \cup (i,j) \subseteq d''$. Define the A-attachment $\bar{\tau} : d'' \xrightarrow{\circ} A$ by $\bar{\tau}(p) = \tau(p)$ for $p \in Y$ and $\bar{\tau}(i,j) = a$. Put a copy of $w_{\bar{\tau}}$ into $R(z,a)$. The multiset $R(z,a)$ is the *result of a basic reaction* between z and a. If a is binding with z and some symbols are released from z when $R(z,a)$ is formed then we denote the set of released symbols by $Out(z,a)$. If nothing is released when a binds then $Out(z,a)$ will be $\{\lambda\}$.

We also need to consider *basic reactions* between words $z, z' \in (x \cup \alpha)^+$, where z and z' need not be different. Again we want to define the resulting multiset $R(z,z')$. We use w, d and τ as above. Now $z' = w'_{\tau'}$ where $\tau' : d' \xrightarrow{\circ} A$ for some $d' \in \partial_E w'$. Consider $(i',j') \in \mathsf{dom}\,\tau'$ and let $a = \tau'(i'j')$. Moreover, let $e'_{i',j'}$ be the infix of w' which starts at i' and ends at j'. Suppose a dominates (i,j) in z for some $(i,j) \in \bar{d} \in \partial_E w$ and $\beta(e_{i,j}, a) > \beta(e'_{i',j'}, a)$, then the reaction is as follows.

Since the basic reactions between two words z and z' are with respect to a, we represent these by $R_a(z,z')$. They take place in two steps:

1. The reaction $\mathsf{Sep}R_a(z,z')$ produces a multiset containing z, z'' and a, where z'' is defined as follows: let τ'' be the restriction of τ' to $\mathsf{dom}\,\tau' \setminus (i'j')$; then $z'' = w'_{\tau''}$.

2. Next is the reaction leading to $R(z,a)$.

As before $Out(z_1,z_2)$ denotes the set of symbols released from z_1 when a binds with z_1. When z and z' are the same occurrence of a word then $\mathsf{Sep}R_a(z,z')$ consists only of z'' and a.

The basic reactions resulting in $R(z,a)$ and $R_a(z,z')$ take place only when there is instability. Instability between z and a occurs when a dominates $(i,j) \in \partial_E w$ where $z = w_\tau$. Likewise instability between two words z and z' occurs when there is a symbol $a = \tau'(i',j')$ where $(i',j') \in \mathsf{dom}\,\tau'$ and $\tau' : d' \xrightarrow{\circ} A$ for some $d' \in \partial_E(w')$.

A basic reaction can trigger a sequence of reactions; this might even lead to a cycle which in turn will not result in a stable configuration. In the sequel we refer to $R(z,a)$ and $R_a(z_1,z_2)$ as the results of a basic reaction or as multisets, whichever is more appropriate in the context.

Definition 2 *Let $\mathcal{P}$ be a peptide computer. A* peptide configuration *is a finite multiset of words in $(X \cup \alpha)^+ \cup A$.*

To a peptide configuration P a basic reaction may apply when instability exists in the configuration, that is, there may be $z, z' \in (X \cup \alpha)^+$ or $a \in A$ which occur in P such that $R(z,a)$ differs from the multiset consisting of z and a or $R(z,z')$ differs from the multiset consisting of z and z'. In either case a basic reaction non-deterministically removes (z,a) or (z,z') from P and adds $R(z,a)$ or $R(z,z')$, respectively. Let $R(P)$ be the set of peptide configurations which result from P through one basic reaction. For $n \in \mathbb{N}_0$, let R^n be the n-fold iteration of R.

Definition 3 *A peptide configuration P is said to be* stable *if $R(P) = \{P\}$.*

If $R^n(P)$ consists of stable configurations only, for some n, define $R^*(P) = R^n(P)$ for this n. Otherwise, $R^*(P) = \emptyset$. Let Γ be the class of stable peptide configurations.

To define peptide computations, we also need the following objects:

Definition 4 *A* peptide instruction *has the form $+P$ or $-P$ where P is a peptide configuration.*

When P' is a peptide configuration and I is a peptide instruction then

$$I(P') = \begin{cases} P' \cup P, & \text{if } I = +P, \\ P' \setminus P, & \text{if } I = -P, \end{cases}$$

with union and difference taken as multiset operations.

The instruction $-P$ is called a flushing instruction if $P = P' \cap A$; hence in this case all the symbols in A which are not binding to any sequence in X^+ are removed from the configuration.

Definition 5 *A* peptide program *is a pair $(\mathfrak{P}, \chi)$ where $\mathfrak{P}$ is a mapping from Γ^* into the set of peptide instructions and χ is a (halting) function $\chi : \Gamma \to \mathbb{B}$.*

Definition 6 *Let $\mathcal{P}$ be a peptide computing model and let $(\mathfrak{P}, \chi)$ be a peptide program for $\mathcal{P}$. A* peptide computation *is a word $c = c_0c_1 \cdots c_t \in \Gamma^*$ with $c_0, c_1, \ldots, c_t \in \Gamma$ such that*

$$c_i \in R^*(\mathfrak{P}(c_0c_1 \cdots c_{i-1})(c_{i-1}))$$

for $i = 0, 1, \ldots, c_t$.

A computation as above starts with $c_0 \in R^*(\mathfrak{P}(\lambda))$ and ends when $\chi(c_i) = 1$ for the first time.

To encode inputs we need a mapping γ from inputs to Γ, an input encoding; we also need an output decoding, that is, a mapping δ from Γ to outputs.

Definition 7 *A function f from inputs to outputs is* peptide computable *if there is a peptide program $\mathfrak{P}$, a computable input encoding γ of inputs into $\mathfrak{P}(\lambda)$ and a computable decoding of Γ into outputs such that, for every $x \in \mathsf{dom}\, f$, there is a peptide computation $c_0c_1 \cdots c_t$ with $c_0, c_1, \ldots, c_t \in \Gamma$ and $\gamma(x) = c_0$ satisfying $\chi(c_t) = 1$ and $\delta(c_t) = f(x)$.*

3 Non-Determinism in Peptide Computer

As mentioned briefly in the introduction non-determinism comes into picture in peptide computer in two ways. First one is, many-to-many interactions between symbols and sequences and the other is, due to multiplicities of the sequences and symbols. But when we look upon a multiset as a pair $M = (I, \iota)$ we can visualize both the non-determinism as a single one since M can be virtually thought of as a set but holding all the original properties of multiset. With this important note we proceed to define non-determinism in peptide computer.

We define non-determinism in peptide computer in three levels: global, locally-global and local. We first describe all three of them very informally in the sequel.

Informally speaking there are two ways a non-determinism can occur in a peptide computer: one, when there are more than one epitopes defined for a symbol and two, when there are more than one symbols in competition to bind to their epitopes which overlap with each other. The global definition is similar to the definition of non-determinism in accepting devices like finite state automata, pushdown automata and so on. It is defined in a more generic way on the system as a whole.

The other two local definitions are more interesting ones. Basically when reaction take place we have a peptide configuration that contains all the sequences and symbols participating in the reaction. The locally-global non-determinism is defined by restricting the global definition of non-determinism to the sequences and the symbols present in the configuration.

The third one is defined on the reactions taking place between symbols and a sequence or between two sequences. As defined earlier, these reactions happen only when there is an instability in the medium. When instability is present in the medium and either of the following conditions are satisfied: there is a possibility of more than one epitopes for a symbol to bind or there are more than one symbol that can attach to an epitope, then we say that it is locally non-deterministic.

All the above definitions will be explained more in detail when we define them formally.

It can be seen that the local definitions are more restrictive versions of the global definition. We prove it formally in the next section. There can be many instances that even though the definition might be non-deterministic globally it might not be non-deterministic when reaction occurs. We will present under what conditions this happens.

The reason behind defining non-determinism in three levels is explained in the sequel. The peptide computer is a generic system used to solve a set of problems. Hence it can be defined as a non-deterministic or deterministic one. To solve a problem by peptide computer we write a peptide program to work with the available sequences and symbols. Even when the generic system is a non-deterministic one, when writing a program for a problem there might not be a need for non-determinism. Hence we have separated out the non-determinism as global and locally-global ones. Similarly even when a program uses non-determinism when actual processing take place due to the structure of the sequence and the binding properties of the symbols there might not be any non-determinism present when reactions take place. These arguments show that we have to separate non-determinism into three levels.

Before defining non-determinism formally, we define few technical terms which we use in the paper later.

Definition 8 *For a peptide configuration P we say a sequence $z \in (X \cup \alpha)^+$ as a* participating sequence *if $z \in P$. Likewise a symbol a is a* participating symbol *if $a \in P$.*

The set of all participating sequences is denoted by *Pseq* and the set of all participating symbols is denoted by *Psym*. A participating sequence is denoted by *pseq* and a participating symbol by *psym*.

Definition 9 *For a peptide configuration P, an epitope $e \in E$ is said to be a* participating epitope *if $e \in Sub(w)$ where $w_\tau = z$ is a pseq and τ is an arbitrary $A-$attachment.*

The set of all participating epitopes is denoted by *Pepi*. A participating epitope is denoted by *pepi*.

We recall that only when the configuration attains stability the next peptide instruction is applied. When a peptide instruction is carried out by peptide computer, the configuration becomes instable and reactions occur to attain stability. Now we have the following definition:

Definition 10 *The time period between applying a peptide instruction and attaining stability is defined as the* instability period. *The series of reactions happening in the instability period are collectively called as a* step.

Definition 11 *For a sequence $x \in V^+$ represented as $x = x_1x_2\cdots x_n$ where $x_i \in V$, any epitope e in x is of the form $x_i\cdots x_j$ where $i \leq j \leq n$.*

For any two epitopes e, e' in x with $e = x_i\cdots x_j$ and $e' = y_k\cdots y_l$ where $i \leq j \leq n$ and $k \leq l \leq n$ we say e and e' overlap when either $i \leq k \leq j$ or $k \leq i \leq l$.

Now we present formal definitions for three types of non-determinism in peptide computer.

Definition 12 *A* peptide computer $\mathcal{P} = (X, E, A, \alpha, \beta)$ *is said to be* globally non-deterministic *if either of the following conditions are satisfied:*

- *there exists a symbol $a \in A$ and epitopes $e_1, e_2, \cdots, e_n, n \geq 2$ such that $\{(e_1, a), (e_2, a), \cdots, (e_n, a)\} \subseteq \alpha$*
- *there are symbols $a_1, a_2, \cdots, a_m, m \geq 2$ and there exists epitopes $e_1, e_2, \cdots, e_p$, $p \geq 1$ such that*
  - *each e_j overlaps with each other e_i where $1 \leq i \neq j \leq p$, and*
  - *for each a_i there exists at least one j such that $(e_j, a_i) \in \alpha$ where $1 \leq i \leq m$ and $1 \leq j \leq p$.*

Definition 13 *A* peptide computer $\mathcal{P} = (X, E, A, \alpha, \beta)$ *is said to be* locally-global non-deterministic *if either of the following conditions are true for all peptide configuration P:*

- *there exists a symbol $a \in Psym$ and epitopes $e_1, e_2, \cdots, e_n, n \geq 2, e_i \in Pepi$, $1 \leq i \leq n$ such that $\{(e_1, a), (e_2, a), \cdots, (e_n, a)\} \subseteq \alpha$*
- *there are symbols $a_1, a_2, \cdots, a_m, m \geq 2, a_i \in Psym, 1 \leq i \leq m$ and there exists epitopes $e_1, e_2, \cdots, e_p, p \geq 1, e_j \in Pepi, 1 \leq j \leq p$ such that*
  - *each e_j overlaps with each other e_k where $1 \leq j \neq k \leq p$, and*
  - *for each a_i there exists at least one l such that $(e_l, a_i) \in \alpha$ where $1 \leq l \leq p$ and $1 \leq i \leq m$.*

Before defining locally non-deterministic we recall that a step in peptide computer consists of a sequence of set of reactions $R^1(P), R^2(P), \cdots\cdots$ where P is the configuration of the peptide computer. At each i, $R^i(P)$ consists of reactions between z and a and reactions between two sequences z_1 and z_2. Now we define local non-determinism:

Definition 14 *A step in peptide computer is said to be non-deterministic if there exists a $m \geq 1$ such that $R^m(P)$ satisfies either of the following conditions:*

- *if $a \in Psym$ dominates more than one pair, say the set of pairs $\{(i_1, j_1), (i_2, j_2), \cdots, (i_n, j_n)\}$*
- *if there are symbols $a_1, a_2, \cdots, a_m, m \geq 2, a_i \in Psym, 1 \leq i \leq m$ and there exists epitopes $e_1, e_2, \cdots, e_p, e_i \in Pepi, 1 \leq i \leq p$ such that*
  - *for each a_i there exists at least one j such that $(e_j, a_i) \in \alpha$ and a_i dominates e_j, and*
  - *each e_j overlaps with each other e_i.*

In all the three definitions presented above we note that there are two different views of non-determinism – one, with respect to a symbol which can non-deterministically bind to one of its epitopes and the other one with respect to epitopes, where non-deterministically one of many symbols can bind to it. We note here that epitopes for those symbols need not be the same epitope but a set of epitopes with the property that epitopes overlap with each other.

Definition 15 *A symbol $a \in A$ is a* non-deterministic symbol *if $\mid \alpha(a) \mid > 1$.*

The set of all non-deterministic symbols in A is denoted by A_{nd}. In the following definitions we classify the set of epitopes as non-deterministic and deterministic epitopes.

Definition 16 *A set $F \subseteq E$ is said to be* overlapping set *if every two epitopes $e_i, e_j \in F$ overlap.*

By definition it should be obvious to note that every singleton set of E is an overlapping set. For any sequence x over V^* if any epitope in x is associated with the overlapping set F then we denote it as x_F.

Definition 17 *An overlapping set $F = \{f_1, f_2, \cdots, f_m\}, m \geq 1$ is said to be a* non-deterministic epitope-set *if there is a set $A' \subseteq A$, say $A' = \{a_1, a_2, \cdots, a_n\}$ where $n \geq 2$ such that for all $a_i \in A'$ there is at least one j satisfying the condition that $(e_j, a_i) \in \alpha$.*

The set of all non-deterministic epitope set is denoted by $\mathcal{E}_{nd}$. If $\mathcal{E}_{nd} = \{E_1, E_2, \cdots, E_n\}$ and $F \subseteq E$ then we define $\mathcal{E}_{nd}^F$ as the family of set $\{E_1 \cap F, E_2 \cap F, \cdots E_n \cap F\}$.

Definition 18 *Let F be an overlapping set and $e \in F$. The* weight *of F is defined as $\beta(e, a)$ where (e, a) is a subsequence of x_F.*

We note that the above definition is not ambiguous since all epitopes in F overlap and so at any instance only one epitope will be bounded by a symbol. We denote weight of F as $w(F)$.

Definition 19 *Let $e \in E$. We say e is* closed *if any overlapping set F containing e has a non-zero weight. If all overlapping sets containing E are zero weight then it is said to be* open.

Using the definitions closed and open we define a characteristic function $\chi : E \longrightarrow \{0,1\}$ with $\chi(e) = 0$ if e is closed and $\chi(e) = 1$ if e is open.

Definition 20 *Let $X = \{x_1, x_2, \cdots, x_n\}, n \geq 1$ be a set. We define tuple(X) as a n-tuple $(x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ where $(i_1, i_2, \cdots, i_n)$ is any permutation of the set $\{1, 2, \cdots, n\}$. We extend the definition of χ to tuples over any subset of E, say $(e_1, e_2, \cdots, e_k)$, as $(\chi(e_1), \chi(e_2), \cdots, \chi(e_n))$.*

Definition 21 *A peptide computer is said to be strictly globally non-deterministic if it is globally non-deterministic but not locally-global non-deterministic. Likewise a peptide computer is said to be strictly locally-global non-deterministic if it is locally-global non-deterministic but not locally non-deterministic.*

We use the following notations in our paper. The set of all peptide computer $\mathcal{P}$ is denoted by PC. The set of all peptide computers which are globally non-deterministic is denoted by PC_{gnd}. Likewise peptide computers which are locally-global non-deterministic (locally non-deterministic) is denoted by PC_{lgnd} (PC_{lgd}).

4 Results on Non-Determinism

Theorem 1

1. $PC_{lgnd} \subseteq PC_{gnd}$,
2. $PC_{lnd} \subseteq PC_{lgnd}$.

Proof. Let $\mathcal{P} \in PC_{lgnd}$. We prove $\mathcal{P}$ is also a globally non-deterministic one. This simply follows from definition. Since $Psym \subseteq A$ and $Pepi \subseteq E$, it directly follows that if condition (1) of Definition 13 is true then condition (1) of Definition 12 is true, or if the condition (2) of Definition 13 is true then condition (2) of Definition 12 is true. This shows that $\mathcal{P} \in PC_{gnd}$.

