Computational completeness of minimal communication with small number of cells

Erzsébet Csuhaj-Varjú[1] and Sergey Verlan[2]

[1] Department of Algorithms and Their Applications,
Faculty of Informatics,
ELTE Eötvös Loránd University, Budapest,
Pázmány Péter sétány 1/c, 1117, Hungary
csuhaj@inf.elte.hu
[2] Univ Paris Est Creteil, LACL, F-94010, Creteil, France
verlan@u-pec.fr

Abstract. Generalized communicating P systems (GCPSs for short) are a common generalization of tissue-like P systems (networks of cells) where each interaction rule can move only two objects through the cells. Depending on the source and target cells, nine types of such rules are distinguished. Previous works have shown that several GCPSs families where the GCPSs use only one type of rules and have only three cells are computationally complete computing devices. In this paper, we show that GCPSs with only parallel-shift rules and only four cells, and GCPSs with only presence-move rules and only three cells are computationally complete as well. With these new results, we contribute to the research goal of providing a sharp lower bound on the number of cells needed to achieve computational completeness for all families of GCPSs where GCPSs use only one type of interaction rules.

1 Introduction

Purely communicating P systems are membrane systems that operate with moving objects across regions or cells and between regions and their common environment. These models have been studied for a long time, since many of these P system versions are computationally complete, i.e. in their case, to achieve computational completeness, no evolution is needed, only communication. This computing power is ensured by the fact that, in order to perform a state transition, the necessary number of copies of a given object type in the environment is always available for communication between the P system and the environment. About purely communicating P systems, the reader can find information in the book [9].

An important variant of purely communicating P systems is the generalized communicating P system (or the GCPSs), introduced in [18], with the aim of providing a common generalization for the purely communicating P system models.

A generalized communicating P system is a P system having a hypergraph structure (a network of cells) where each node represents a cell and each hyperedge corresponds to a rule. Each node contains a multiset of objects that can be communicated. The system is embedded in an environment, considered as cell 0. The environment may have certain objects in an arbitrary number of copies and certain objects only in a finite number of copies.

The cells of the generalized communicating P system and the environment interact by using the communication (interaction) rules. Communication means a move of objects between the cells according to prescribed interaction (communication) rules.

The form of an interaction rule is $(i, a)(j, b) \to (k, a)(l, b)$ where a and b are objects and i, j, k, l are labels for the input and the output cells. Such a rule means that an object a from cell i and an object b from cell j move synchronously (in one step) to cell k and cell l, respectively. The reader may notice the simple form of the rules, i.e., they describe the movement of only two objects. In the case of the communication between the P system and the environment, there is a restriction, namely, that at every computation step from the environment only a finite number of objects is allowed to enter any cell.

In each computation step, the rules are applied in a maximally parallel manner, i.e. a multiset of rules with maximal number of elements is performed at the same time. This derivation variant is common in P systems theory.

A computation step may change the multisets of objects in the cells, which form the configuration of the GCPS. These multisets represent the contents of the cells. A computation in a GCPS is a sequence of configurations directly following each other, starting from the initial configuration to a halting configuration. The result of the computation is the number of objects in a distinguished cell, called the output cell.

The generalized communicating P system has been inspired and can be considered as generalization of several known models in bio-inspired computing. For example, such model is the symport/antiport P system [14] that provides a formalization of the biological process of co-transport in terms of membrane systems. A simplification of the symport/antiport P system, called the P system with conditional-uniport rules was introduced in [17] to model communication through ion channels. The GNPS model, defined in [18], aimed at unifying the definition of the previous two concepts. In this model more complex rules are defined, which correspond to hypergraph communication. This concept inspired the formal framework for P systems [11] which captures many formal features of P system variants [10,6,19,20]. The concept of the GCPS is also closely related to Petri nets [4,3].

One of the most challenging problems is the minimum number of cells the GPCS should have for a given computational power and to what extent its interaction rules can be simplified.

It has been shown that generalized communicating P systems with restrictions on the form of rules are able to generate any recursively enumerable set of numbers. In [5], nine possible restrictions on the interaction rules (modulo

symmetry) were distinguished, called GCPSs with minimal interaction: split, join, symport2, antiport1, chain, conditional-uniport-in, conditional-uniport-out, parallel-shift, and presence move.

Furthermore, computational completeness can be obtained with a small number of cells and a simple underlying hypergraph architecture [7,5,4,9,17,18,12]. For example, GCPSs with only three cells and with only join rules, or only split rules, or only chain rules are computationally complete [7]. In these cases, any rule operates with three cells. It is also shown that GCPSs with only conditional-uniport-in rules, only two cells and their common environment are Turing complete. Furthermore, there exist $k > 0$ such that the family of recursively enumerable sets of integers greater than or equal to k is equal to the family of sets of numbers generated by GCPSs with only conditional uniport-out rules, only two cells and their common environment. These two latter results were presented in [8]. It is also shown that the maximal computational power can also be obtained if the alphabet of objects of the GPCSs is a singleton [4].