Let $\mathcal{P}' \in PC_{lnd}$. We will show that $\mathcal{P}' \in PC_{lgnd}$. Our assumption implies that either the condition (1) or condition (2) of Definition 14 is satisfied. If condition (1) is true then $a \in Psym$ dominates more than one pair, i.e., the set of pairs $\{(i_1, j_1), (i_2, j_2), \cdots (i_n, j_n)\}$ where $n \geq 2$. By the definition of a symbol dominating a sequence it implies that for the symbol $a \in Psym$, $\{(e_1, a), (e_2, a), \cdots, (e_n, a)\} \subseteq \alpha$ where if (i_k, j_k) is a pair from the sequence $x^k \in Pseq$ then $e_k = x^k_{i_k} x^k_{i_k+1} \cdots x^k_{j_k}, 1 \leq k \leq n$. This proves condition (1) of Definition 13 is true.

If condition (2) of Definition 14 is satisfied, then there are symbols $a_1, a_2, \cdots, a_m$, $m \geq 2, a_i \in Psym, 1 \leq i \leq m$ and there exists epitopes $e_1, e_2, \cdots, e_p, p \geq 1, e_i \in$

$Pepi, 1 \leq i \leq p$ such that for each a_i there exists at least one j such that $(e_j, a_i) \in \alpha$. This shows condition (2) of Definition 13 is satisfied.

Hence in either case we have $\mathcal{P} \in PC_{lgnd}$. □

Now we study some properties of peptide computer which will help us to exhibit the conditions for a peptide computer to be strictly globally-non-deterministic. Similarly we also examine under what conditions peptide computer is strictly locally-global non-deterministic one.

Theorem 2 *A peptide computer $\mathcal{P}$ is strictly globally non-deterministic if it satisfies either of the following conditions: for all i,*

- *P_i contains no symbols from A_{nd} and no epitopes from E_{nd}.*
- *For all $a \in A_{nd} \cap P_i seq$, $\mid \alpha^{-1}(a) \cap P_i epi \mid = 1$, and for all $E \in \mathcal{E}_{nd}^{P_i epi}$, $\mid \alpha(E) \mid = 1$.*

Proof. If P_i contains no symbols from A_{nd} and no epitopes from E_{nd} then it is trivial that $\mathcal{P}$ is strictly globally non-deterministic since no other symbols and epitopes will contribute to non-determinism.

We assume that P_i contain symbols from A_{nd}. Let $a \in A_{nd} \cap P_i seq$. Suppose a satisfies the condition $\mid \alpha^{-1}(a) \cap P_i epi \mid = 1$ then it signifies that there is exactly one $e \in P_i epi$ such that $(e, a) \in \alpha$. Hence the configuration P_i has only one epitope for all non-deterministic symbols in the configuration. This implies the condition (1) is not satisfied in the Definition 13.

If there is a non-deterministic epitope set E in the configuration P_i and satisfies the condition $\mid \alpha(E) \mid = 1$ then it shows that there is only one $a \in P_i sym$ such that $(e, a) \in \alpha$ where $e \in E$. The possibility of more than one e is ruled out by our first assumption that $\mid \alpha^{-1}(a) \cap P_i epi \mid = 1$. This shows that $\mathcal{P}$ is not locally-global non-deterministic.

Hence $\mathcal{P}$ is strictly globally non-deterministic. □

Theorem 3 *A peptide computer $\mathcal{P}$ is strictly locally-global non-deterministic if it satisfies either of the following conditions: for all i,*

- *For all $a \in A_{nd} \cap P_i seq$, if $\chi(tuple(\alpha^{-1}(a) \cap P_i epi)$ is a zero vector or an unit vector.*
- *For all $E \in \mathcal{E}_{nd}^{P_i}$, $\mid \alpha(E) \mid \leq 1$ if $w(E) = 0$ and $\mid \alpha(E) \mid \geq 0$ if $w(E) = 1$.*

Proof. Let $\mathcal{P}$ be locally-global non-deterministic satisfying the above conditions we will show that it is not locally non-deterministic. Since $\mathcal{P}$ is locally-global non-deterministic either of the conditions in Definition 13 is true. Suppose the condition (1) is true for the configuration P_i. Let $a \in P_i sym \cap A_{nd}$. Then there exists epitopes $e_1, e_2, \cdots, e_n, n \geq 2$ such that $e_j \in P_i epi$ and $(e_j, a) \in \alpha$. Hence a has n choices of epitopes to bind. But since we are looking for a locally deterministic one, these choices should not exist when reaction take place, i.e., there should be at most one choice for the symbol a. For this to happen except at most one epitope, say e_i,

a should not dominate any other epitope. This implies all epitopes $e_j (j \neq i)$ are bounded by a symbol – if some of them are not bounded by symbols any one of the epitope overlapping with it is bounded by a symbol. This implies that if we consider $(e_1, e_2, \cdots, e_n)$ as an n-dimensional vector then $\chi((e_1, e_2, \cdots, e_n))$ is a unit vector or zero vector. Hence if $\chi((e_1, e_2, \cdots, e_n))$ is a unit vector or zero vector for all $a \in P_i sym \cap A_{nd}$, then condition (1) of Definition 14 is not satisfied.

Now suppose there is a non-deterministic epitope set E in the configuration. Since we look for a peptide computer that is not locally deterministic there should not be $n (n \geq 2)$ possibilities of symbols binding with epitopes in E. There are only two choices for that: (1) $w(E) > 0$ and (2) $w(E) = 0$ and $n = 1$. If $w(E) > 0$ then there are no open epitopes and hence n can be arbitrary. In other case since $w(E) = 0$ all epitopes in E are open. Hence there should be at most one symbol in competition for an epitope in E.

The discussion shows that $\mathcal{P}$ is strictly locally-global non-deterministic. □

5 Conclusion

We defined three levels of non-determinism in peptide computer: global, locally-global and local. We showed global is a more general definition, locally-global is a restrictive version of global and local is further restrictive version of locally-global. We also characterized conditions for a global system to not to be a locally-global one and locally-global to not to be a local one.

The three levels of non-determinism defined in peptide computer is helpful in the following way: once a (locally-global) global non-deterministic peptide computer is given we can either select the system to be (locally) locally-global non-deterministic or strictly (locally-global) globally-non-deterministic. More interesting question is to study how dynamically under some contextual conditions the system can pick a step to be a locally non-deterministic one or a locally deterministic one. This will control the use of non-determinism to a greater extent.

References

[1] M. S. Balan and H. Jürgensen. Peptide computing: Universality and theoretical model. In *Unconventional Computation*, volume LNCS 4135, pages 57–71. Springer-Verlag, 2006.

[2] M. S. Balan and H. Jürgensen. On the universality of peptide computing. *Natural Computing*, 2007. In print.

[3] M. S. Balan, K. Krithivasan, and Y. Sivasubramanyam. Peptide computing: Universality and computing. In N. Jonoska and N. Seeman, editors, *Proceedings of Seventh International Conference on DNA based Computers, LNCS 2340*, pages 290–299, 2002.

[4] M.S. Balan, H. Jürgensen, and K. Krithivasan. Peptide computing: A survey. Technical Report preprint 4/2005, ISSN 0946-7580, Universität Potsdam, Germany, 2005.

[5] C.R. Cantor and P.R. Schimmel. *Biophysical Chemistry*. W. H. Freeman and Company, San Francisco, 1980. 3 volumes.

[6] H. Hug and R. Schuler. Strategies for the development of a peptide computer. *Bioinformatics*, 17:364–368, 2001.

[7] Y. Ishida. *Immunity-Based Systems: A Design Perspective*. Springer, Berlin, 2004.

[8] H. Jürgensen and S. Konstantinidis. Codes. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 511–607. Springer-Verlag, Berlin, 1997.

[9] A. O. Tarakanov, V. A. Skormin, and S. P. Sokolova. *Immunocomputing, Principles and Applications*. Springer, New York, 2003.

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 120 - 128, 2007.

On Local Testability in Watson-Crick Finite Automata*

José M. Sempere

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera, s/n, 46022 Valencia, Spain
jsempere@dsic.upv.es

Abstract

Watson-Crick finite automata were first proposed in [2] inspired by formal language theory, finite states machines and some ingredients from DNA computing such as working with molecules as double stranded complementary strings. Here, we define different kinds of local testability in this model. Mainly, we will explore local testability in the upper (lower) strand and in the double strand.

1 Introduction

Watson-Crick finite automata (WKFA) [2] is a good example of how DNA biological properties can be adapted to propose computation models in the framework of DNA computing. A recent survey on WKFA has been published in [1]. The WKFA model works with double strings inspired by double-stranded molecules with a complementary relation between symbols (here, inspired by classical complementary relation between nucleotides A-T and C-G). Different restriction over the model have been proposed, mainly devoted to restrict the number of final states (i.e., *all final* and *stateless* WKFA) and the way of processing the upper and lower string (i.e., *1-limited* and *simple* WKFA). Here we propose a different characterization of the model based on a classical concept of formal language theory such as local testability.

Local testable languages were first defined by McNaughton and Papert [5]. These languages have been widely studied in the framework of learning systems (i.e., [3, 10]), DNA and protein analysis (i.e., [12, 13]) and formal languages and semigroups (i.e., [6]), among others.

Here, we will introduce local testability in different ways. First, we will introduce a representation theorem for languages accepted by WKFA, which allows us to study WKFA through linear and even linear languages. Then, we will study two possibilities of defining local testability: in the upper (lower) strand and in the double strand. Finally, we will give some guidelines for future works.

*Work partially supported by the Generalitat Valenciana under research project GV06/068

2 Basic Concepts and Notation

In this section we will introduce basic concepts from formal language theory according to [4, 8] and from DNA computing according to [7].

Formal language theory

An alphabet Σ is a finite nonempty set of elements named symbols. A string defined over Σ is a finite ordered sequence of symbols from Σ. The infinite set of all the strings defined over Σ will be denoted by Σ^*. Given a string $x \in \Sigma^*$ we will denote its length by $|x|$. The empty string will be denoted by λ and Σ^+ will denote $\Sigma^* - \{\lambda\}$. Given a string x we will denote by x^r the reversal string of x. A language L defined over Σ is a set of strings from Σ. Finally, $\Sigma^{\leq k}$ will denote the set of strings with length less than or equals to k and Σ^k will denote the set of strings with length equals to k.

A grammar is a construct $G = (N, \Sigma, P, S)$ where N and Σ are the alphabets of auxiliary and terminal symbols with $N \cap \Sigma = \emptyset$, $S \in N$ is the *axiom* of the grammar and P is a finite set of productions in the form $\alpha \to \beta$. The language of the grammar is denoted by $L(G)$ and it is the set of terminal strings that can be obtained from S by applying symbol substitutions according to P. Formally, $w_1 \underset{G}{\Rightarrow} w_2$ if $w_1 = u\alpha v$, $w_2 = u\beta v$ and $\alpha \to \beta \in P$. We will denote by $\overset{*}{\underset{G}{\Rightarrow}}$ the reflexive and transitive closure of $\underset{G}{\Rightarrow}$.

We will say that a grammar $G = (N, \Sigma, P, S)$ is *right linear* (regular) if every production in P is in the form $A \to uB$ or $A \to w$ with $A, B \in N$ and $u, w \in \Sigma^*$. The class of languages generated by right linear grammars coincides with the class of regular languages and will be denoted by $\mathcal{REG}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is *linear* if every production in P is in the form $A \to uBv$ or $A \to w$ with $A, B \in N$ and $u, v, w \in \Sigma^*$. The class of languages generated by linear grammars will be denoted by $\mathcal{LIN}$. We will say that a grammar $G = (N, \Sigma, P, S)$ is *even linear* if every production in P is in the form $A \to uBv$ or $A \to w$ with $A, B \in N$, $u, v, w \in \Sigma^*$ and $|u| = |v|$. The class of languages generated by even linear grammars will be denoted by $\mathcal{ELIN}$. A well known result from formal language theory is the inclusions $\mathcal{REG} \subset \mathcal{ELIN} \subset \mathcal{LIN}$.

A homomorphism h is defined as a mapping $h : \Sigma \to \Gamma^*$ where Σ and Γ are alphabets. We can extend the definition of homomorphisms over strings as $h(\lambda) = \lambda$ and $h(ax) = h(a)h(x)$ with $a \in \Sigma$ and $x \in \Sigma^*$. Finally, the homomorphism over a language $L \subseteq \Sigma^*$ is defined as $h(L) = \{h(x) : x \in L\}$.

Local testability

Here, we will introduce the definition of local testability and local testability in the strict sense. For any string $x \in \Sigma^*$ and any integer value $k > 0$, the testability vector $v_k(x)$ is defined by the tuple $(i_k(x), t_k(x), f_k(x))$ where

$$i_k(x) = \begin{cases} x, & \text{if } |x| < k \\ u : x = uv, |u| = k - 1 & \text{if } |x| \geq k \end{cases}$$

$$f_k(x) = \begin{cases} x, & \text{if } |x| < k \\ v : x = uv, |v| = k-1 & \text{if } |x| \geq k \end{cases}$$

$$t_k(x) = \{v : x = uvw, u, w \in \Sigma^* \wedge |v| = k\}.$$

We will define the equivalence relation $\equiv_k$ in $\Sigma^* \times \Sigma^*$ as $x \equiv_k y$ iff $v_k(x) = v_k(y)$. It has been proved in [5] that $\equiv_k$ is a finite index relation and that $\equiv_k$ covers $\equiv_{k+1}$.

So, we will say that any language L is k-testable iff it is defined as the union of some equivalence classes of $\equiv_k$. In addition, L is local testable iff it is k-testable for any integer value $k > 0$. The family of k-testable languages will be denoted by $k - \mathcal{LT}$ while $\mathcal{LT}$ will denote the class of testable languages.

A different kind of testability is the so called testability in the strict sense which was again proposed in [5]. Here, for any alphabet Σ we will take the sets $I_k, F_k \subseteq \Sigma^{\leq k-1}$ and $T_k \subseteq \Sigma^k$. Then, a language L is said to be k-testable in the strict sense if the following equation holds

$$L \cap \Sigma^{k-1}\Sigma^* = (I_k\Sigma^*) \cap (\Sigma^* F_k) - (\Sigma^* T_k \Sigma^*).$$

Observe that, according to the last equation, any word in L with length greater than or equals to $k-1$ begins with a segment in I_k, ends with a segment in F_k and has no segment from T_k. Any language L is locally testable in the strict sense iff it is k-testable in the strict sense for any $k > 0$. The family of k-testable languages in the strict sense will be denoted by $k - \mathcal{LTSS}$ while $\mathcal{LTSS}$ will denote the class of testable languages in the strict sense.

It has been proved that $k - \mathcal{LT}$ is the boolean closure of $k - \mathcal{LTSS}$ [14]. Finally, it can be easily proved that both classes $k - \mathcal{LT}$ and $k - \mathcal{LTSS}$ are subclasses of $\mathcal{REG}$.

Watson-Crick finite automata

Given an alphabet $\Sigma = \{a_1, \cdots, a_n\}$, we will use the symmetric (and injective) relation of complementarity $\rho \subseteq \Sigma \times \Sigma$. For any string $x \in \Sigma^*$, we will denote by $\rho(x)$ the string obtained by substituting the symbol a in x by the symbol b such that $(a, b) \in \rho$ (remember that ρ is injective) with $\rho(\lambda) = \lambda$.

Given an alphabet Σ, a *sticker* over Σ will be the pair (x, y) such that $x = x_1 v x_2$, $y = y_1 w y_2$ with $x, y \in \Sigma^*$ and $\rho(v) = w$. The sticker (x, y) will be denoted by $\binom{x}{y}$. A sticker $\binom{x}{y}$ will be a complete and complementary *molecule* if $|x| = |y|$ and $\rho(x) = y$. A complementary and complete molecule $\binom{x}{y}$ will be denoted as $\begin{bmatrix} x \\ y \end{bmatrix}$. Obviously, any sticker $\binom{x}{y}$ or molecule $\begin{bmatrix} x \\ y \end{bmatrix}$ can be represented by $x\#y^r$ where $\# \notin \Sigma$. Here, we will use $x\#y^r$ instead of $x\#y$ due to the grammar construction that we will propose in the following. Furthermore, inspired by DNA structure $x\#y^r$ represents the upper and lower nucleotide strings within the same direction $3' - 5'$ (or $5' - 3'$).