In this paper, we prove that GCPSs with only parallel-shift rules and only four cells, and GCPSs with only presence-move rules and only three cells are computationally complete. With these results, we contribute to the research goal of providing a sharp lower bound on the number of cells needed to achieve computational completeness for all families of GCPSs where GCPSs use only one type of interaction rules. The presence-move rule moves the object b from cell j to cell l, provided that there is an object a in cell i and i, j, l are pairwise different cells. For pairwise different numbers i, j, k, l, the parallel-shift rule moves objects a and b from two different cells to another two different cells.

The paper is organized as follows. Section 2 gives the definitions and introduces the model. The two main results are presented in Section 3. Finally, in Section 4 open problems for the further research are suggested.

2 Definitions

The reader is supposed to be familiar with formal language theory and membrane computing; for further details consult [16] and [15]. NRE denotes the family of recursively enumerable sets of natural numbers. $\mathbb{N}_k$ denotes the set of natural numbers greater or equal to k and N_kRE denotes the family of recursively enumerable sets of numbers with numbers greater than or equal to k.

For a finite multiset of symbols X over an alphabet V, $supp(X)$ denotes the set of symbols in X (the support of X) and $|X|$ denotes the total number of its symbols (its size). The number of occurrences of symbol x in X is denoted by $|X|_x$.

Throughout the paper, every finite multiset X is presented as a string w, where X and w have the same number of occurrences of symbol a, for each $a \in V$. The empty multiset is denoted by λ.

If no confusion arises, then the set of all finite multisets over V is denoted by V^*.

A counter automaton is a 5-tuple $M = (Q, \mathcal{R}, q_0, q_f, P)$, where Q is a finite non-empty set, called the set of states, $\mathcal{R} = \{A_1, \ldots, A_k\}$, $k \geq 1$, is a set of counters, called also registers, $q_0 \in Q$ is the initial state, and $q_f \in Q$ is the final state. P is a set of instructions of the following forms: $(p, A+, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in \mathcal{R}$, called an increment instruction, or $(p, A-, q)$, where $p, q \in Q, p \neq q_f, A \in \mathcal{R}$, called a decrement instruction, or $(p, A0, q)$, where $p, q \in Q, p \neq q_f, A \in \mathcal{R}$, called a zero-check instruction. Without loosing the generality, it can be supposed that for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q)$, or $(p, A0, q)$.

A configuration of a counter automaton M, defined above, is a $(k+1)$-tuple $(q, m_1, \ldots, m_k)$, where $q \in Q$ and $m_1, \ldots, m_k$ are non-negative integers; q is the current state of M and $m_1, \ldots, m_k$ are the current numbers stored in the registers (the current contents of the registers or the value of the registers) $A_1, \ldots A_k$, respectively.

A transition of the counter automaton consists in executing an instruction. An increment instruction $(p, A+, q, s) \in P$ is performed if M is in state p, the number stored in register A is increased by 1, and after that M enters either state q or state s, chosen non-deterministically. A decrement instruction $(p, A-, q) \in P$ is performed if M is in state p, and if the number stored in register A is positive, then it is decreased by 1, and then M enters state q. If the number stored in register A is zero, then the computation "blocks" and the corresponding non-deterministic computation branch is considered to fail. A zero-check instruction $(p, A0, q) \in P$ is performed if M is in state p, and if the number stored in register A is 0, then the contents of A remains unchanged and M enters state q. If the contents of register A is not zero, then the computation "blocks" and the corresponding non-deterministic computation branch is considered to fail.

A counter automaton $M = (Q, \mathcal{R}, q_0, q_f, P)$, with k registers, given as above, generates a non-negative integer n, if starting from the initial configuration $(q_0, 0, 0, \ldots, 0)$ it enters (in a non-deterministic manner) the final configuration $(q_f, n, 0, \ldots, 0)$. The set of non-negative integers generated by M is denoted by $N(M)$.

We remark that counter automata are very closely related to register machines [13]. In fact, a in register machine the operations of minus and zero check are combined in a single instruction $(p, A-, r, s)$ that corresponds to instructions $(p, A-, r)$ and $(p, A0, s)$ of the counter automaton.

Next we recall the basic definitions concerning generalized communicating P systems [18].

A generalized communicating P system (a GCPS) of degree n, where $n \geq 1$, is an $(n+4)$-tuple $\Pi = (O, E, w_1, \ldots, w_n, R, h)$ where

1. O is an alphabet, called the set of objects of Π;
2. $E \subseteq O$; called the set of environmental objects of Π;
3. $w_i \in O^*$, $1 \leq i \leq n$, is the multiset of objects initially associated to cell i;
4. R is a finite set of interaction rules or communication rules of the form $(i, a)(j, b) \to (k, a)(l, b)$, where $a, b \in O$, $0 \leq i, j, k, l \leq n$, and if $i = 0$ and

$j = 0$, then $\{a, b\} \cap (O \setminus E) \neq \emptyset$; i.e., at least one of a and b is not element of E.

5. $h \in \{1, \ldots, n\}$ is the output cell.

The system consists of n cells, labeled by natural numbers from 1 to n, which contain multisets of objects over O. Initially, cell i contains multiset w_i (the initial contents of cell i is w_i). An additional special cell, labeled by 0 and called the environment is distinguished. The environment contains objects of E in an infinite number of copies.

The cells interact by means of the rules $(i, a)(j, b) \to (k, a)(l, b)$, with $a, b \in O$ and $0 \leq i, j, k, l \leq n$. As the result of the application of the rule, object a moves from cell i to cell k and b moves from cell j to cell l. If two objects from the environment move to some other cell or cells, then at least one of them must not appear in the environment in an infinite number of copies.

The structure of the system is a hypergraph implicitly deduced from the set of rules. Indeed, a rule $(i, a)(j, b) \to (k, a)(l, b)$ induces an hyperedge $\{i, k, j, l\}$. A generalization of GNPS using more cells in a rule and also performing the rewriting lead to the notion of the formal framework for P systems [11].

A configuration of a GCPS Π, as above, is an $(n + 1)$-tuple $(z_0, z_1, \ldots, z_n)$ with $z_0 \in (O \setminus E)^*$ and $z_i \in O^*$, for all $1 \leq i \leq n$; z_0 is the multiset of objects present in the environment in a finite number of copies, whereas, for all $1 \leq i \leq n$, z_i is the multiset of objects present inside cell i. The initial configuration of Π is the $(n + 1)$-tuple $(\lambda, w_1, \ldots, w_n)$.

Given a multiset of rules $\mathcal{R}$ over R and a configuration $u = (z_0, z_1, \ldots, z_n)$ of Π, we say that $\mathcal{R}$ is applicable to u if all its elements can be applied simultaneously to the objects of multisets $z_0, z_1, \ldots, z_n$ such that every object is used by at most one rule. If there is no multiset $\mathcal{R}'$ where $\mathcal{R}$ is a proper submultiset of $\mathcal{R}'$ can be applied to configuration $u = (z_0, z_1, \ldots, z_n)$ of Π, then a new configuration $u' = (z'_0, z'_1, \ldots, z'_n)$ is obtained by applying $\mathcal{R}$ in a non-deterministic maximally parallel manner.

One such application of a multiset of rules satisfying the conditions listed above represents a transition in Π from configuration u to configuration u'. A transition sequence is said to be a successful generation by Π if it starts with the initial configuration of Π and ends with a halting configuration, i.e., with a configuration where no further transition step can be performed.

In this paper, we deviate from the standard notation of the configuration to make it easier to follow the movement between cells. Instead of $u = (z_0, z_1, \ldots, z_n)$, we will use the notation $u = (0, z_0)(1, z_1) \cdots (n, z_n)$. The numbers $1, \ldots, n$ refer to the label of the cell, and z_i refers to the contents of cell i.

Π generates a non-negative integer n if there is a successful generation by Π such that n is the size of the multiset of objects present inside the output cell in the halting configuration. The set of non-negative integers generated by a GCPS Π in this way is denoted by $N(\Pi)$. If instead of counting all the objects present inside the output cell in the halting configuration at the end of successful generations of Π we consider only the number of objects from a nonempty subset $O' \subseteq O$, then we denote the corresponding set of numbers generated by $N_{O'}(\Pi)$.

In the following we recall the notions of the possible restrictions on the interaction rules (modulo symmetry). We distinguish the following cases, called GCPSs with minimal interaction:

1. $i = j = k \neq l$: the conditional-uniport-out rule (the *uout* rule) sends b to cell l provided that a and b are in cell i [17];
2. $i = k = l \neq j$: the conditional-uniport-in rule (the *uin* rule) brings b to cell i provided that a is in that cell [17];
3. $i = j$, $k = l$, $i \neq k$: the symport2 rule (the $sym2$ rule) corresponds to the minimal symport rule [15], i.e., a and b move together from cell i to k;
4. $i = l$, $j = k$, $i \neq j$: the antiport1 rule (the $anti1$ rule) corresponds to the minimal antiport rule [15], i.e., a and b are exchanged in cells i and k;
5. $i = k$ and $i \neq j$, $i \neq l$, $j \neq l$: the presence-move rule (the $presence$ rule) moves the object b from cell j to l, provided that there is an object a in cell i and i, j, l are pairwise different cells;
6. $i = j$, $i \neq k$, $i \neq l$, $k \neq l$: the *split* rule sends a and b from cell i to cells k and l, respectively;
7. $k = l$, $i \neq j$, $k \neq i$, $k \neq j$: the $join$ rule brings a and b together to cell k;
8. $l = i$, $i \neq j$, $i \neq k$ and $j \neq k$: the *chain* rule moves a from cell i to cell k while b is moved from cell j to cell i, i.e., to the cell where a located previously;
9. i, j, k, l are pairwise different numbers: the parallel-shift rule (the $shift$ rule) moves a and b from two different cells to another two different cells.