Formally, an *arbitrary* WK finite automaton is defined by the tuple $M = (V, \rho, Q, s_0, F, \delta)$, where Q and V are disjoint alphabets (states and symbols), ρ is a symmetric (and injective) relation of complementarity between symbols of V, s_0 is the initial state, $F \subseteq Q$ is a set of final states and $\delta : Q \times \binom{V^*}{V^*} \to \mathcal{P}(Q)$ (which denotes the power set of Q, that is the set of all possible subsets of Q).

The language of complete and complementary molecules accepted by M will be denoted by the set $L_m(M)$, while the upper strand language accepted by M will be denoted by $L_u(M)$ and defined as the set of strings x such that M, after analyzing the molecule $\begin{bmatrix} x \\ y \end{bmatrix}$ enters into a final state.

A Representation Theorem

Now, given any WKFA M, we will introduce a representation theorem for the languages $L_m(M)$ and $L_u(M)$. First, observe that any double string $\binom{x}{y}$ can be represented by the string $x\#y^r$. Then, the following result holds.

Theorem 1 *(Sempere, [11]) Let $M = (V, \rho, Q, s_0, F, \delta)$ be an arbitrary WK finite automaton. Then there exists a linear language L_1 and an even linear language L_2 such that $L_m(M) = L_1 \cap L_2$.*

The construction for L_1 and L_2 proposed in the theorem is defined as follows. First, the grammar $G_1 = (N, V, P, s_0)$ where $N = Q$, s_0 is the axiom of the grammar and P is defined as

- If $q \in F$ then $q \to \# \in P$.
- If $p \in \delta(q, \binom{x_1}{x_2})$ then $q \to x_1\ p\ x_2^r \in P$.

The language L_2 is defined by the grammar $G_2 = (\{S\}, V, P, S)$ where P is defined as follows

- $S \to \# \in P$.
- For every pair of symbols $a, b \in V$, such that $(a, b) \in \rho$, $S \to aSb \in P$.

It can be easily proved that $L(G_2) = \{x_1\#x_2^r \in V^* : |x_1| = |x_2| \text{ and } \rho(x_1) = x_2\}$. That is, L_2 can be established as the set of complete and complementary molecules $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ in the form $x_1\#x_2^r$.

From L_1 and L_2 it is clear that $L_1 \cap L_2$ is the set of complete and complementary molecules accepted by M in the form $x\#y^r$.

In order to characterize the upper strand language we will provide the following result.

Corollary 1 *(Sempere, [11]) Let $M = (V, \rho, Q, s_0, F, \delta)$ be an arbitrary WK finite automaton. Then $L_u(M)$ can be expressed as $g(h^{-1}(L_1 \cap L_2) \cap R)$ with L_1 being a linear language, L_2 an even linear language, R a regular language and g and h homomorphisms.*

3 Local Testability in Watson-Crick Finite Automata

In this section, we will introduce local testability in the upper or lower strand, and in the double strand of the WKFA model. Given that the languages accepted by arbitrary WKFA can be represented by linear and even linear languages, we will introduce two reductions from these language classes to the class $\mathcal{REG}$.

The first transformation, the so called σ operator, was first introduced in [9] and it was applied for the definition of local testable even linear languages in [10]. It is defined inductively as follows: $\sigma : \Sigma^* \to (\Sigma \times \Sigma)^*(\Sigma \cup \{\lambda\})$ with

1. $\sigma(\lambda) = \lambda$,
2. $(\forall a \in \Sigma)\ \sigma(a) = a$,
3. $(\forall a, b \in \Sigma)\ (\forall x \in \Sigma^*)\ \sigma(axb) = [ab]\sigma(x)$.

The operation σ is applied over languages as $\sigma(L) = \{\sigma(x) : x \in L\}$.

The inverse transformation σ^{-1} can be easily deduced from σ. It has been proved that for every even linear language L, $\sigma(L)$ is regular [9].

The second transformation is a grammatical construction that transforms every linear grammar into an even linear one. It is defined as follows.

Let $G_1 = (N, \Sigma, P, S)$ be a linear grammar. Then $G_2 = (N, \Sigma \cup \{*\}, P', S)$ is an even linear grammar where the productions of P' are defined as follows.

- If $A \to w \in P$ then $A \to w \in P'$.
- If $A \to uBv \in P$ with $|u| = |v|$, then $A \to uBv \in P'$.
- If $A \to uBv \in P$ with $|u| < |v|$, then $A \to u *^{|v|-|u|} Bv \in P'$.
- If $A \to uBv \in P$ with $|u| > |v|$, then $A \to uBv*^{|u|-|v|} \in P'$.

The last grammar is an even linear one and it can be easily proved that $g(L(G_2)) = L(G_1)$ where g is a morphism such that $g(*) = \lambda$ and $g(a) = a$ for every $a \in \Sigma$.

Local testability in the double strand

We will take the representation proposed in theorem 2.1. So, any molecule $\begin{bmatrix} x \\ y \end{bmatrix}$ can be represented by $x\#y^r$. Let us take G_1 as the linear grammar proposed in the theorem and let us take G_2 as the transformed even linear grammar corresponding to G_1. Obviously, for any string $x\#y^r$ of $L(G_1)$ we obtain a string $u\#v$ in $L(G_2)$ such that $g(u)\#g(v) = x\#y^r$, where g is the morphism defined before.

Now, we can work with G_2 and we apply the transformation σ over $L(G_2)$. Observe that $\sigma(L(G_2))$ is regular.

Example 1 *Let $M = (V, \rho, Q, s_0, F, \delta)$ be the WKFA defined by the following transitions*

$$\delta(q_0, \begin{pmatrix} a \\ \lambda \end{pmatrix}) = \{q_a\}, \quad \delta(q_a, \begin{pmatrix} a \\ \lambda \end{pmatrix}) = \{q_a\}, \quad \delta(q_a, \begin{pmatrix} b \\ a \end{pmatrix}) = \{q_b\},$$

$$\delta(q_b, \begin{pmatrix} b \\ a \end{pmatrix}) = \{q_b\}, \quad \delta(q_b, \begin{pmatrix} c \\ b \end{pmatrix}) = \{q_c\}, \quad \delta(q_c, \begin{pmatrix} c \\ b \end{pmatrix}) = \{q_c\},$$

$$\delta(q_c, \begin{pmatrix} \lambda \\ c \end{pmatrix}) = \{q_f\}, \quad \delta(q_f, \begin{pmatrix} \lambda \\ c \end{pmatrix}) = \{q_f\}.$$

Let us take q_f as the final state, q_0 as the initial stated and the complementarity relation $\rho = \{(a,a),(b,b),(c,c)\}$. Then, every complete and complementary molecule accepted by M takes the form $\begin{bmatrix} a^n b^n c^n \\ a^n b^n c^n \end{bmatrix}$ with $n \geq 1$.

Now, the representation linear grammar G_M, according to M is defined by the following productions (take q_0 as the axiom)

$$\begin{array}{ll} q_0 \to aq_a, & q_a \to aq_a \mid bq_ba, \\ q_b \to bq_ba \mid cq_cb, & q_c \to cq_cb \mid q_dc, \\ q_d \to q_dc \mid \#. & \end{array}$$

The corresponding even linear grammar is the following

$$\begin{array}{ll} q_0 \to aq_a*, & q_a \to aq_a* \mid bq_ba, \\ q_b \to bq_ba \mid cq_cb, & q_c \to cq_cb \mid *q_dc, \\ q_d \to *q_dc \mid \#. & \end{array}$$

Finally, we can provide the following right linear grammar to obtain the transformation σ over the last grammar

$$\begin{array}{ll} q_0 \to [a*]q_a, & q_a \to [a*]q_a \mid [ba]q_b, \\ q_b \to [ba]q_b \mid [cb]q_c, & q_c \to [cb]q_c \mid [*c]q_d, \\ q_d \to [*c]q_d \mid \#. & \end{array}$$

Observe that the last grammar generates the language defined as $L = \{[a*]^n[ba]^m[cb]^p[*c]^q\# : n,m,p,q \geq 1\}$. Then, if we take the morphism g with $g(*) = \lambda$ and $g(d) = d$ for every $d \in \{a,b,c,\#\}$ we can obtain $g(\sigma^{-1}(L)) = \{a^n b^m c^p \# c^q b^p a^m : n,m,p,q \geq 1\}$ which, together with the complementary relation ρ, corresponds to the language accepted by M.

So, the definition of local testability (in the strict sense) will be applied over the regular language obtained by the result $\sigma(L(G_M))$ for any WKFA M. Observe that every transformed language in $k-\mathcal{LT}$ ($k-\mathcal{LTSS}$) has a corresponding local testable language defined by the transitions of the WKFA.

Local testability in the upper and lower strand

Now, we will deal only with the upper (lower) strand. Observe that, the definition of the WKFA transitions can be transformed into FA transitions by taking the upper or lower strand (i.e., the transition $p \in \delta(q, \binom{x}{y})$ implies that $p_u \in \delta_u(q, x)$ and $p_l \in \delta_l(q, y)$). So, for every WKFA we can obtain two different finite automata which control the transitions in the upper and lower strands. Here, we will work with *simple* WKFA [7]. We will say that a WKFA is *simple* if for every transition $\delta(q, \binom{x}{y})$ $x = \lambda$ or $y = \lambda$. It has been proved that simple WKFA are normal forms for arbitrary WKFA. That is, for every arbitrary WKFA there exists an equivalent simple WKFA. Furthermore, we can work with the so called 1*limited*WKFA which are simple WKFA where every transition is performed by analyzing only one symbol every time.

Now, we will obtain finite automata from arbitrary 1*limited* WKFA through the following construction. Let $M = (V, \rho, Q, s, F, \delta)$ be an arbitrary 1*limited*WKFA. Then, we can define the finite automaton $A_u = (Q, V, \delta_u, s, F)$, where δ_u is defined as follows

1. $p \in \delta_u(q, a)$ if and only if $p \in \delta(q, \binom{a}{\lambda})$,
2. $p \in \delta_u(q, \lambda)$ if and only if $p \in \delta(q, \binom{\lambda}{a})$.

We can define the finite automaton $A_l = (Q, V, \delta_l, s, F)$ where δ_l is defined as follows

1. $p \in \delta_l(q, a)$ if and only if $p \in \delta(q, \binom{\lambda}{a})$,
2. $p \in \delta_l(q, \lambda)$ if and only if $p \in \delta(q, \binom{a}{\lambda})$.

Example 2 *Let us take the WKFA of example 3.1. Then A_u is defined through the following transitions*

$$\begin{array}{lll} \delta_u(q_0, a) = \{q_a\}, & \delta_u(q_a, a) = \{q_a\}, & \delta_u(q_a, b) = \{q_{bb}\}, \\ \delta_u(q_{bb}, \lambda) = \{q_b\}, & \delta_u(q_b, b) = \{q_{bbb}\}, & \delta_u(q_{bbb}, \lambda) = \{q_b\}, \\ \delta_u(q_b, c) = \{q_{cc}\}, & \delta_u(q_{cc}, \lambda) = \{q_c\}, & \delta_u(q_c, c) = \{q_{ccc}\}, \\ \delta_u(q_{ccc}, \lambda) = \{q_c\}, & \delta_u(q_c, \lambda) = \{q_f\}. & \end{array}$$

*In the previous definitions, the states q_{bb}, q_{bbb}, q_{cc} and q_{ccc} have been introduced in order to obtain an equivalent 1*limited*WKFA from the one proposed initially. In this case $L(A_u) = a^+b^+c^+$. The same holds for $L(A_l)$.*

Observe that, in both automata A_u and A_l, the empty transitions correspond to the case that the WKFA is working in the other strand, so the finite automata ignores all the movements in that way.

Now, the first definitions for local testability come from a natural way of looking up to the FA A_u and A_l. We will say that a 1*limited*WKFA is upper (lower) locally testable (in the strict sense) if the language accepted by A_u (resp. A_l) is locally testable (in the strict sense). Observe that this definition implies the existence of different classes of languages accepted by WKFA which have local testability. These classes are defined as follows

- the class $k - \mathcal{LT}_u$ of languages accepted by 1*limited*WKFA which have k-local testability in the upper strand,
- the class $k - \mathcal{LTSS}_u$ of languages accepted by 1*limited*WKFA which have k-local testability in the strict sense in the upper strand,
- the class $k - \mathcal{LT}_l$ of languages accepted by 1*limited*WKFA which have k-local testability in the lower strand,
- the class $k - \mathcal{LTSS}_u$ of languages accepted by 1*limited*WKFA which have k-local testability in the strict sense in the lower strand.

We can make a step further the definition of a new kind of local testability in every strand by introducing a combination of testability classes considered up to now in an isolated way. Let us take the finite automata A_l and A_u proposed before. Observe that every state in the previous automata defines an equivalence class according to $\equiv_k$ defined in section 2. Now, remember that the relation $\equiv_{k-1}$ covers $\equiv_k$. So, if $L(A_l)$ is in $j - \mathcal{LT}$, then $L(A_l)$ belongs to $k - \mathcal{LT}$ for every $j \leq k$. The same holds for A_u. So, we can combine different equivalence classes in the upper and the lower strand and they define new classes $(k, j) - \mathcal{LT}$ of languages accepted by 1*limited*WKFA which have k-local testability in the upper strand and j-local testability in the lower strand, and the class $(k, j) - \mathcal{LTSS}$ of languages accepted by 1*limited*WKFA which have k-local testability in the strict sense in the upper strand and j-local testability in the strict sense in the lower strand.

4 Conclusions and Future Work

We have presented different ways of introducing local testability in WKFA. The new definitions come from a previous representation result. The new classes inherit the properties of local languages defined in a classical way. Anyway, there exist different relations which should be explored between the language classes defined in the double strand and in every strand separately. In addition, the relation between the language classes defined for upper and lower strand simultaneously should be explored too.

Furthermore, the relation between languages accepted by locally testable WKFA and arbitrary languages should be explored in order to test the power of local testability in these models. These issues will be investigated in future works.

References

[1] E. Czeizler, E. Czeizler. A Short Survey on Watson-Crick Finite Automata. *Bulletin of the EATCS No. 88*, pages 104-119. February, 2006.

[2] R. Freund, G. Păun, G. Rozenberg, A. Salomaa. Watson-Crick finite automata In *Proceedings of DNA Based Computers III DIMACS Workshop (June, 1997)*, pages 297-327. The American Mathematical Society. 1999.

[3] T. Head, S. Kobayashi, T. Yokomori. Locality, Reversibility, and Beyond: Learning Languages from Positive Data. In *Proceedings of 9th International Conference ALT98 (October, 1998)*, pages 191-204. LNCS 1501, Springer, 1998

[4] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley Publishing Co.,1979.

[5] R. McNaughton, S. Papert. *Counter-free automata.* MIT Press, 1971.

[6] J.E. Pin. *Varieties of Formal Languages.* Plenum Publishing Co., 1986.

[7] Gh. Păun, G. Rozenberg, A. Salomaa. *DNA Computing. New computing paradigms.* Springer, 1998

[8] G. Rozenberg, A. Salomaa, editors. *Handbook of Formal Languages, Vol. 1.* Springer, 1997.

[9] J. M. Sempere, P. García. A Characterization of Even Linear Languages and its Application to the Learning Problem. In *Proceedings of the Second International Colloquium on Grammatical Inference, ICGI-94* (September, 1994), pages 28-44. LNAI 862, Springer-Verlag, 1994.

[10] J. M. Sempere, P. García. Learning Locally Testable Even Linear Languages from Positive Data. In *Proceedings of the 6th International Colloquium on Grammatical Inference ICGI 2002 (September, 2002)*, pages 225-236. LNAI 2484, Springer-Verlag, 2002.

[11] J. M. Sempere. A Representation Theorem for Languages accepted by Watson-Crick Finite Automata. *Bulletin of the EATCS No. 83*, pages 187-191. 2004.

[12] T. Yokomori, N. Ishida, S. Kobayashi. Learning local languages and its application to protein α-chain identification. In *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences (January, 1994), Vol.5: Biotechnology Computing,* pages 113-122. IEEE, 1994.

[13] T. Yokomori, S. Kobayashi. Learning local languages and their application to DNA sequence analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1067-1079, 1998.

[14] Y. Zalcstein. Locally Testable Languages. *Journal of Computer and System Sciences*, 6:151-167, 1972.

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 129 - 143, 2007.

Properties of Eco-colonies

Šárka Vavrečková, Alica Kelemenová

Institute of Computer Science, Faculty of Philosophy and Science
Silesian University in Opava
Bezručovo nám. 13, Opava, Czech Republic
{sarka.vavreckova,alica.kelemenova}@fpf.slu.cz

Abstract

Eco-colonies are new grammar systems with very simple grammars called agents placed in a common dynamic environment. Every agent generates its own finite language, all agents cooperate on the shared environment. The environment is developing not only by the action of agents, but also using its own developmental rules.