$NOtP_n(x)$ denotes the set of numbers generated by generalized communicating P systems with minimal interaction of degree n, $n \geq 1$, and with rules of type x, where $x \in \{uout, uin, sym2, anti1, presence, split, join, chain, shift\}$. $NOtP_*(x)$ is the notation for $\bigcup_{n=1}^{\infty} NOtP_n(x)$.

3 Results

We shall start with the case of GCPSs having only parallel shift rules. We show that every recursively enumerable set of non-negative integers can be computed by a GCPSs with only parallel shift rules and four cells. Notice that to apply the parallel shift rule four cells are involved.

Theorem 1. $NOtP_4(pshift) \supseteq NRE$.

Proof. To prove that any recursively enumerable set of non-negative integers can be generated by a GCPS with four cells and with only parallel shift rules, we show that to any register machine M we can construct a simulating GCPS Π with four cells and only parallel shift rules.

Let $M = (Q, \mathcal{R}, q_0, q_f, P)$ be a register machine, with $\mathcal{R} = \{A_1, \ldots, A_k\}$, $k \geq 1$, defined as in Section 2. We give a GCPS $\Pi = (O, E, w_1, w_2, w_3, w_4, R, 1)$ with only parallel shift rules such that any halting transition sequence of Π corresponds to a halting transition sequence of M. Furthermore, the numbers generated by these two transition sequences are the same.

Let Π have the following components. (Since it is clear from the context, we use the symbol A_j also in case of Π).

$O = Q \cup \{A_j \mid 1 \leq j \leq k\} \cup \{p_1 \mid p \in Q\} \cup \{p_2 \mid (p, A_j-, q, s) \in P\}$.

$E = \{A_j \mid 1 \leq j \leq k\}$.

We note that integer n, $n \geq 0$ stored in register A_j in M, $1 \leq j \leq k$, is represented by A_j^n in Π.

We give the initial configuration of Π. Let $w_1 = \{q_0\}$, $w_2 = \{p_2 \mid (p, A-, q, s) \in P\}$, $w_3 = \emptyset$ and let $w_4 = Q \setminus \{q_0\} \cup \{p_1 \mid p \in Q\}$.

We give the sets of rules of Π which simulate the instructions of M and then provide the necessary details of the proof.

We start with simulating the instructions for increment in M.

For any instruction (p, A_i+, q, s) of M, where $A_i \in \mathcal{R}$, the set of rules in P consists of the following rules.

$p.1.1 : (1, p)(4, p_1) \to (3, p)(2, p_1)$

$p.2.1 : (2, p_1)(0, A_i) \to (3, p_1)(1, A_i)$

$p.3.1 : (3, p_1)(4, q) \to (1, p_1)(2, q)$ $\qquad$ $p.3.2 : (3, p_1)(4, s) \to (1, p_1)(2, s)$

$p.4.1 : (1, p_1)(3, p) \to (0, p_1)(4, p)$

$p.5.1 : (0, p_1)(2, q) \to (4, p_1)(1, q)$ $\qquad$ $p.5.2 : (0, p_1)(2, s) \to (4, p_1)(1, s)$

At the beginning, cell 1 contains p and as many occurrences of symbols A_j, $1 \leq j \leq k$, as represent the number stored in the jth register of M. All other elements of Q are in cell 4, and auxiliary, assistant symbols p_1 and p_2, for every $p \in Q$ are in cell 4 and cell 2, respectively. At the end of the simulation of the instruction, one symbol A_i moves from the environment to cell 1, p to cell 4, and either $q \in Q$ or $s \in Q$ moves to cell 1. Furthermore, the other symbols are in their original location. The procedure is governed by the symbol p_1, where its route guarantees the execution of the rules in the correct order. At the beginning of the simulation, the GCPS is in configuration $(0, [A^+]), (1, pu)(2, w_1), (3, \emptyset), (4, w_2)$, where u is a multiset of elements of $\{A_1, \ldots, A_k\}$, $w_2 = \{p_2 \mid (p, A-, q, s) \in P\}$, $w_4 = Q \setminus \{q_0\} \cup \{p_1 \mid p \in Q\}$, and $[A^+]$ denotes that there are arbitrarily many copies of every element of $\{A_1, \ldots, A_k\}$ in the environment.

Next, we describe the transitions of Π.

We start with the simulation of the increment instruction. First, by applying rule $p.1.1$, p_1 moves to cell 2 and in the meantime p leaves cell 1 and enters cell 3. After that, by rule $p.2.1$, a symbol A_i moves to cell 1 and p_1 leaves cell 2 and enters cell 3. This implies that rule $p.2.1.$ cannot be repeated, thus only one copy of A_i enters the system. In the following computation steps, either q or s in Q should move to cell 1, and p_1 should return to cell 4. Let us consider the case of q, the case of s is analogous. Rules $p.3.1$, $p.4.1$, and $p.5.1$, performed after each other, lead to the required configuration, namely, cell 1 contains q, the number of A_is increased by one, p is in cell 4, and cell 2 contains w_1 and cell 3 is empty. Notice that the execution of the rule sequence $p.3.1$, $p.4.1$, $p.5.1$ cannot interfere the execution of the rule sequence $p.3.2$, $p.4.1$, and $p.5.2$, which belongs

to $s \in Q$. It can be seen that the above sequence of computation steps simulates the execution of instruction (p, A_i+, q, s) of M and only that, no interference with the simulation of other instructions is possible.