The generative power of eco-colonies was discussed in several papers, eco-colonies were compared especially with various types of colonies, but not all relations were proved. In this paper we summarize previous results and present some new results about the generative power of eco-colonies.

1 Introduction

Colonies were introduced in [5] as collections of simple grammars (called components) working on a common environment. A component is specified by its start symbol and by its finite language. This language determines actions to do with the start symbol, it is usually a list of words, the component substitutes its start symbol by some of these words. The environment is static itself, only the components can modify it.

There exist several variants of colonies with various types of derivation. The original model was sequential (only one component works in one derivation step), the other basic types of derivation are sequential with parallely working components or parallel. Parallel colonies were introduced in [4], the parallel behavior of a colony means the working of all the components that can work (the components whose start symbols are in the environment and no other component is occupying this symbol for the actual derivation step), one component processes one occurrence of its start symbol.

Eco-colonies were first studied in [10], their E0L form in [11] and [12]. Eco-colonies are colonies with developing environment. The concept of developing of the environment is inspired by another type of grammar systems, eco-grammar systems ([3]). The environment of eco-colonies is specified not only by its alphabets but as 0L or E0L scheme. Every symbol of the environment not processed by agents (components) is overwritten by some of the developing rules of this scheme.

In [1] there is defined a related system, *e-colonies* (extended colonies). Similarly as eco-colonies are based on parallel colonies and their environment is 0L- or E0L-scheme, e-colonies in [1] are based on sequential colonies and their environment is T0L-scheme.

The presented paper consists of four parts. In Section 2 preliminaries are mentioned, than in Section 3 we introduce eco-colonies with two different derivation modes and illustrate these systems on the examples.

In Section 4 we deal with the derivation power of eco-colonies. We compare them mutually, and we compare the generative power of various types of colonies with the generative power of the both types of eco-colonies. We will discuss the systems with single alphabet and also the systems with terminal alphabets.

Section 5 is devoted to the conclusions.

2 Preliminaries

In this section we define colonies and the types of derivation in colonies, and we preface lemmas used in the next sections. For other prerequisites from the theory of formal languages and grammars we refer to [9], related information about theory of grammar systems can be found in [2]. L-systems, 0L-, E0L-, ET0L- and T0L-systems are defined in [8]. For definitions of some properties of languages (e.g. logarithmically clustered, pump-generated) see the paper [7].

In this paper we denote by $|w|_S$ the number of occurrences of S in w for a word w and a symbol S.

Definition 1 *A colony is a (n+3)-tuple* $\mathcal{C} = (V, T, A_1, \ldots A_n, w_0)$, *where*

- V *is a total (finite and non-empty) alphabet of the colony,*
- T *is a non-empty terminal alphabet of the colony,* $T \subset V$,
- $A_i = (S_i, F_i),\ 1 \leq i \leq n$, *is a component, where*
  - $S_i \in V$ *is the start symbol of the component,*
  - $F_i \subseteq (V - \{S_i\})^*$, F_i *is the finite language of this component,*
- w_0 *is the axiom.*

The derivations for colonies were introduced in several ways. Basic of them are following modes:

b-mode is *sequential* type of derivation, one component is active in one derivation step, the active component replaces one occurrence of its start symbol by some word of its finite language F,

t-mode is *sequentially-parallel* – one component is active in one derivation step and this component rewrites all occurrences of its start symbol by words of its language,

wp-**mode** is *parallel* mode, where every component which can work must work in the following sense: each component rewrites at most one occurrence of its start symbol, a component is active if its start symbol is in the environment and no other component with the same start symbol occupies this occurrence of the symbol,

sp-**mode** is *parallel* mode similar to *wp*, but if there is an occurrence of a symbol in the environment, every component with this start symbol has to be active – if all occurrences of this symbol are occupied by another components with the same start symbol, the derivation is blocked.

Definition 2 *We define a basic derivation step (b mode) in a colony* $\mathcal{C}$, $\mathcal{C} = (V, T, A_1, \ldots, A_n, w_0)$ *as the relation* $\overset{b}{\Rightarrow}$ *– α directly derives β in b mode of derivation (written as $\alpha \overset{b}{\Rightarrow} \beta$) if*

- $\alpha = v_1 S v_2$, $\beta = v_1 f v_2$, *where* $v_1, v_2 \in V^*$, $S \in V$, $f \in (V - \{S\})^*$,
- *there exists a component* (S, F) *in* $\mathcal{C}$ *such as* $f \in F$.

Definition 3 *We define a terminal derivation step (t mode) in a colony* $\mathcal{C}$, $\mathcal{C} = (V, T, A_1, \ldots, A_n, w_0)$ *as the relation* $\overset{t}{\Rightarrow}$ *– α directly derives β in t mode of derivation (written as $\alpha \overset{t}{\Rightarrow} \beta$) if*

- $\alpha = v_0 S v_1 S v_2 \ldots v_{n-1} S v_k$,
- $\beta = v_0 f_1 v_1 f_2 v_2 \ldots v_{n-1} f_k v_k$,
- *where* $v_i \in (V - \{S\})^*$, $0 \leq i \leq k$, $S \in V$, $f_i \in (V - \{S\})^*$,
- *there exists a component* (S, F) *in* $\mathcal{C}$ *such as for all strings* f_i, $1 \leq i \leq k$, *is* $f_i \in F$.

Definition 4 *We define a strongly parallel derivation step (sp mode) in a colony* $\mathcal{C} = (V, T, A_1, \ldots, A_n, w_0)$ *as the relation* $\overset{sp}{\Longrightarrow}$ *– α directly derives β in sp mode of derivation (written as $\alpha \overset{sp}{\Longrightarrow} \beta$) if*

- $\alpha = v_0 S_{i_1} v_1 S_{i_2} v_2 \ldots v_{k-1} S_{i_k} v_k$,
- $\beta = v_0 f_{i_1} v_1 f_{i_2} v_2 \ldots v_{k-1} f_{i_k} v_k$,
- *where* $v_j \in V^*$, $0 \leq j \leq k$, $S_{i_j} \in V$, $1 \leq j \leq k$, $f_{i_j} \in \left(V - \{S_{i_j}\}\right)^*$, $1 \leq j \leq k$,
- *there exist components* (S_{i_j}, F_{i_j}) *in* $\mathcal{C}$ *such as* $f_{i_j} \in F_{i_j}$, $1 \leq j \leq k$,
- $i_t \neq i_s$ *for all* $t \neq s$, $1 \leq t, s \leq k$ *(one component can rewrite at most one occurrence of its start symbol),*
- *if* $|\alpha|_S > 0$ *for some symbol* $S \in V$, *then for every component* (S_t, F), *where* $S_t = S$, *is* $t = i_j$ *for some* j, $1 \leq j \leq k$ *(if some symbol occurs in environment, then all components with this symbol as the start symbol must work).*

Definition 5 *We define a weakly parallel derivation step (wp mode) in a colony* $\mathcal{C} = (V, T, A_1, \ldots, A_n, w_0)$ *as the relation* $\overset{wp}{\Longrightarrow}$ *– α directly derives β in wp mode of derivation (written as $\alpha \overset{wp}{\Longrightarrow} \beta$) if*

- $\alpha = v_0 S_{i_1} v_1 S_{i_2} v_2 \dots v_{k-1} S_{i_k} v_k$,
- $\beta = v_0 f_{i_1} v_1 f_{i_2} v_2 \dots v_{k-1} f_{i_k} v_k$,
- *where* $v_j \in V^*,\ 0 \leq j \leq k,\ S_{i_j} \in V,\ 1 \leq j \leq k,\ f_{i_j} \in \left(V - \{S_{i_j}\}\right)^*,\ 1 \leq j \leq k$,
- *there exist components* (S_{i_j}, F_{i_j}) *in* $\mathcal{C}$ *such as* $f_{i_j} \in F_{i_j},\ 1 \leq j \leq k$,
- $i_t \neq i_s$ *for all* $t \neq s,\ 1 \leq t, s \leq k$ *(one component can rewrite at most one occurrence of its start symbol),*
- *for every* $S \in V$, *if the number of agents with the start symbol* S *is denoted by* t, *then*

$$\sum_{\substack{j=1 \\ S_{i_j}=S}}^{r} |\alpha|_{S_{i_j}} = min\left(|\alpha|_S, t\right)$$

(all components which can work – their start symbol is in the environment and some of the occurrences of this symbol is not occupied by any other agent – they must work; the left side of the equation means the number of components with the start symbol S *which work in the given derivation step).*

The formal definitions of an eco-grammar system and its type of derivation are in [3].

For all the relations $\overset{x}{\Longrightarrow},\ x \in \{b, t, wp, sp\}$, we define the reflexive and transitive closure $\overset{x}{\Longrightarrow}^*$.

Definition 6 *Let* $\mathcal{C}$ *be a colony and* $\mathcal{C} = (V, T, A_1, \dots A_n, w_0)$. *The language generated by the derivation step* $x,\ x \in \{b, t, wp, sp\}$ *in* $\mathcal{C}$ *is*

$$L(\mathcal{C}, x) = \{w \in T^* \ :\ w_0 \overset{x}{\Longrightarrow}^* w\}.$$

For more information about languages of colonies see [6].

We use the notations for colonies with various types of derivation:

COL_x for class of languages generated by colonies with x type of derivation, $x \in \{b, t, wp, sp\}$,

COL_x^T for class of languages generated by colonies with $T = V$ and x type of derivation, $x \in \{b, t, wp, sp\}$.

Lemma 1

$$COL_x^T \subseteq COL_x$$

where $x \in \{b, t, wp, sp\}$.

Proof. Colonies generating the class COL_x^T are colonies with only one alphabet $(T = V)$, it is a special type of colonies generating COL_x. □

Let $\mathcal{C}$ be a colony, $\mathcal{C} = (V, T, A_1, \dots A_n, w_0)$, with n components. Denote by m the length of the longest word in the languages of components, over the all components $A_1, \dots, A_n$:

$$m = max\left\{|u| \ :\ u \in F_i,\ A_i = (S_i, F_i),\ 1 \leq i \leq n\right\}.$$

Lemma 2 (Pumping lemma for parallel colonies) *Let L be an infinite language generated by a colony $\mathcal{C}$ with $x \in \{wp, sp\}$ derivation mode. Then the length set of L contains infinite linearly dependent subsets, i.e.*

$$\{a \cdot i + b \ : \ i \geq 0\} \subseteq \{|w| \ : \ w \in L\}$$

for some natural numbers $a, b > 0$.

Proof. Let $\mathcal{C} = (V, T, A_1, \ldots A_n, w_0)$ be a colony with $x \in \{wp, sp\}$ derivation mode and $L(\mathcal{C}, x) = L$ for some infinite language L. Let m be length of the longest word in the languages of components $A_1, \ldots, A_n$,

$$m = max\,\{|u| \ : \ u \in F_i, \ A_i = (S_i, F_i), \ 1 \leq i \leq n\}\,.$$

Let us choose some w in L, $|w| \geq |w_0| \cdot m \cdot n \cdot 2^n$, the derivation of word w from the axiom consists of at least 2^n steps. Since in one derivation step $w_i \overset{x}{\Longrightarrow} w_{i+1}$ we have $|w_{i+1}| - |w_i| \leq m \cdot n$. Therefore there are indices $i, j, \ i < j$, such that the same set of agents is active in the derivation steps $w_i \overset{x}{\Longrightarrow} w_{i+1}$ and $w_j \overset{x}{\Longrightarrow} w_{j+1}$.

We split this derivation to the parts

$$w_0 \overset{x}{\Longrightarrow}^* w_i \overset{x}{\Longrightarrow}^* w_j \overset{x}{\Longrightarrow}^* w$$

Denote by

- n_0 number of terminal symbols generated in the subderivation $w_0 \overset{x}{\Longrightarrow}^* w_i$, which are not rewritten in any next derivation step,
- n_i the same for the subderivation $w_i \overset{x}{\Longrightarrow}^* w_j$,
- n_j the same for the subderivation $w_j \overset{x}{\Longrightarrow}^* w$.

Now we transform the derivation as follows:

- in the derivation step $w_j \overset{x}{\Longrightarrow} \ldots$ we use the same components and words of languages of these components as in the derivation step $w_i \overset{x}{\Longrightarrow} \ldots$,
- in this way we link up a copy of processing the sets of symbols from the subderivation $w_i \overset{x}{\Longrightarrow}^* w$ to the subderivation $w_j \overset{x}{\Longrightarrow} \ldots$ (we link up only the way of rewriting symbols, the other symbols stay in the word),
- we apply the previous operation z-times, $z \geq 0$,
- the word derived using the described method of "pumping" the derivation is denoted by w'_z.

The described derivations for the numbers z generate the words with the following length:

$$|w| = |w'_1| \ = \ n_0 + n_i + n_j, \tag{1}$$

$$|w'_z| \ = \ n_0 + z \cdot n_i + n_j. \tag{2}$$

We can construct the derivation of w'_z for any $z \geq 0$, so $w'_z \in L$. Linear dependence is obvious. □

The following theorems are used in the proof of Theorem 7.

Theorem 1 ([7]) *If K is an infinite $ET0L_{[1]}$ language[1] then either K contains an infinite logarithmically clustered language or K contains a pump-generated language.*

Theorem 2 ([4])

$$COL_t = ET0L_{[1]}.$$

3 Eco-Colonies

In this section we define two types of eco-colonies and then two types of derivation in eco-colonies.

Definition 7 *An E0L eco-colony of degree n, $n \geq 1$, is an $(n+2)$-tuple $\Sigma = (E, A_1, A_2, \ldots, A_n, w_0)$, where*

- *$E = (V, T, P)$ is E0L scheme, where*
  - *V is an alphabet,*
  - *T is a terminal alphabet, $T \subseteq V$,*
  - *P is a finite set of E0L rewriting rules over V,*
- *$A_i = (S_i, F_i)$, $1 \leq i \leq n$, is the i-th agent, where*
  - *$S_i \in V$ is the start symbol of the agent,*
  - *$F_i \subseteq (V - \{S_i\})^*$ is a finite set of action rules of the agent (the language of the agent),*
- *w_0 is the axiom.*

An *0L eco-colony* is defined similarly, the environment is 0L scheme $E = (V, P)$, P is a finite set of 0L rewriting rules over V.

As we can see, agents are defined in the same way as components in colonies, an environment is determined by the alphabets in colonies, and by E0L or 0L scheme in eco-colonies.

We define two derivation modes for eco-colonies – the first one, *wp*, is inspired by the *wp* mode for colonies, we only add the possibility of developing for the environment. In every derivation step each agent (S, F) looks for its start symbol S. If it finds some occurrence of this symbol not occupied by any other agent, the agent becomes active, occupies this symbol and rewrites it by some of words of its language F.

Definition 8 *We define a weakly competitive parallel derivation step in an eco-colony $\Sigma = (E, A_1, A_2, \ldots, A_n, w_0)$ as the relation $\overset{wp}{\Longrightarrow}$ – α directly derives β in wp mode of derivation (written as $\alpha \overset{wp}{\Longrightarrow} \beta$) if*

[1] $ET0L_{[1]}$ languages are languages generated by 1-restricted $ET0L$ systems: 1-restricted $ET0L$ system is $ET0L$ system $G = (\Sigma, \mathcal{P}, S, \Delta)$ such that for every table $P \in \mathcal{P}$ there exists a letter $b \in \Sigma$ such that if $c \in \Sigma - \{b\}$ and $(c \to \alpha) \in P$ then $\alpha = c$ (in every table only one rule is not static).

- $\alpha = v_0 S_{i_1} v_1 S_{i_2} v_2 \dots v_{r-1} S_{i_r} v_r,\ r > 0,$
- $\beta = v'_0 f_{i_1} v'_1 f_{i_2} v'_2 \dots v'_{r-1} f_{i_r} v'_r, \quad$ *for* $A_{i_k} = (S_{i_k}, F_{i_k}),\ f_{i_k} \in F_{i_k},\ 1 \leq k \leq r,$
- $i_k \neq i_m$ *for every* $k \neq m,\ 1 \leq k, m \leq r$ *(the agent* A_{i_k} *is* active *in this derivation step),*
- $\{i_1, i_2, \dots, i_r\} \subseteq \{1, 2, \dots, n\},$
- *for every* $S \in V$*, if the number of agents with the start symbol* S *is denoted by* t*, then*

$$\sum_{\substack{j=1 \\ S_{i_j}=S}}^{r} |\alpha|_{S_{i_j}} = min\,(|\alpha|_S, t)$$

  (all agents which can work – their start symbol is in the environment and some of the occurrences of this symbol is not occupied by any other agent – they must work; the left side of the equation means the number of agents with the start symbol S *which work in the given derivation step),*

- $v_k \overset{E}{\Longrightarrow} v'_k,\ v_k \in V^*,\ 0 \leq k \leq r,$ *is the derivation step of the scheme* E*.*

The second type of derivation step, *ap*, means that all agents must work in every derivation step and if some agent is not able to work (there is not any free occurrence of its start symbol), the derivation is blocked. This type of derivation is inspired by the basic type of derivation in eco-grammar systems.