We refer to the appendix (picture `pshift_plus`) for the graphical representation of the simulation of the increment instruction.

In the following we consider the simulation of the decrement instructions of M.

For any instruction (p, A_i-, q, s) of M, where $A_i \in \mathcal{R}$ the set of rules in P consists of the following rules.

$$p.1.1 : (1, p)(4, p_1) \to (3, p)(2, p_1)$$

$$p.2.1 : (3, p)(2, p_2) \to (4, p)(1, p_2) \qquad p.2.2 : (2, p1)(1, A_i) \to (3, p1)(0, A_i)$$

$$p.3.1 : (1, p_2)(3, p_1) \to (0, p_2)(4, p_1) \qquad p.3.2 : (1, p_2)(2, p_1) \to (3, p_2)(4, p_1)$$

$$p.4.1 : (0, p_2)(4, q) \to (2, p_2)(1, q) \qquad p.4.2 : (3, p_2)(4, s) \to (2, p_2)(1, s)$$

As in the case of the increment instruction, the simulation starts with the following configuration of Π: $(0, [A^+]), (1, pu)(2, w_1), (3, \emptyset), (4, w_2)$, where u is a multiset of elements of $\{A_1, \ldots, A_k\}$, $w_2 = \{p_2 \mid (p, A_i-, q, s) \in P\}$,$w_4 = Q \setminus \{q_0\} \cup \{p_1 \mid p \in Q\}$, $[A^+]$ denotes that there are arbitrarily many copies of the elements of $\{A_1, \ldots, A_k\}$ in the environment. Cell 1 contains as many occurrences of A_j, $1 \leq j \leq k$, as represent the number stored in the jth register of M. All other elements of Q are in cell 4, and auxiliary, assistant symbols p_1 and p_2, for every $p \in Q$ are in cell 4 and cell 2, respectively. At the end of the simulation of the instruction, if cell 1 contains at least one copy of A_i, then the number of symbols A_i is decremented by 1 (one A_i moves to the environment from cell 1), p and q swap their locations. If cell 1 has no copy of A_i, then p and s swap their cells and the other symbols are in their original location.

The simulation starts with the application of rule $p.1.1$. After that p moves to cell 3 and p_1 moves to cell 2. Next, by applying rule $p.2.1$, p continues its way to cell 4, its target cell, and p_2 moves from cell 2 to cell 1. The following step can be either the application of rule $p.2.2$ or the rule $p.3.2$. In the first case, a copy of A_i leaves cell 1 and moves to the environment and p_1 moves to cell 3. In the other case, no symbol A_i is in cell 1, thus $p.2.2$ cannot be applied, furthermore $p.3.1$ cannot be applied, too. This rule application is followed by rule $p.3.1$, which moves p_1 to cell 4 and p_2 to the environment. In the absence of A_i in cell 1, the rule $p.3.2$ is applied which moves p_1 to cell 4 (its original cell) and p_2 to cell 3. The procedure is closed by the application of the rule $p.4.1$ (if cell 1 had a copy of A_i) or the rule $p.4.2$ (if no A_i was present in cell 1. In the first case q and in the second case s moves to cell 1, and in both cases p_2 returns to cell 4. It can be seen that the above procedure simulates the decrement instruction (p, A_i-, q, s) of M and only that, no interference with the simulation of other instructions is possible.

We refer to the Appendix (pictures `pshift_zm_z` and `pshift_zm_nz`) for the graphical representation of the simulation of the decrement instruction (for the two cases — when the register is zero and non-zero).

Π simulates the halting of the register machine as follows. Π is defined in such way that for $q_f \in Q$, where q_f corresponds to the final state of M, there is no increment and no decrement instruction of M that can be simulated by Π starting with q_f in the cell 1.

Hence we proved that any halting transition sequence of Π corresponds to a halting transition sequence of M. Obviously, the numbers generated by these two transition sequences are the same. This implies that the statement of the theorem holds.

Since every set of numbers generated by a GCPS is a recursively enumerable set of non-negative integers, therefore the reverse inclusion holds. This implies the following corollary.

Corollary 1. $NOtP_4(pshift) = NRE.$

Now, we consider the case of the presence-move (pm) rule.

Theorem 2. $NOtP_3(pm) \supseteq NRE.$

Proof. We show that any recursively enumerable set of non-negative integers can be generated by a GCPS with three cells and with only presence-move rules.

Let $M = (Q, \mathcal{R}, q_0, q_f, P)$ be a counter automaton, with $\mathcal{R} = \{A_1, \ldots, A_k\}$, $k \geq 1$, defined as in Section 2. We give a GCPS $\Pi = (O, E, w_1, w_2, w_3, R, 1)$ with only presence-move rules such that any halting transition sequence of Π corresponds to a halting transition sequence of M. Furthermore, these two transition sequences generate the same numbers.

Let Π have the following components. (Since it is clear from the context, we use the symbol A_j also in case of Π). Let

$O = Q \cup \{A_j \mid 1 \leq j \leq k\} \cup \{p_1, p_2 \mid p \in Q\} \cup \{p_3 \mid (p, A_j+, q, s) \in P\} \cup \{L_1, L_2, Z\}$,

$E = \{A_j \mid 1 \leq j \leq k\} \cup \{L_1\}$.

We note that number n, $n \geq 0$ stored in register A_j in M, $1 \leq j \leq k$, is represented by A_j^n in Π.

We give the initial configuration. Let $w_1 = q_0$, $w_2 = \{Z, L_1\}$ and $w_3 = Q \setminus \{q_0\} \cup \{p_1, p_2, p_3 \mid p \in Q\} \cup \{Z, L_2\}$.

We give the sets of rules of Π which simulate the instructions of M and only that and provide the necessary details of the proof.

We start with simulating the instructions for increment in M.

For any instruction (p, A_i+, q, s) of M, the set of rules in P consists of the following rules.

$p.1.1 : (1,p)(3,p_1) \to (1,p)(2,p_1)$

$p.2.1 : (1,p)(3,p_2) \to (1,p)(0,p_2)$ $\quad p.2.2 : (2,p_1)(0,A_i) \to (2,p_1)(3,A_i)$

$p.3.1 : (0,p_2)(3,p_3) \to (0,p_2)(2,p_3)$ $\quad p.3.2 : (3,A_i)(2,p_1) \to (3,A_i)(0,p_1)$

$p.4.1 : (0,p_2)(1,p) \to (0,p_2)(3,p)$ $\quad p.4.2 : (2,p_3)(3,A_i) \to (2,p_3)(1,A_i)$

$p.4.3 : (0,p_1)(3,q) \to (0,p_1)(2,q)$ $\quad p.4.4 : (0,p_1)(3,s) \to (0,p_1)(2,s)$

$p.5.1 : (2,p_3)(0,p_1) \to (2,p_3)(3,p_1)$ $\quad p.5.2 : (0,p_2)(2,q) \to (0,p_2)(1,q)$

$p.5.3 : (0,p_2)(2,s) \to (0,p_2)(1,s)$

$p.6.1 : (3,p_1)(2,p_3) \to (3,p_1)(0,p_3)$ $\quad p.6.2 : (1,q)(0,p_2) \to (1,q)(3,p_2)$

$p.6.3 : (1,s)(0,p_2) \to (1,s)(3,p_2)$

$p.7.1 : (1,q)(0,p_3) \to (1,q)(3,p_3)$ $\quad p.7.2 : (1,s)(0,p_3) \to (1,s)(3,p_3)$

$p.L.1 : (0,p_1)(2,L_1) \to (0,p_1)(1,L_1)$ $\quad p.L.2 : (0,p_2)(2,L_1) \to (0,p_2)(1,L_1)$

$p.L.3 : (0,p_3)(2,L_1) \to (0,p_3)(1,L_1)$ $\quad p.L.4 : (3,A)(2,L_1) \to (3,A)(1,L_1)$

$p.L.5 : (2,p_1)(0,L_1) \to (2,p_1)(1,L_1)$ $\quad p.L.6 : (2,p_3)(0,L_1) \to (2,p_3)(1,L_1)$

$L_1.1 : (2,Z)(1,L_1) \to (2,Z)(3,L_1)$ $\quad L_1.2 : (2,Z)(3,L_1) \to (2,Z)(1,L_1)$

The simulation starts in a configuration where cell 1 contains p and a multiset of symbols A_j, $1 \leq j \leq k$, which represents the numbers stored in the corresponding counter of M. In cell 2 there are auxiliary symbols $\{Z, L_1\}$ and and in cell 3 there are symbols of $Q \setminus \{q_0\} \cup \{p_1, p_2, p_3 \mid p \in Q\} \cup \{Z, L_2\}$, where each symbol is in one copy. At the end of the simulation of the instruction, cell 1 has one more copy of A_i and a copy of $q \in Q$ or $s \in Q$, symbol p in cell 3. All the other symbols are in the same location (in the same cell) as they were at the beginning of the computation.