Definition 9 *We define a derivation step ap (all are working parallely) in an eco-colony* $\Sigma = (E, A_1, A_2, \dots, A_n, w_0)$ *as the relation* $\overset{ap}{\Longrightarrow}$ *–* α *directly derives* β *in ap mode of derivation (written as* $\alpha \overset{ap}{\Longrightarrow} \beta$*) if*

- $\alpha = v_0 S_{i_1} v_1 S_{i_2} v_2 \dots v_{n-1} S_{i_n} v_n,$
- $\beta = v'_0 f_{i_1} v'_1 f_{i_2} v'_2 \dots v'_{n-1} f_{i_n} v'_n, \quad$ *for* $A_{i_k} = (S_{i_k}, F_{i_k}),\ f_{i_k} \in F_{i_k},\ 1 \leq k \leq n,$
- $\{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\}$ *(every agent works in every derivation step),*
- $v_k \overset{E}{\Longrightarrow} v'_k,\ v_k \in V^*,\ 0 \leq k \leq n,$ *is the derivation step of the scheme* E*.*

For the relations $\overset{x}{\Longrightarrow},\ x \in \{wp, ap\}$, we define the reflexive and transitive closure $\overset{x}{\Longrightarrow}^*$.

Definition 10 *Let* Σ *be an 0L eco-colony,* $\Sigma = (E, A_1, A_2, \dots, A_n, w_0)$*. The language generated by the derivation step* $x,\ x \in \{wp, ap\}$*, in* Σ *is*

$$L(\Sigma, x) = \{w \in V^* \ :\ w_0 \overset{x}{\Longrightarrow}^* w\}.$$

Let Σ *be an E0L eco-colony,* $\Sigma = (E, A_1, A_2, \dots, A_n, w_0)$*. The language generated by the derivation step* $x,\ x \in \{wp, ap\}$*, in* Σ *is*

$$L(\Sigma, x) = \{w \in T^* \ :\ w_0 \overset{x}{\Longrightarrow}^* w\}.$$

Example 1 Let $\Sigma = (E, A_1, A_2, AbB)$ be an E0L eco-colony, where

$$E = (\{A, B, a, b\}, \{a, b\}, \{a \to a, b \to bb, A \to A, B \to B\}),$$

$$A_1 = (A, \{aB, \varepsilon\}), \quad A_2 = (B, \{aA, \varepsilon\}).$$

Let us construct derivations with *ap* and *wp* types of derivations:

$$AbB \overset{ap}{\Longrightarrow} aBb^2aA \overset{ap}{\Longrightarrow} a^2Ab^4a^2B \overset{ap}{\Longrightarrow} \ldots \overset{ap}{\Longrightarrow} a^nAb^{2^n}a^nB \overset{ap}{\Longrightarrow} a^nb^{2^{(n+1)}}a^n,$$

$$AbB \overset{wp}{\Longrightarrow} aBb^2aA \overset{wp}{\Longrightarrow} a^2Ab^4a^2B \overset{wp}{\Longrightarrow} a^2b^8a^3A \overset{wp}{\Longrightarrow} a^2b^{16}a^4B \overset{wp}{\Longrightarrow} \ldots$$

The *wp* derivation allows "resting" of non-active agents. If we use the *ap* type of derivation, a terminal word is generated only if the both agents use the ε-rule in the same derivation step, otherwise the derivation is blocked without creating the final word.

The generated languages are:

$$\begin{aligned} L(\Sigma, ap) &= \left\{a^nb^{2^{(n+1)}}a^n \ : \ n \geq 0\right\}, \\ L(\Sigma, wp) &= \left\{a^ib^{2^n}a^j \ : \ 0 \leq i, j < n\right\}. \end{aligned}$$

4 Generative Power of Eco-Colonies

We compare the generative power of eco-colonies and colonies, for systems with terminal alphabets as well as for special systems with the terminal alphabet equal to the alphabet of the system. For eco-colonies we use the notations:

$0EC_x$ for the class of languages generated by 0L eco-colonies with x type of derivation, $x \in \{wp, ap\}$,

EEC_x for the class of languages generated by E0L eco-colonies with x type of derivation, $x \in \{wp, ap\}$.

Theorem 3

$$COL_{wp} \subset EEC_{wp}. \tag{3}$$

Proof. The relation $COL_{wp} \subseteq EEC_{wp}$ is trivial, colonies with *wp* derivation are a special version of E0L eco-colonies with a static environment (with rules $a \to a$ for every letter from V). To prove the proper inclusion we use the language

$$L_1 = \left\{a^{2^n} \ : \ n \geq 0\right\}.$$

The language L_1 is generated by the eco-colony $\Sigma = (E, A, b)$, where

$$E = (\{a, b\}, \ \{a\}, \ \{a \to aa, \ b \to b\}), \quad A = (b, \ \{a\}).$$

The language L_1 does not include infinite subsets of words with linearly dependent length so according to Lemma 2 there is no colony C with *wp* derivation which generates the language L_1. □

Corollary 1

$$COL_b \subset EEC_{wp}, \tag{4}$$
$$COL_b^T \subset EEC_{wp}, \tag{5}$$
$$COL_{wp}^T \subset EEC_{wp}. \tag{6}$$

Proof. Equation (4) follows from $COL_b \subset COL_{wp}$ ([4]) and from Equation (3). Equations (5) and (6) follow from Lemma 1 and from Equations (3) and (4). □

Theorem 4

$$0EC_{wp} \subset EEC_{wp}. \tag{7}$$

Proof. The relation $0EC_{wp} \subseteq EEC_{wp}$ is trivial, 0L eco-colonies are the special type of E0L eco-colonies with the terminal alphabet $T = V$.

We can find a language $L_2 \in EEC_{wp} - 0EC_{wp}$:

$$L_2 = \left\{a^{2^i} \ : \ i \geq 0\right\} \cup \left\{b^{3^i} \ : \ i \geq 0\right\}.$$

This language is generated by the E0L eco-colony $\Sigma = (E, A, S)$, where

$$\begin{aligned} E &= (\{S, a, b\},\ \{a, b\},\ \{a \to aa,\ b \to bbb,\ S \to S\}), \\ A &= (S, \{a, b\}) \text{ (this agent is active only in the first derivation step)}, \end{aligned}$$

$S \overset{wp}{\Longrightarrow} a \overset{wp}{\Longrightarrow} a^2 \overset{wp}{\Longrightarrow} a^4 \overset{wp}{\Longrightarrow} a^8$,
$S \overset{wp}{\Longrightarrow} b \overset{wp}{\Longrightarrow} b^3 \overset{wp}{\Longrightarrow} b^9 \overset{wp}{\Longrightarrow} b^{27}$.

Assume that some 0L eco-colony $\Sigma_0 = (E, A_1, A_2, \ldots, A_n, w_0),\ E = (V, P)$, generates the language L_2. Every state in the environment including the axiom is one of the elements of the language of Σ_0.

Let the rule $a \to \varepsilon$ is in P (the case for b is analogous). If we have only the ε-rule for a there, the exponential growing would be carried by agents, but the agents work similarly to the components in colonies. Components are not able to ensure exponential growing (see Lemma 2), nor agents in this eco-colony.

If there are some non-ε-rules in the environment, the ε-rule is not allowed, because the random application of this rule would mean random disappearing of symbols in the environment, so some words not contained in L_2 could be generated. That is why the axiom is one of the two shortest words – a or b.

Suppose the axiom a. We need to generate every word of the language L_2 including the words b^{3^i}, so the rule $a \to b$ is in the language of some agent or it is a rule of the 0L scheme in the environment.

If this rule is used by some agent, the eco-colony can generate only the words $a \cup b^{3^i}$, because the agent must work whenever it can work. If some another agent rewrites symbols b to a, it is able to do it in the next derivation step, but every state of the environment belongs to the language generated by Σ_0, including the states before and after application of this derivation step. In this case only one derivation is possible, $a \overset{wp}{\Longrightarrow} b \overset{wp}{\Longrightarrow} a \overset{wp}{\Longrightarrow} b \overset{wp}{\Longrightarrow} \ldots$, it generates the language $\{a, b\}$.

The superior indexes 2 and 3 in the definition of L_2 have not any common divisor, so the alternate rewriting of all the symbols a to b and then b to a with the growing length of the words by the environment is not possible.

So if the rule $a \to b$ (or some rule rewriting a to more than one b) is in the 0L scheme of the environment and the 0L scheme is deterministic, the eco-colony is not able to generate any word of the form a^{2^i} longer than the power of the number of agents in this system, because the deterministic 0L scheme does not contain any rule rewriting a to a sequence of a. The rules rewriting b to a sequence of a are not usable as suggested in the previous paragraph.

If the 0L scheme is not deterministic, this situation allows to have more than one rule for rewriting the symbol a – one rule $a \to b$ and some rule rewriting a to a sequence of a. But in this case the eco-colony can generate some words containing both the symbols a and b, and these words are not elements of the language L_2.

The case of the axiom b can be solved similarly, so any 0L eco-colony cannot generate the language L_2. □

Theorem 5

$$0EC_{ap} \subset EEC_{ap}. \tag{8}$$

Proof. 0L eco-colonies are special types of E0L eco-colonies where $T = V$, so the relation $0EC_{ap} \subseteq EEC_{ap}$ is trivial.

To prove the proper subset we use language

$$L_3 = L_1 - \{a\} = \left\{a^{2^n} \ : \ n \geq 1\right\}.$$

This language is generated by the E0L eco-colony $\Sigma = (E, A_1, A_2, UVa)$, where $E = (\{a, U, V\}, \{a\}, \{a \to aa, U \to U, V \to V\})$, $A_1 = (U, \{V, \varepsilon\})$, $A_2 = (V, \{U, \varepsilon\})$.

$$UVa \overset{ap}{\Longrightarrow} VUa^2 \overset{ap}{\Longrightarrow} UVa^4 \overset{ap}{\Longrightarrow} \ldots \overset{ap}{\Longrightarrow} UVa^{2^{n-1}} \overset{ap}{\Longrightarrow} a^{2^n}.$$

Each agent generates the empty word only and using the ap derivation agents A_1 and A_2 are active in every derivation step and they alternate symbols U, V until the terminal word is generated.

Suppose that the language L_3 can be generated by some 0L eco-colony Σ with ap derivation. Σ contains at least one agent, which is active in every derivation step. $V = \{a\}$, so the start symbol of each agent is a. The agent generates a finite language over $V - \{a\}$, so we have $A = (a, \{\varepsilon\})$ for each agent in Σ.

P is deterministic, it contains exactly one rule for a. (Otherwise the system generates an infinite set of pairs of words with the constant difference of their length and there is no such an infinite subset in L_3.)

The language generated with the 0L eco-colony where $P = \{a \to a^s\}$ with n agents $A = (a, \{\varepsilon\})$ using the ap mode from the axiom a^m is equal to $\left\{a^{2^n} \ : \ n \geq 1\right\}$ for no parameters m, n, s and $L_3 \notin EEC_{ap}$. □

Theorem 6 *The classes of languages $0EC_{wp}$ and $0EC_{ap}$ are incomparable.*

Proof. 1) $0EC_{wp} - 0EC_{ap} \neq \emptyset$:

In Theorem 5 we proved that the language $L_3 = \left\{a^{2^n} \ : \ n \geq 1\right\}$ is not generated by any 0L eco-colony with *ap* derivation. This language is generated by the following 0L eco-colony with *wp* derivation: $\Sigma = (E, A, a)$, where $E = (\{a, b\}, \{a\}, \{a \to aa, b \to b\})$, $A = (b, \{a\})$.

We need at least one agent, but using *wp* derivation this agent does not work if its start symbol is not in the environment.

2) $0EC_{ap} - 0EC_{wp} \neq \emptyset$:

To prove this we use language

$$\begin{aligned} L_4 &= \left\{a^{15-2n}b^ncb^nd \ : \ 0 \leq n < 7, n \text{ is even}\right\} \\ &\cup \left\{a^{15-2n}b^ndb^nc \ : \ 0 < n \leq 7, n \text{ is odd}\right\}. \end{aligned}$$

This language is generated by the 0L eco-colony $\Sigma = (E, A_1, A_2, A_3, A_4, a^{15}cd)$ with *ap* derivation, where

$$E = (\{a, b, c, d\}, \{a \to a, b \to b, c \to c, d \to d\}),$$

$$A_1 = (a, \{\varepsilon\}), \ A_2 = (a, \{\varepsilon\}), \ A_3 = (c, \{bd\}), \ A_4 = (d, \{bc\}).$$

This language consists only of eight words derived as follows:

$$a^{15}cd \overset{ap}{\Longrightarrow} a^{13}bdbc \overset{ap}{\Longrightarrow} a^{11}b^2cb^2d \overset{ap}{\Longrightarrow} a^9b^3db^3c \overset{ap}{\Longrightarrow} a^7b^4cb^4d \overset{ap}{\Longrightarrow} a^5b^5db^5c \overset{ap}{\Longrightarrow}$$

$$\overset{ap}{\Longrightarrow} a^3b^6cb^6d \overset{ap}{\Longrightarrow} ab^7db^7c.$$

Assume that there exists an 0L eco-colony Σ_0 with *wp* derivation generating L_4. Suppose that the axiom is $a^{15-2i}b^icb^id$ for some i, $0 \leq i \leq 7$, i is even (the proof for the axiom with odd number i is analogous). Σ_0 generates all words of the language for $n > i$ and/or $n < i$.

a) Words for $n < i$: $\quad a^{15-2i}b^icb^id \overset{wp}{\Longrightarrow}{}^{+} a^{15}dc\ldots$

The number of a-s increases, the number of b-s decreases. But with using the *wp* type of derivation the system is not able to stop growing of a-s, so it is possible to generate words not included in L_4 such as $a^{19}cd$.

b) Words for $n > i$: $\quad a^{15-2i}b^icb^id \overset{wp}{\Longrightarrow}{}^{+} a^{13-2i}b^{i+1}db^{i+1}c\ldots$

The number of a-s decreases, the number of b-s increases. As in the previous part of this proof, the system is not able to stop growing of b-s, the words b^8cb^8d, etc. not included in L_4 are generated.

The outcome is identical for growing by agents as well as by the environment. □

Theorem 7

$$xEC_y - COL_z \neq \emptyset \tag{9}$$

where $x \in \{0, E\}$, $y \in \{wp, ap\}$, $z \in \{b, t, wp, sp\}$.

Proof. In this proof we use language

$$L_5 = \left\{ cda^{2^{2n}} b^{2^{2n}} \ : \ n \geq 0 \right\} \cup \left\{ dca^{2^{2n+1}} b^{2^{2n+1}} \ \middle| \ n \geq 0 \right\}.$$

This language can be generated by the eco-colony $\Sigma = (E, A_1, A_2, cdab)$, where $E = (\{a, b, c, d\}, \{a \to aa, b \to bb, c \to c, d \to d\})$, $A_1 = (c, \{d\})$, $A_2 = (d, \{c\})$.

$$cdab \Rightarrow dca^2b^2 \Rightarrow cda^4b^4 \Rightarrow dca^8b^8 \Rightarrow cda^{16}b^{16} \Rightarrow \ldots$$

Considering $T = V$ this is 0L as well as E0L eco-colony. Both agents are active for all words, i.e. in every derivation step so wp and ap coincide in it.

The language L_5 is not context-free, so $L_5 \notin COL_b$ and it grows exponentially so $L_5 \notin COL_{wp}$ and $L_5 \notin COL_{sp}$ according to Lemma 2.

$L_5 \notin COL_t$ according to the results of Kleijn and Rozenberg, see Theorems 1 and 2 in Preliminaries. □

Corollary 2

$$xEC_y - COL_z^T \neq \emptyset \tag{10}$$

where $x \in \{0, E\}$, $y \in \{wp, ap\}$, $z \in \{b, t, wp, sp\}$.

Proof. Follows from Theorem 7 and Lemma 1. □

Corollary 3

$$COL_{wp}^T \subset 0EC_{wp}, \tag{11}$$
$$COL_b^T \subset 0EC_{wp}. \tag{12}$$

Proof. Colonies COL_{wp}^T are a special type of eco-colonies $0EC_{wp}$ with the static environment (only rules of type $a \to a$), so $COL_{wp}^T \subseteq 0EC_{wp}$. Equation (11) follows from this fact and from Corollary 2.