The simulation starts with the rule $p.1.1$, where in the presence of p in cell 1, p_1 moves to cell 2. After that, by applying rules $p.2.1$ and $p.2.2$ in parallel, we obtain a configuration, where p is still in cell 1, p_2 is in the environment, p_1 is in cell 2 and a copy of A_i entered cell 3 from the environment. Our aim is to move A_i to cell 1, p, p_1, p_2 to cell 3, and q (or s) to cell 1. In the next step, applying in parallel rules $p.3.1$ and $p.3.2$, we make preparations. After then, p_3 moves to cell 2 and p_1 moves to the environment. In the meantime, p_2 is present in the environment and A_i is present in cell 3. Recall that each symbol can only be used in one rule at a time. In the next step, by parallel application of the rules $p.4.1$, $p.4.2$, and $p.4.3$, in the presence of p_2 in the environment, p_3 in cell 2, p_1 in the environment, p moves to cell 3, A_i moves to cell 1, and q moves to cell 2, respectively. By the rule $p.4.4$, s can be taken instead of q. The rest of the work is to move q or s to cell 1 and return the other symbols to their original location. Applying the rules $p.5.1$ and $p.5.2$ (or $p.5.3$) in parallel, p_1 returns to cell 3 and q or s, depending on the previous steps, moves to cell 1. These moves take place in the presence of p_3 in cell 2 and p_2 in the environment. After that, by parallel application of rules $p.6.1$ and $p.6.2$ or $p.6.3$ instead of $p.6.2$, p_2 returns to cell 3, and finally, p_3 moves to cell 3 by the rule $p.7.1$ or $p.7.2$. It can easily be seen that

the above procedure simulates the increment instruction of M, we obtain the required configuration at the end of the transition sequence. It can also be seen that in the first few steps, rules other than the above rules can also be applied. To avoid that the transition sequence in Π does not correspond to the transition sequence in M, we introduced rules that generate infinite transition loops. If these rules are applied, then an infinite loop arises, and thus the simulation of the transition fails. These rules guarantee that given symbols are in a given cell in the right moment or a given symbol stays in a given cell until a right moment. These rules are $p.L.1, \ldots p.L.6$, and $L_1.1$, $L_1.2$. Rules $p.L.1, \ldots p.L.4$ move symbol $L1$ to cell 1. After that, rules generate an infinite loop of transitions with symbols L_1 and Z. It can easily be seen that these rules simulate the increment instruction of M and only that.

In the following we present a simulation of the increment instruction. To help the readability, we indicate only those symbols in the configurations which are necessary to follow the changes. We note that $[A^+]$ denotes that in cell 0, i.e., in the environment, there are arbitrarily many copies of A_i, for every i, $1 \leq i \leq k$. In the case discussed below, symbol p is changed to q, i.e. M enters from state p to state q. The other possibility, namely, where p changes to s is analogous.

$$
\begin{aligned}
&(0, [A^+])(1, p)(2, ZL_1L_2)(3, p_1p_2p_3qs) \Rightarrow_{p.1.1} \\
&(0, [A^+])(1, p)(2, p_1ZL_1L_2)(3, p_2p_3qs) \Rightarrow_{p.2.1,p.2.2} \\
&(0, p_2[A^+])(1, p)(2, p_1ZL_1L_2)(3, A_ip_3qs) \Rightarrow_{p.3.1,p.3.2} \\
&(0, p_1p_2[A^+])(1, p)(2, p_3ZL_1L_2)(3, A_iqs) \Rightarrow_{p.4.1,p.4.2,p.4.3} \\
&(0, p_1p_2[A^+])(1, A_ip)(2, qp_3ZL_1L_2)(3, ps) \Rightarrow_{p.5.1,p.5.2} \\
&(0, p_2[A^+])(1, A_iq)(2, p_3ZL_1L_2)(3, p_1ps) \Rightarrow_{p.6.1,p.6.2} \\
&(0, p_3[A^+])(1, A_iq)(2, ZL_1L_2)(3, p_1p_2ps) \Rightarrow_{p.7.1,p.7.2} \\
&(0, [A^+])(1, A_iq)(2, ZL_1L_2)(3, p_1p_2p_3ps)
\end{aligned}
$$

A detailed description of the transition tree for the increment instruction, as a figure, can be found in the Appendix (figure `pm_plus`), where dashed nodes correspond to configurations where the loop symbol is activated.

Now, we consider the decrement instruction. For any instruction (p, A_i-, q) of M, the set of rules in P consists of the following rules.

$p.1.1 : (1, p)(3, p_1) \to (1, p)(2, p_1)$

$p.2.1 : (1, p)(3, p_2) \to (1, p)(0, p_2)$ $\quad p.2.2 : (2, p_1)(1, A_i) \to (2, p_1)(0, A_i)$

$p.3.1 : (0, p_2)(2, p_1) \to (0, p_2)(1, p_1)$

$p.4.1 : (1, p_1)(0, p_2) \to (1, p_1)(2, p_2)$

$p.5.1 : (2, p_2)(1, p_1) \to (2, p_2)(0, p_1)$

$p.6.1 : (0, p_1)(3, q) \to (0, p_1)(1, q)$ $\quad p.6.2 : (2, p_2)(1, p) \to (2, p_2)(3, p)$

$p.7.1 : (1, q)(0, p_1) \to (1, q)(3, p_1)$ $\quad p.7.2 : (3, p)(2, p_2) \to (3, p)(0, p_2)$