Colonies COL_b^T can be simulated by colonies COL_{wp}^T where every possible pair of components has different start symbols, so $COL_b^T \subseteq COL_{wp}^T$. This gives inclusion (12). □

Theorem 8

$$COL_x - 0EC_{wp} \neq \emptyset, \quad x \in \{b, t, wp, sp\}. \tag{13}$$

Proof. In Theorem 6 we proved that the language

$$\begin{aligned} L_4 \ &= \ \left\{ a^{15-2n} b^n c b^n d \ : \ 0 \leq n < 7, n \text{ is even} \right\} \\ &\cup \ \left\{ a^{15-2n} b^n d b^n c \ : \ 0 < n \leq 7, n \text{ is odd} \right\} \end{aligned}$$

is not in $0EC_{wp}$. It is a finite language, so $L_4 \in COL_x$ for $x \in \{b, t, wp, sp\}$. □

Theorem 9

$$COL_x - 0EC_{ap} \neq \emptyset, \quad x \in \{b, t, wp, sp\}. \tag{14}$$

Proof. The finite language

$$L_6 = \{a, aa\}$$

is produced by a colony with one component $(S, \{a, aa\})$ and axiom S for any derivation mode $x, x \in \{b, t, wp, sp\}$, therefore $L_6 \in COL_x$.

In an 0L eco-colony we have only one alphabet. So all active agents have the start symbol a and the form $(a, \{\varepsilon\})$. The axiom is one of the words of the language – a or aa.

Assume that the axiom is a. There exists at least one agent rewriting a to ε, so the generated language is $\{a, \varepsilon\}$. But the empty word $\varepsilon \notin L_6$.

Suppose that the axiom is aa. There exists some agent rewriting one of the both a-s to ε, so the word a can be generated. But this agent works in the next derivation step (or steps) too: $aa \overset{ap}{\Longrightarrow}{}^* a \overset{ap}{\Longrightarrow}{}^* \varepsilon$, and the word not contained in L_6 is generated. So $L_6 \notin 0EC_{ap}$. □

Theorem 10

$$COL_b \subset EEC_{ap}. \tag{15}$$

Proof. We have a colony with the b mode of derivation $\mathcal{C} = (V, T, A_1, \dots A_n, w_0)$, and we create an equivalent E0L eco-colony $\Sigma = (E, A_1, A_2, BCw_0)$ with the ap derivation and agents $A_1' = (B, \{C, \varepsilon\})$, $A_2' = (C, \{B, \varepsilon\})$.

We create rules of the environment from the components $A_1, \dots A_n$. We can suppose that all these components have different start symbols.

For each component $(a, \{\alpha_1, \alpha_2, \dots, \alpha_k\})$ we create developing rules for the environment:

$$a \to a \mid \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$$

and for every symbol b which is not the start symbol in any component we create one rule $b \to b$.

So the environment simulates the action of the components in the colony. The simulation of a sequential derivation is possible using the identical rules rewriting symbol to itself for all but one letter of the word.

From the construction it follows that $w_0 \overset{b}{\Rightarrow}{}^* w$ implies $BCw_0 \overset{ap}{\Longrightarrow}{}^* w$ and $BCw_0 \overset{ap}{\Longrightarrow}{}^* w$ implies $w_0 \overset{b}{\Rightarrow}{}^* w$.

The proper subset comes from Theorem 5. □

Example 2 We demonstrate the construction of the proof on the colony generating the language

$$L_7 = \left\{ waw^R a^i \ : \ w \in \{0,1\}^*, \ i > 0 \right\}.$$

We have a colony $\mathcal{C} = (\{S, H, H', A, A', 0, 1, a\}, \{0, 1, a\}, A_1, A_2, A_3, A_4, A_5, S)$ generating the language L_7 where

$$A_1 = (S, \ \{HA\}), \qquad A_2 = (H, \ \{0H'0, 1H'1, a\}), \qquad A_4 = (A, \ \{aA', a\}),$$
$$A_3 = (H', \ \{H\}), \qquad A_5 = (A', \ \{A\}).$$

Now we create an E0L eco-colony with ap derivation $\Sigma = (E, A_1, A_2, BCS)$, $E = (\{B, C, S, H, H', A, A', 0, 1, a\}, \ \{0, 1, a\}, \ P)$, $A_1 = (B, \{C, \varepsilon\})$, $A_2 = (C, \{B, \varepsilon\})$, the set of rules P in the environment is

$$P = \{\ H \to H \mid 0H'0 \mid 1H'1 \mid A, \quad H' \to H' \mid H, \quad 1 \to 1, \quad a \to a,$$
$$A \to A \mid aA' \mid a, \quad A' \to A' \mid A, \quad 0 \to 0, \quad S \to S \mid HA\ \}.$$

One of the derivations in $\mathcal{C}$:

$$S \overset{b}{\Rightarrow} HA \overset{b}{\Rightarrow} 1H'1A \overset{b}{\Rightarrow} 1H1A \overset{b}{\Rightarrow} 10H'01A \overset{b}{\Rightarrow} 10H01A \overset{b}{\Rightarrow} 10a01A \overset{b}{\Rightarrow}$$
$$\overset{b}{\Rightarrow} 10a01aA' \overset{b}{\Rightarrow} 10a01aA \overset{b}{\Rightarrow} 10a01aa.$$

Two of possible derivations of the same word in Σ:

$$BCS \overset{ap}{\Longrightarrow} CBHA \overset{ap}{\Longrightarrow} BC1H'1A \overset{ap}{\Longrightarrow} CB1H1A \overset{ap}{\Longrightarrow} BC10H'01A \overset{ap}{\Longrightarrow}$$
$$\overset{ap}{\Longrightarrow} CB10H01A \overset{ap}{\Longrightarrow} BC10a01A \overset{ap}{\Longrightarrow} CB10a01aA' \overset{ap}{\Longrightarrow} BC10a01aA \overset{ap}{\Longrightarrow}$$
$$\overset{ap}{\Longrightarrow} 10a01aa, \text{ and}$$

$$BCS \overset{ap}{\Longrightarrow} CBHA \overset{ap}{\Longrightarrow} BC1H'1aA' \overset{ap}{\Longrightarrow} CB1H1aA \overset{ap}{\Longrightarrow} BC10H'01aa \overset{ap}{\Longrightarrow}$$
$$\overset{ap}{\Longrightarrow} CB10H01aa \overset{ap}{\Longrightarrow} 10a01aa.$$

Corollary 4

$$COL_b^T \subset EEC_{ap}. \tag{16}$$

Proof. Follows from Theorem 10 and Lemma 1. □

5 Conclusions

In this paper we study the type of grammar systems, eco-colonies based on colonies and eco-grammar systems. We summarize the results in the table 1. The symbol *T<number>* means Theorem with the referred number, the symbol *C<number>* means Corollary with the referred number. The symbol ⓒⓒ in the table means incomparable classes of languages.

References

[1] E. Csuhaj-Varjú. Colonies – A Multi-agent Approach to Language Generation. In *Proc. ECAI'96 Workshop on Finite State Models of Languages*, pages 12–16. NJSZT, Budapest, 1996.

[2] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun. *Grammar Systems. A Grammatical Approach to Distribution and Cooperation.* Gordon & Beach, London, 1994

[3] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun. Eco-grammar Systems. Grammatical Framework for Studying Lifelike Interactions. *Artificial Life*, 3:1–28, 1997.

	$0EC_{wp}$	$0EC_{ap}$	EEC_{wp}	EEC_{ap}
COL_b	$\bigcirc\!\!\bigcirc$ $T7,T8$	$\bigcirc\!\!\bigcirc$ $T7,T9$	$\subset$ $C1$	$\subset$ $T10$
COL_t	$\bigcirc\!\!\bigcirc$ $T7,T8$	$\bigcirc\!\!\bigcirc$ $T7,T9$	$\not\supseteq$ $T7$	$\not\supseteq$ $T7$
COL_{wp}	$\bigcirc\!\!\bigcirc$ $T7,T8$	$\bigcirc\!\!\bigcirc$ $T7,T9$	$\subset$ $T3$	$\not\supseteq$ $T7$
COL_{sp}	$\bigcirc\!\!\bigcirc$ $T7,T8$	$\bigcirc\!\!\bigcirc$ $T7,T9$	$\not\supseteq$ $T7$	$\not\supseteq$ $T7$
COL_b^T	$\subset$ $C3$	$\not\supseteq$ $C2$	$\subset$ $C1$	$\subset$ $C4$
COL_t^T	$\not\supseteq$ $C2$	$\not\supseteq$ $C2$	$\not\supseteq$ $C2$	$\not\supseteq$ $C2$
COL_{wp}^T	$\subset$ $C3$	$\not\supseteq$ $C2$	$\subset$ $C1$	$\not\supseteq$ $C2$
COL_{sp}^T	$\not\supseteq$ $C2$	$\not\supseteq$ $C2$	$\not\supseteq$ $C2$	$\not\supseteq$ $C2$
$0EC_{wp}$	$=$	$\bigcirc\!\!\bigcirc$ $T6$	$\subset$ $T4$	
$0EC_{ap}$	$\bigcirc\!\!\bigcirc$ $T6$	$=$		$\subset$ $T5$
EEC_{wp}	$\supset$ $T4$		$=$	
EEC_{ap}		$\supset$ $T5$		$=$

Table 1: Results from theorems and corollaries

[4] J. Dassow, J. Kelemen, Gh. Păun. On Parallelism in Colonies. *Cybernetics and Systems*, 24:37–49, 1993.

[5] J. Kelemen, A. Kelemenová. A Grammar-theoretic Treatment of Multiagent Systems. *Cybernetics and Systems*, 23:621–633, 1992.

[6] A. Kelemenová, E. Csuhaj-Varjú. Languages of Colonies. *Theoretical Computer Science,* 134:119–130, 1994.

[7] H. C. Kleijn, G. Rozenberg. A Study in Parallel Rewriting Systems. *Information and Control,* 44:134–163, 1980.

[8] G. Rozenberg, A. Salomaa. *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.

[9] A. Salomaa.*Formal Languages*. Academic Press, New York, 1973.

[10] Š. Vavrečková. Eko-kolonie. In J. Kelemen, V. Kvasnička, J. Pospíchal, editors, *Kognice a umělý život V*, pages 601–612. Silesian University, Opava, 2005.

[11] Š. Vavrečková. Eco-colonies. In L. Matyska, A. Kučera, T. Vojnar, Y. Kotásek, D. Antoš, editors, *MEMICS 2006, Proceedings of the 2nd Doctoral Workshop*, pages 253–259. University of Technology, FIT, Brno, 2006.

[12] Š. Vavrečková. Properties of Eco-colonies. In A. Kelemenová, D. Kolář, A. Meduna, J. Zendulka, editors, *Information Systems and Formal Models 2007*, pages 235–242. Silesian University, Opava, 2007.

Proceedings of the International Workshop, Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, pages 144 - 157, 2007.

On Stateless Automata and P Systems *

Linmin Yang[1], Zhe Dang[1], and Oscar H. Ibarra[2]

[1] School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA
lyang1@eecs.wsu.edu, zdang@eecs.wsu.edu

[2] Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract

We introduce the notion of stateless multihead two-way (respectively, one-way) NFAs and stateless multicounter systems and relate them to P systems and vector addition systems. In particular, we investigate the decidability of the emptiness and reachability problems for these stateless automata and show that the results are applicable to similar questions concerning certain variants of P systems, namely, token systems and sequential tissue-like P systems.

1 Introduction

There has been a flurry of research activities in the area of membrane computing (a branch of molecular computing) initiated seven years ago by Gheorghe Paun [8]. Membrane computing identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. It abstracts from the way living cells process chemical compounds in their compartmental structures. Thus, regions defined by a membrane structure contain objects that evolve according to given rules. The objects can be described by symbols or by strings of symbols, in such a way that multisets of objects are placed in regions of the membrane structure. The membranes themselves are organized as a Venn diagram or a tree structure where one membrane may contain other membranes. By using the rules in a nondeterministic and maximally parallel manner, transitions between the system configurations can be obtained. A sequence of transitions shows how the system is evolving. At a high-level, a P system has the following key features:

*The work of Zhe Dang and Linmin Yang was supported in part by NSF Grant CCF-0430531. The work of Oscar H. Ibarra was supported in part by NSF Grants CCF-0430945 and CCF-0524136, and a Nokia Visiting Fellow scholarship at the University of Turku.

- Objects are typed but addressless (i.e., without individual identifiers),
- Objects can transfer between membranes,
- Membranes themselves form a structure (such as a tree),
- Object transferring rules are in (either maximally or locally) parallel, and
- **The system is stateless**.

Biologically inspired computing models like P systems [9] are often stateless. This is because it is difficult and even unrealistic to maintain a global state for a massively parallel group of objects. Naturally, a membrane in a P system, which is a multiset of objects drawn from a given finite type set $\{a_1, \ldots, a_k\}$, can be modeled as having counters $x_1, \ldots, x_k$ to represent the multiplicities of objects of types $a_1, \ldots, a_k$, respectively. In this way, a P system can be characterized as a counter machine in a nontraditional form; e.g., without states, and with parallel counter increments/decrements, etc. The most common form of stateless counter machines are probably the Vector Addition Systems (VASs), which are well-studied. Indeed, VASs have been shown intimately related to certain classes of P systems [5]. However, with new applications of P systems in mind [10], the investigation of other classes of stateless automata deserve further investigation.

In this paper, we present some results in this direction, with applications to reachability problems for variants of P systems, namely, token systems and sequential tissue-like P systems.

2 Preliminaries

A nondeterministic multicounter automaton is a nondeterministic automaton with a finite number of states, a one-way input tape, and a finite number of integer counters. Each counter can be incremented by 1, decremented by 1, or stay unchanged. Besides, a counter can be tested against 0. It is well-known that counter machines with two counters have an undecidable halting problem. Thus, to study decidable cases, we have to restrict the behaviors of the counters. One such restriction is to limit the number of reversals a counter can make. A counter is *n-reversal-bounded* if it changes mode between nondecreasing and nonincreasing at most n times. For instance, the following sequence of counter values:

$$0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 3, 2, 1, 1, 1, 1, \ldots$$

demonstrates only one counter reversal. A counter is *reversal-bounded* if it is n-reversal-bounded for some fixed number n independent of computations. A *reversal-bounded nondeterministic multicounter automaton* is a nondeterministic multicounter automaton in which each counter is reversal-bounded.

Let Y be a finite set of variables over integers. For all integers a_y, with $y \in Y$, b and c (with $b > 0$), $\sum_{y \in Y} a_y y < c$ is an *atomic linear relation* on Y and $\sum_{y \in Y} a_y y \equiv_b c$ is a *linear congruence* on Y. A *linear relation* on Y is a Boolean combination (using $\neg$ and $\wedge$) of atomic linear relations on Y. A *Presburger formula* on Y is the

Boolean combination of atomic linear relations on Y and linear congruences on Y. A set P of tuples of nonnegative integers is *Presburger-definable* or a *Presburger relation* if there exists a Presburger formula $\mathcal{F}$ on Y such that P is exactly the set of the solutions for Y that make $\mathcal{F}$ true. It is well known that Presburger formulas are closed under quantification.

Let N be the set of nonnegative integers and n be a positive integer. A subset S of N^n is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in N^n such that $S = \{v \mid v = v_0 + a_1 v_1 + \ldots + a_t v_t, \forall 1 \leq i \leq t, a_i \in \mathrm{N}\}$. S is a *semilinear set* if it is a finite union of linear sets. It is known that S is a semilinear set if and only if S is Presburger-definable [2].

Let Σ be an alphabet consisting of n symbols $a_1, \ldots, a_n$. For each string (word) w in Σ^*, we define the *Parikh map* of w, denoted by $p(w)$, as follows:

$$p(w) = (i_1, \ldots, i_n), \quad \text{where } i_j \text{ is the number of occurrences of } a_j \text{ in } w.$$

If L is a subset of Σ^*, the *Parikh map* of L is defined by $p(L) = \{p(w) \mid w \in L\}$. L is a *semilinear language* if its Parikh map $p(L)$ is a semilinear set.

The following result is known [4]:

Theorem 1 *$p(L(M))$ is an effectively computable semilinear set when M is a reversal-bounded nondeterministic multicounter automaton.*

Consider a reversal-bounded nondeterministic multicounter machine (which is a reversal-bounded nondeterministic multicounter automaton without input). Let $(j, v_1, \ldots, v_k)$ denote the configuration of M when it is in state j and counter i has value v_i for $1 \leq i \leq k$. Define $R(M) = \{(\alpha, \beta) \mid$ configuration α can reach configuration β in 0 or more moves $\}$, which is called the reachability relation of M. Using Theorem 1, one can easily show that $R(M)$ is Presburger definable.

Theorem 2 *The reachability relation of a reversal-bounded nondeterministic multicounter machine is Presburger definable.*

An n-dimensional *vector addition system* (VAS) is specified by W, a finite set of vectors in $\mathbb{Z}^n$, where $\mathbb{Z}$ is the set of all integers (positive, negative, zero). For two vectors x and z in $\mathbb{N}^n$, we say that x can *reach* z if for some $j, z = x + v_1 + \ldots + v_j$, where, for all $1 \leq i \leq j$, each $v_i \in W$ and $x + v_1 + \ldots + v_i \geq 0$. The Presburger reachability problem for VAS is to decide, given two Presburger formulas P and Q, whether there are x satisfying P and z satisfying Q such that x can reach z. The following theorem follows from the decidability of the reachability problem for VASs (which are equivalent to Petri nets) [7].

Theorem 3 *The Presburger reachability problem for VAS is decidable.*

3 Stateless Multihead Two-way (One-way) NFAs/DFAs and Token Systems

Let Σ and Π be two alphabets. An object of some type in Σ (resp., Π) is called a *standard object* (resp., a *token*). Consider a chain (with length n) of membranes

(i.e., membranes are organized as a linear structure)

$$A_1, \ldots, A_n \tag{1}$$

for some n. The chain is called *initial* if the following conditions are met:

1 Each A_i holds exactly one standard object,

2 A_1 contains one token of each type in Π; the rest of the A_i's do not contain any tokens,

3 The standard object on the first membrane A_1 is of type $\rhd \in \Sigma$ and the standard object on the last membrane A_n is of type $\lhd \in \Sigma$; the membranes in between A_1 and A_n do not contain any $\rhd$-objects and $\lhd$-objects.

Let $\Pi' \subseteq \Pi$ be given. The chain is called *halting* if we change the condition 2 in above into "A_n contains one token of each type in Π'." A *rule* specifies how objects are transferred between two neighboring membranes (i.e., A_i and A_{i+1} for $1 \leq i \leq n-1$) and is in one of the following two forms:

- $(a, p)^{\rightarrow}$,
- $(a, p)^{\leftarrow}$,

where $a \in \Sigma$ and $p \in \Pi$. For instance, when $(a, p)^{\rightarrow}$ is applied on A_i, the A_i must contain a standard a-object and a p-token. The result is to move the token from A_i to A_{i+1} (where $1 \leq i \leq n-1$). The semantics of $(a, p)^{\leftarrow}$ is defined similarly but the token p moves from A_{i+1} back to A_i. We are given a set of rules R which are applied sequentially; i.e., each time, a rule and an i are nondeterministically picked and the rule is applied on A_i. We are interested in studying the computing power of such *token* systems. Specifically, we focus on decision algorithms solving the following reachability problem: whether there is an n and an initial chain with length n such that, after a certain number of applications of rules in R, the initial chain evolves into a halting chain. Notice that an instance of a chain in the form of (1) is a special instance of tissue-like P systems [6] and in the future we will further study more general intra-membrane structures (such as graphs) than linear structures in (1). Also note that, in the reachability problem, the chain is not given. Instead, we are looking for a desired chain. This is different from the case for a tissue-like P system where a concrete instance (with the n in (1) given) is part of the system definition.

One can generalize the aforementioned token systems by allowing some of the rules synchronized; i.e., a *synchronized rule* in the form of

$$[r_1, r_2, \ldots, r_k]$$

for some k and distinct rules $r_1, \ldots, r_k$. The semantics of the synchronized rule is to apply each r_i at the same time. For instance, a synchronized rule $[(a, p)^{\rightarrow}, (b, q)^{\leftarrow}]$, when applied, is to nondeterministically pick an A_i and A_j (where i could be the same as j) and apply the rule $(a, p)^{\rightarrow}$ on A_i and the rule $(b, q)^{\leftarrow}$ on A_j (whenever both are applicable). Such systems with synchronized rules are called *generalized*

token systems and we can raise the same reachability problem for generalized token systems.

We first observe that the (generalized) token systems are essentially the same as *stateless multihead two-way NFAs* M studied in the following, where each input tape cell corresponds to a membrane in (1) and each token corresponds to a two-way head. The stateless NFA M is equipped with an input on alphabet Σ and heads $H_1, \ldots, H_k$ for some k. The heads are two-way, the input is read-only, and there are no states. An H_i-move (also called a *local move*) MOVE_i of the NFA can be described as a triple (H_i, a, D), where H_i is the head involved in the move, a is the input symbol under the head H_i, and $D \in \{+1, -1, 0\}$ meaning that, as a result of the move, the head H_i goes to the right, goes to the left, or simply stays. When a head H_i tries to execute a local move (H_i, a, D), it requires that the symbol under H_i must be a, otherwise M just crashes. A generalized move is in the form of $(H_i, \mathcal{S}, \mathcal{D})$, where $\mathcal{S}$ is a set of symbols, and $\mathcal{D}$ is a set of directions (i.e., $+1$, -1, 0). When executing a generalized move $(H_i, \mathcal{S}, \mathcal{D})$, the symbol H_i reads must belong to $\mathcal{S}$, and H_i nondeterministically picks a direction from $\mathcal{D}$.

Note that a local move is a special case of a generalized move. An *instruction* of M is a sequence of local or general moves, in the form of [MOVE_{i_1}, MOVE_{i_2}, ..., MOVE_{i_m}], for some m, $1 \leq m \leq k$, and $i_1 < \ldots < i_m$. (If $m = 1$, the instruction is simply called a *local* instruction.) When the instruction is executed, the heads $H_{i_1}, \ldots, H_{i_m}$ perform the moves MOVE_{i_1}, ..., MOVE_{i_m}, respectively and simultaneously. Any head falling off the tape will cause M to crash. The NFA M has a finite set of such instructions. At each step, it nondeterministically picks a maximally parallel set of instructions to execute. M has a set of accepting heads $F \subseteq \{H_1, ..., H_k\}$. For most constructions in this paper, F consists of all the heads. Initially, all heads are at the leftmost cell of the input tape. *M halts and accepts* the input when the accepting heads are all at the rightmost cell. We assume that the input tape of M has a left end marker $\rhd$ and a right end marker $\lhd$. Thus, for any input $a_1 ... a_n$, $n \geq 2$, $a_1 = \rhd$, $a_n = \lhd$, and for $2 \leq i \leq n-1$, each a_i is different from the end markers.

We emphasize that in a stateless multihead one-way (two-way) NFA, at each step, the application of the instructions is "maximally parallel", i.e., all instructions that can be applied to the heads must be applied. Note that, in general, the set of instructions that can be applied maximally parallel need not be unique. If at most m instructions are applicable at each step, then we say the machine is m-maxpar.

A stateless one-way (two-way) DFA is one in which at each step of the computation, at most one maximally parallel set of instructions is applicable.

Our first result is the following:

Theorem 4 *The reachability problem for generalized token systems is undecidable. The problem is decidable for token systems.*

The first part of Theorem 4 is a direct consequence of the next theorem. The second part follows from the fact that the emptiness problem for two-way NFAs is decidable.

Theorem 5 *The emptiness problem for stateless (1-maxpar) 3-head one-way DFAs is undecidable.*

Proof. Given a deterministic TM Z, let $A_Z = C_1\#C_2\#...\#C_n$ be the halting computation of Z starting on blank tape. Hence C_1 is the initial configuration (on blank tape), C_n is the halting configuration, and C_{i+1} is the direct successor of C_i. We assume that $n \geq 2$. Let Γ and Q_Z be the tape alphabet and state set, respectively, of Z.

Clearly, from Z, we can construct a 2-head one-way DFA M_0 (with states) with heads H_1 and H_2 and input alphabet $\Sigma = \Gamma \cup Q_Z \cup \{\#\}$, which accepts a nonempty language (actually only the string A_Z) iff Z halts. Because from configuration C_i the next step of Z that results in configuration C_{i+1} may move its read-write head left, H_2 may not always move to the right at every step in M_0's computation. However, we can modify M_0 into another two-head one-way DFA M by putting "dummy" symbols α's on the tape so that H_2 can read these symbols instead of not moving right. H_1, of course, ignores these dummy symbols. M has now the property that H_2 always moves to the right at every step in the computation until M accepts. Clearly, $L(M_0) = \emptyset$ if and only if $L(M) = \emptyset$, and if and only if Z does not halt on blank tape. We may assume that M accepts with H_2 falling off the right end of the tape in a unique accepting state f. (This assumption on H_2 falling off the right end of the tape should not be confused with the condition that in a stateless automaton, a head falling off the tape will cause the machine to crash.) We also assume that there are no transitions from state f. Let Q_M be the state set of M.

Thus, if $\delta(q, a_1, a_2) = (p, d_1, d_2)$, then $d_2 = +1$. This transition is applicable if M is currently in the state q and the heads H_1 and H_2 are reading a_1 and a_2, respectively. When the transition is applied, H_2 moves right, H_1 moves right or remains stationary depending on whether d_1 is $+1$ or 0, and M enters state p.

We construct a stateless 3-head one-way DFA M' to simulate M. The heads of M' are H_1, H_2, and H_3. The input alphabet of M' is $(\Sigma \times Q_M \cup \{\rhd, \lhd\}\ )$ ($\rhd$ and $\lhd$ are left and right end markers for the input to M'.) The instructions of M' are as follows:

1. $[(H_1, \rhd, 0), (H_2, \rhd, 0), (H_3, \rhd, +1)]$.
2. $[(H_1, \rhd, +1), (H_2, \rhd, +1), (H_3, (a, q), +1)]$ for every $a \in \Sigma$ and every $q \in Q_M$.
3. Suppose $\delta(q, a_1, a_2) = (p, d_1, d_2)$ and $p \neq f$, then $[(H_1, (a_1, s), d_1), (H_2, (a_2, q), +1), (H_3, (b, p), +1)]$ is a rule for every $s \in Q_M$ and every $b \in \Sigma$.
4. Suppose $\delta(q, a_1, a_2) = (f, d_1, +1)$, then $[(H_1, (a_1, s), d_1), (H_2, (a_2, q), +1), (H_3, \lhd, 0)]$ is a rule for every $s \in Q_M$.
5. $[(H_1, (a, q), +1), (H_2, (b, p), +1), (H_3, \lhd, 0)]$ is a rule for every $a, b \in \Sigma$ and $q, p \in Q_M$.
6. $[(H_1, (a, q), +1), (H_2, \lhd, 0), (H_3, \lhd, 0)]$ is a rule for every $a \in \Sigma$ and $q \in Q_M$.

M' accepts if and only if all three heads are on $\lhd$. From the construction, it is clear that M' is a stateless 3-head one-way DFA, and $L(M) = \emptyset$ if and only if Z does not halt on blank tape. The result follows from the undecidability of the halting problem for TMs on blank tape. □

It is an interesting open question whether the 3 heads in the above theorem can be reduced to 2, even if the 2 heads are allowed to be two-way. Note that in the theorem, all 3 heads are involved in every instruction. We can strengthen this result by a more intricate construction. Define a stateless k-head one-way 2-move NFA (DFA) to be one where in every instruction, at most two heads are involved. Then we have:

Theorem 6 *The emptiness problem for stateless 3-head one-way 2-move DFAs is undecidable.*

Proof. Let M be the 2-head one-way DFA **with states** constructed in the previous proof. The transition $\delta(q, a, b) = (p, d_1, +1)$ of M can be represented by the tuple

$$[q, (H_1, a, d_1), (H_2, b, +1), p]$$

Suppose M has n such transitions, and we number them as $1, \ldots, n$. We may assume that M starts its computation with rule number 1.

Note that H_2 moves to the right at every step, and that M accepts with H_2 falling off the right end of the tape in a unique accepting state f and there are no transitions from state f.

We say that transition numbers i and j are compatible if they correspond to transition instructions $[q, (H_1, a, d_1), (H_2, b, +1), p]$ and $[p, (H_1, a', d_1'), (H_2, b', +1), r]$, respectively, for some states q, p, r, symbols a, a', b, b', and directions d_1, d_1'.

The input alphabet of M' is $(\Sigma \times N \times \Delta) \cup \{\rhd, \lhd\}$, where $N = \{1, \ldots, n\}$ (set of transition numbers of M) and $\Delta = \{\delta_1, \delta_2\}$ ($\rhd$ and $\lhd$ are the end markers of M'). The heads of M' are H_1, H_2, H_3. The instructions of M' are defined as follows:

1. $[(H_1, \rhd, +1)]$
2. $[(H_3, \rhd, 0), (H_2, \rhd, +1)]$
3. $[(H_3, \rhd, +1), (H_2, (c, 1, \delta_1), 0)]$ for every $c \in \Sigma$.

Suppose transition number k corresponds to $[q, (H_1, a, d_1), (H_2, b, +1), p]$. Then the following instructions are in M':

4. $[(H_3, (b, k, \delta_1), 0), (H_2, (b, k, \delta_1), +1]$
5. $[(H_3, (c, i, \delta_1), +1), (H_2, (d, i, \delta_2), 0)]$, for every $1 \leq i \leq n$ and every $c, d \in \Sigma$.
6. $[(H_1, (a, i, \delta), d_1), (H_2, (c, k, \delta_2), 0)]$, for every $1 \leq i \leq n$, every $c \in \Sigma$, and every $\delta \in \Delta$.
7. $[(H_3, (c, i, \delta_2), 0), (H_2, (c, i, \delta_2), +1)]$, for every $1 \leq i \leq n$ and every $c \in \Sigma$.
8. $[(H_3, (c, i, \delta_2), +1), (H_2, (d, j, \delta_1), 0)]$, for every $1 \leq i, j \leq n$ with i and j compatible, and every $c, d \in \Sigma$.
9. $[(H_3, (c, i, \delta_2), +1), (H_2, \lhd, 0)]$, every $c \in \Sigma$ and for every $1 \leq i \leq n$, with i corresponding to a transition of the form $[q, (H_1, a', d_1), (H_2, b', +1), f]$ for every state q and a', b' in Σ. (Note that f is the unique accepting state of M.)

10. $[(H_1, (c, i, \delta), +1), (H_3, \lhd, 0)]$, for every $1 \leq i \leq n$, every $c \in \Sigma$, and every $\delta \in \Delta$.

Define a homomorphism h that maps each symbol (α, i, δ) to (i, δ). Then we require that the homomorphic image of the input tape of M' (excluding the end markers) is a string in

$$(1, \delta_1)(1, \delta_2)\{(1, \delta_1)(1, \delta_2), ..., (n, \delta_1)(n, \delta_2)\}^*$$

so that a sequence of valid transitions can be executed properly. H_2 and H_3 are used for this purpose (i.e., to check the format).

M' accepts if and only if all heads are on the right end marker. From the construction, it is clear that $L(M') = \emptyset$ iff $L(M) = \emptyset$. The undecidability follows.

□

Next, we will study the emptiness problem when the inputs are restricted. Recall that a language is bounded if it is a subset of $a_1^* a_2^* \ldots a_k^*$ for some given symbols $a_1, \ldots, a_k$.

It is known [3] that if M is a multihead one-way NFA with states but with bounded input, the language it accepts is a semilinear set effectively constructable from M. In fact, this result holds, even if M has two-way heads, but the heads can only reverse directions from right to left or from left to right at most r times, for some fixed r independent of the input. It follows that Theorem 5 can not be strengthened to hold for one-way machines accepting bounded languages. However, for two-way machines, we can prove the following:

Theorem 7 *The emptiness problem for stateless 5-head two-way NFAs over bounded input is undecidable.*

Proof. We show how a stateless 5-head two-way NFA M' can simulate a 2-counter machine M. Suppose M has states $q_1, ..., q_n$, where we assume that $n \geq 3$, q_1 is the initial state, and q_n is the unique halting state. Assume that both counters are zero upon halting, and the number of steps is odd. The transition of M is of the form $\delta(q_i, s_1, s_2) = (q_j, d_1, d_2)$ where s_r (sign of counter r) $= 0$ or $+$ and d_r (change in counter r) $= +1$, 0, -1 for $r = 1, 2$.

A valid input to M' is a string of the form $\rhd q_1 q_2 ... q_n a^d \lhd$ for some $d \geq 1$. We construct a stateless 5-head two-way NFA M' (with heads H_1, H_2, H_3, C_1, C_2) to simulate M. We begin with the following instructions:

$[(H_1, \rhd, 0), (H_2, \rhd, +1)]$

$[(H_1, \rhd, +1), (H_2, q_1, +1)]$

$[(H_1, q_1, +1), (H_2, q_2, +1)]$

...

...

$[(H_1, q_{n-1}, +1), (H_2, q_n, +1)]$

$[(H_1, q_n, +1), (H_2, a, +1)]$

$[(H_1, a, +1), (H_2, a, +1)]$

$[(H_1, a, +1), (H_2, \lhd, 0)]$

The instructions above check that the input is of the form $\rhd q_1 q_2 ... q_n a^d \lhd$ for some $d \geq 1$. At the end of the process H_1 and H_2 are on the right end marker $\lhd$. Next add the following instructions:

$[(H_1, \lhd, -1), (H_2, \lhd, -1), (H_3, \rhd, +1)]$

$[(H_1, t, -1), (H_2, t, -1), (H_3, q_1, 0)]$ for all $t \neq q_1$

$[(H_1, q_1, 0), (H_2, q_1, 0), (H_3, q_1, +1)]$

$[(H_2, q_k, +1), (H_3, q_2, 0)]$ for $1 \leq k \leq n-1$

$[(H_2, q_k, -1), (H_3, q_2, 0)]$ for $2 \leq k \leq n$

$[(H_1, q_k, +1), (H_3, q_3, 0)]$ for $1 \leq k \leq n-1$

$[(H_1, q_k, -1), (H_3, q_3, 0)]$ for $2 \leq k \leq n$

In the instructions above, if the symbol under H_3 is q_2 (resp., q_3) , the machine positions H_2 (resp., H_1) to some nondeterministically chosen state (for use below).

Let C_1 and C_2 be the counters of M. Initially they are set to zero. In the instructions below, we use heads C_1 and C_2 to correspond to the counters.

If $\delta(q_i, s_1, s_2) = (q_j, d_1, d_2)$ where $s_r = 0$ or $+$ and $d_r = +1, 0, -1$, then add the following instructions:

$[(H_1, q_i, 0), (H_2, q_j, 0), (H_3, q_2, +1), (C_1, t_1, d_1), (C_2, t_2, d_2)]$

$[(H_1, q_j, 0), (H_2, q_i, 0), (H_3, q_3, -1), (C_1, t_1, d_1), (C_2, t_2, d_2)]$
where $t_r = \rhd$ if $s_r = 0$ and $t_r \neq \rhd$ if $s_r = +$

If $\delta(q_i, 0, 0) = (q_n, 0, 0)$ (i.e., the 2-counter machine M halts in the unique state q_n with both counters zero after an odd number of steps), then add the following instructions:

$[(H_1, q_i, 0), (H_2, q_n, 0), (C_1, \rhd, +1), (C_2, \rhd, +1)]$

$[(H_1, q_i, 0), (H_2, q_n, 0), (C_1, t, +1), (C_2, t, +1)]$ for all $t \neq \lhd$

$[(H_r, t, +1), (C_1, \lhd, 0), (C_2, \lhd, 0)]$ for all $t \neq \lhd$ and for $r = 1, 2, 3$

M' accepts if all heads $(H_1, H_2, H_3, C_1, C_2)$ are on the right end marker. It is easily verified that M' accepts the empty set if and only if M does not halt. □

The above theorem says that the emptiness problem is undecidable if the number of heads is 5 (i.e., fixed) but the size of the input alphabet is arbitrary. The next result shows that the emptiness problem is also undecidable if the size of input alphabet is fixed (in fact, can be unary) but the number of heads is arbitrary.

Theorem 8 *The emptiness problem for stateless multihead two-way NFAs is undecidable even when the input is unary but with the left end marker (resp., right end marker).*

Proof. We only show the case with the left end marker. (The construction can easily be modified for the case with the right end marker.) We assume that the input has length at least 1, excluding the left end marker. We use a stateless NFA M' (whose input is unary with a left end marker) to simulate a two-counter machine M with counters C_1 and C_2, and states $q_0, q_1, \ldots, q_n$. The idea is to use $\lceil \log_2(n+1) \rceil$ heads, $H_1, \ldots, H_{\lceil \log_2(n+1) \rceil}$, to control the states, and another two heads $H_{\lceil \log_2(n+1) \rceil+1}$ and $H_{\lceil \log_2(n+1) \rceil+2}$ to control the value of counters C_1 and C_2. The two-counter machine M starts in state q_0, and $C_1 = C_2 = 0$. Initially, all heads of M' are at the leftmost cell (i.e., $\rhd$). If H_i, $1 \leq i \leq \lceil \log_2(n+1) \rceil$, is at $\rhd$, we consider it as 0; if H_i is at the first a, we consider it as 1. Hence $H_{\lceil \log_2(n+1) \rceil} \ldots H_1$ is a binary string (with $H_{\lceil \log_2(n+1) \rceil}$ as its most significant bit), which is used to encode the index of a state of M. Note that during the computation, H_i, $1 \leq i \leq \lceil \log_2(n+1) \rceil$, only moves between the left end marker and the first a. We omit the details of the simulation of the instructions of M. □

Theorems 7 and 8 are best possible, since we can not fixed both the number of heads and the size of the input alphabet and get undecidability. This is because for fixed k and n, there are only a finite number of such stateless machines (also observed by Artiom Alhazov). Hence, the emptiness problem has a finite number of instances and therefore decidable.

A special case of Theorem 8 is when the input is unary and without end markers. In this case, the heads are initially at the leftmost input cell and the automaton accepts when the heads are all at the rightmost cell (we assume that there are at least two cells on the input). Using a VAS to simulate the multihead position changes,we have:

Theorem 9 *The emptiness problem for stateless multihead two-way NFAs is decidable when the input is unary and without end markers.*

Proof. Suppose we have a stateless NFA M with $H_1, \ldots, H_k$ for some k, and with unary input

$$a \ldots a,$$

of length B for some B. We can construct a corresponding VAS $G = \langle x, W \rangle$, where $x \in \mathbb{N}^k$ is the start vector, and W is a finite set of vectors in $\mathbb{Z}^k$. Furthermore, we require that the maximal entry of any vector in G can not exceed $B-1$; otherwise, G crashes. In other words, G is a bounded vector addition system. Since initially in M, all heads are at the leftmost cell, we set $x = (0, \ldots, 0)$ in G. If in M, there is an instruction $I =$[MOVE$_{i_1}$, MOVE$_{i_2}$, ..., MOVE$_{i_m}$], for some m, $1 \leq m \leq k$, and MOVE$_{i_j}$=(H_{i_j}, a, D), $1 \leq j \leq m$, then in G we have $v \in W$. For all j, $1 \leq j \leq m$, the i_j^{th} entry of v is 0 if $D = 0$. The entry is 1 if $D = +1$. And, the entry is -1 if $D = -1$. All other entries are 0.

Clearly, M accepts a nonempty language iff, for some B, $(B-1, \ldots, B-1)$ is reachable in G; directly from Theorem 3, the result follows. □

4 Sequential Tissue-Like P Systems and Stateless Multicounter Systems

We now generalize the rules in a token system by allowing multiple objects to transfer from one membrane to another in (1); the result is a variant of a tissue-like P system [6] with sequential applications of rules [1]. More precisely, let $\Sigma = \{a_1, \ldots, a_m\}$. A *sequential tissue-like P system* G is a directed graph with n nodes (for some n), where each node i is equipped with a membrane A_i which is a multiset of objects in Σ. In particular, we use m counters $\vec{X}_i = (x_{i1}, \ldots, x_{im})$ to denote the multiplicities of objects of types $a_1, \ldots, a_m$ in A_i, respectively. Each membrane A_i is also associated with a Presburger formula P_i, called a *node constraint*, over the m counters. Each edge (say, from node i to node j) in G is labeled with an *addition vector* $\vec{\Delta}_{ij}$ in N^m. Essentially, G defines a stateless multicounter system whose semantics is as follows. Intuitively, G specifies a multicounter system with n groups of counters with each group $\vec{X}_i$ of m counters. In the system, there are no states. Each instruction is the addition vector $\vec{\Delta}_{ij}$ specified on an edge. The semantics of the instruction, when applied, is to decrement counters in group $\vec{X}_i$ by $\vec{\Delta}_{ij}$ and increment counters in group $\vec{X}_j$ by $\vec{\Delta}_{ij}$ (we emphasize the fact that each component in the vector $\vec{\Delta}_{ij}$ is nonnegative by definition). When the system runs, an instruction is nondeterministically chosen and applied. Additionally, we require that at any moment during a run, for each i, the constraint P_i is true when evaluated on the counter values on group $\vec{X}_i$. Formally, a configuration is a tuple of n vectors $(\vec{V}_1, \ldots, \vec{V}_n)$ with each $\vec{V}_i \in \mathrm{N}^m$ satisfying the node constraint $P_i(\vec{V}_i)$. Given two configurations $\mathcal{C} = (\vec{V}_1, \ldots, \vec{V}_n)$ and $\mathcal{C}' = (\vec{V}'_1, \ldots, \vec{V}'_n)$, we say that $\mathcal{C}$ can reach $\mathcal{C}'$ in a *move*, written $\mathcal{C} \ \rightarrow_G \ \mathcal{C}'$, if there is an edge from node i to node j (for some i and j) such that $\mathcal{C}$ and $\mathcal{C}'$ are exactly the same except that $\vec{V}_i - \vec{\Delta}_{ij} = \vec{V}'_i$ and $\vec{V}_j + \vec{\Delta}_{ij} = \vec{V}'_j$. We say that $\mathcal{C}$ can *reach* $\mathcal{C}'$ in G, written

$$\mathcal{C} \ \rightsquigarrow_G \ \mathcal{C}'$$

if, for some t,

$$\mathcal{C}_0 \rightarrow_G \mathcal{C}_1 \ \ldots \ \rightarrow_G \mathcal{C}_t$$

where $\mathcal{C} = \mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_t = \mathcal{C}'$ are configurations. We now study the following *reachability problem*: given a G and two Presburger formulas P and Q, whether there are $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \ \rightsquigarrow_G \ \mathcal{C}'$ and, $\mathcal{C}$ and $\mathcal{C}'$ satisfy P and Q, respectively.

One can show that the reachability problem is undecidable even under a special case:

Theorem 10 *The reachability problem for sequential tissue-like P systems G is undecidable even when G is a DAG.*

We now consider the case when G is *atomic*; i.e., each node constraint P_i in G is a conjunction of atomic linear constraints, i.e., P_i is in the form of

$$\bigwedge(\sum_j a_{ij}x_{ij} \ \# \ c_i),$$

where $\# \in \{\leq, \geq\}$, a_{ij} and c_i are integral constants. Using a VAS to simulate an atomic sequential tissue-like P system, one can show:

Theorem 11 *The reachability problem for atomic sequential tissue-like P systems is decidable.*

In fact, the converse of Theorem 11 can be shown, i.e., atomic sequential tissue-like P systems and VAS are essentially equivalent, in the following sense. Consider a VAS M with k counters $(x_1, \ldots, x_k)$ and a sequential tissue-like P system G with a distinguished node on which the counters are $(z_1, \ldots, z_l; x_1, \ldots, x_k)$. We further abuse the notation $\rightsquigarrow_G$ as follows. We say that $(z_1, \ldots, z_l; x_1, \ldots, x_k)$ *reaches* $(z'_1, \ldots, z'_l; x'_1, \ldots, x'_k)$ in G if there are $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C} \rightsquigarrow_G \mathcal{C}'$ and, $\mathcal{C}$ and $\mathcal{C}'$, when projected on the distinguished node, are $(z_1, \ldots, z_l; x_1, \ldots, x_k)$ and $(z'_1, \ldots, z'_l; x'_1, \ldots, x'_k)$, respectively. We say that M can be *simulated* by G if, for all $(x_1, \ldots, x_k)$ and $(x'_1, \ldots, x'_k)$ in N^k, $(x_1, \ldots, x_k)$ reaches $(x'_1, \ldots, x'_k)$ in M iff $(0, ..., 0; x_1, \ldots, x_k)$ reaches $(0, ..., 0; x'_1, \ldots, x'_k)$ in G. We say that G is *simple* if each constraint P_i in G is a conjunction of $x_{ij} \geq c$ (open constraint) or $x_{ij} \leq c$ (closed constraint), for some constant c and every j, where x_{ij} is a counter in node i . Notice that if G is simple then G must be atomic also. One can show:

Theorem 12 *Every VAS can be simulated by a sequential tissue-like P system G that is a DAG and simple.*

For a sequential tissue-like P system G, a very special case is that there is only one counter in each node, and the instruction on an edge (i, j) is $I = 1$, which means that when I is executed, counter i is decremented by 1, and counter j is incremented by 1. We call such a G *single*. One can show that the reachability relation $\rightsquigarrow_G$ is Presburger definable if G is a DAG and single. The proof technique is to "regulate" the reachability paths in G and use reversal-bounded counter machine arguments, and then appeal to Theorem 2.

Theorem 13 *The reachability relation $\rightsquigarrow_G$ is Presburger definable when sequential tissue-like P system G is a DAG and single.*

Currently, we do not know whether Theorem 13 still holds when the condition of G being a DAG is removed.

As we pointed out, sequential tissue-like P systems are essentially stateless. To conclude this section, we give an example where some forms of sequential tissue-like P systems become more powerful when states are added, and hence states matter (In contrast to this, VAS and VASS (by adding states to VAS) are known to be equivalent).

Consider a sequential tissue-like P system G where each node contains only one counter and, furthermore, G is a DAG. From Theorem 13, its reachability relation $\rightsquigarrow_G$ is Presburger definable. We now add states to G and show that the reachability relation now is not necessarily Presburger definable. G with states is essentially a multicounter machine M with k counters $(x_1, \ldots, x_k)$ and each counter is associated with a *simple* constraint defined earlier. Each instruction in M is in the following form:

$$(s_p, x_i, x_{i+1}, s_q)$$

where $1 \leq i < k$ and, s_p and s_q are the states of M before and after executing the instruction. When the instruction is executed, x_i is decremented by 1, x_{i+1}

is incremented by 1, and the simple constraint on each counter should be satisfied (before and after the execution).

Now, we show that an M can be constructed to "compute" the inequality $x * y \geq z$, which is not Presburger definable. We need 8 counters, $x_1, \ldots, x_8$ in M. The idea is that we use the initial value of x_3, x_5 and x_7 to represent x, y and z, respectively, and the remaining counters are auxiliary. In particular, x_1 acts as a "warehouse" for supplying counter values. The constraint upon every counter is simply $x_i \geq 0$, $1 \leq i \leq 8$. Initially, the state is s_0, $x_2 = 0$, and all the other counters store some values. We have the following instructions:

$$I_1 = (s_0, x_3, x_4, s_1);$$

$$I_2 = (s_1, x_1, x_2, s_2);$$

$$I_3 = (s_2, x_7, x_8, s_0);$$

$$I_4 = (s_0, x_5, x_6, s_3);$$

$$I_5 = (s_3, x_2, x_3, s_0);$$

$$I_6 = (s_0, x_2, x_3, s_0).$$

Note that s_3 is the accepting state. I_1, I_2 and I_3 mean that, when x_3, which represents x, is decremented by 1, x_2 will record the decrement, and x_7, which represents z, will also be decremented by 1. I_4 says that during the decrement of x_3, x_5, which represents y, will be nondeterministically decremented by 1. I_5 and I_6 will restore the value of x_3, and after the restoration, the value of x_3 can never surpass the initial one (i.e., x). One can show that $x * y \geq z$ iff M can reach state s_3 (the accepting state) and, at the moment, $x_7 = 0$.

5 Conclusion

We introduced the notion of stateless multihead two-way (respectively, one-way) NFAs and stateless multicounter systems and related them to P systems and vector addition systems. In particular, we investigated the decidability of the emptiness and reachability problems for these stateless automata and showed that the results are applicable to similar questions concerning certain variants of P systems, namely, token systems and sequential tissue-like P systems. Many issues (e.g., the open problems mentioned in the previous sections) remain to be investigated, and we plan to look at some of these in future work.

References

[1] Z. Dang and O. H. Ibarra. On one-membrane P systems operating in sequential mode. *Int. J. Found. Comput. Sci.*, 16(5):867–881, 2005.

[2] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. of Mathematics*, 16:285–296, 1966.

[3] O. H. Ibarra. A note on semilinear sets and bounded-reversal multihead pushdown automata. *Inf. Processing Letters*, 3(1): 25-28, 1974.

[4] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978.

[5] O. H. Ibarra, Z. Dang, and Ö. Egecioglu. Catalytic P systems, semilinear sets, and vector addition systems. *Theor. Comput. Sci.*, 312(2-3):379–399, 2004.

[6] C. Martín-Vide, Gh. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theor. Comput. Sci.*, 296(2):295–326, 2003.

[7] E. Mayr. An algorithm for the general Petri net reachability problem. *Proc. 13th Annual ACM Symp. on Theory of Computing*, 238–246, 1981.

[8] Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

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

[10] L. Yang, Z. Dang, and O.H. Ibarra. Bond computing systems: A biologically inspired and high-level dynamics model for pervasive computing. Proceedings of the 6th International Conference on Unconventional Computation (UC'07), Lecture Notes in Computer Science, 2007.

Author Index