$p.8.1 : (1, q)(0, p_2) \to (1, q)(3, p_2)$

$p.L.1 : (0, p_1)(2, L_1) \to (0, p_1)(1, L_1)$ $\quad p.L.2 : (0, p_2)(2, L_1) \to (0, p_2)(1, L_1)$

$p.L.3 : (2, p_1)(0, L_1) \to (2, p_1)(1, L_1)$ $\quad p.L.4 : (2, p_2)(0, L_1) \to (2, p_2)(1, L_1)$

$p.L.5 : (1, p_1)(3, L_2) \to (1, p_1)(2, L_2)$

$L_2.1 : (3, Z)(2, L_2) \to (3, Z)(0, L_2)$ $\quad L_2.2 : (3, Z)(0, L_2) \to (3, Z)(2, L_2)$

As in the previous case, the simulation starts with a configuration where cell 1 contains p and a multiset of symbols A_j, $1 \leq j \leq k$, which represents the numbers stored in the corresponding counter of M. In cell 2 there are auxiliary symbols $\{Z, L_1\}$ and and in cell 3 there are symbols of $Q \setminus \{q_0\} \cup \{p_1, p_2, p_3 \mid p \in Q\} \cup \{L_2, Z\}$, where each symbol is in one copy. (We note that in the following we indicate only those symbols in the configurations which are relevant for the transitions.) At the end of the simulation of the instruction cell 1 will have one copy of A_i less, if at the beginning and it had at least one A_i in this cell, otherwise the computation fails. If the computation does not fail, then p from cell 1 and q from cell 3 swap their locations. All the other symbols are in the same cell as they were at the beginning of the computation.

The simulation starts with the rule $p.1.1$, where in the presence of p in cell 1, p_1 moves to cell 2. Suppose that cell 1 contains at least one copy of A_i. Then, by applying rules $p.2.1$ and $p.2.2$, we obtain a configuration, where p is still in cell 1, p_2 is in the environment, p_1 is in cell 2 and a copy of A_i enters to the environment from cell 1. If this is not the case, then $p.2.2$ can be applied, thus we obtain a configuration where p_1 is in cell 1 and p_2 is in the environment. In the next step, $p.3.1$ is applied and then in the presence of p_2 in the environment, p_1 moves to cell 1. Next, in presence of p_1 in cell 1, p_2 moves to cell 2 from the environment, and after then, when p_2 is in cell 2, p_1 leaves to the environment. In the next step, two rules are applied in parallel, $p.6.1$ and $p.6.2$, resulting in a configuration when q moves to cell 1 from cell 3 and p moves from cell 3 to cell 1, i.e., they swap their location. In the meantime, p_1 remains (is present) in the environment and p_2 in cell 2. What remains to do is, to move p_1 and P_2 to their original place, to cell 3. This is done by the assistance of p in cell 3 and q in cell 1, using rules $p.7.1$, $p.7.2$, and $p.8.1$. The obtained configuration meets the requirements described above. However, we should point out that if no A_i is in cell 1, then the computation fails. After performing rules $p.1.1$ and $p.2.1.$, in the absence of A_i in cell 1, rule $p.L.2$ can be applied, and the computation will

fail. Analogously, if the application of the rules in the computation are not the ones given above, then some of the rules $p.L.1, \ldots, p.L.6$ and $L_{2.1}, L_{1.2}$ can be/ are applied, which arises an infinite loop, thus the computation fails.

The reader can see that the given rules simulate the decrement instruction of M and only that.

As in the case of the simulation of the increment instruction of M, we present a simulation of the decrement instruction when A_i is present in cell 1. As in the previous case, we indicate only those symbols in the configurations which are necessary to follow the changes. Recall that $[A^+]$ denotes that in cell 0, i.e., in the environment, there are arbitrarily many copies of A_j, for every j, $1 \leq j \leq k$. In the case described below, one copy on A_i leaves cell 1 and symbol p changes to q.

$$
\begin{aligned}
&(0, [A^+])(1, pA_i)(2, ZL_1)(3, p_1p_2p_3qL_2) \Rightarrow_{p.1.1} \\
&(0, [A^+])(1, pA_i)(2, p_1ZL_1)(3, p_2p_3qL_2) \Rightarrow_{p.2.1, p.2.2} \\
&(0, p_2[A^+]p_2A_i)(1, p)(2, p_1ZL_1)(3, p_3qL_2) \Rightarrow_{p.3.1} \\
&(0, p_2[A^+]A_i)(1, pp_1)(2, p_3ZL_1)(3, p_3qL_2) \Rightarrow_{p.4.1} \\
&(0, [A^+]A_i)(1, pp_1)(2, p_2p_3ZL_1)(3, p_3qL_2) \Rightarrow_{p.5.1} \\
&(0, [A^+]A_ip_1)(1, p)(2, p_2p_3ZL_1)(3, p_3qL_2) \Rightarrow_{p.6.1, p.6.2} \\
&(0, [A^+]A_ip_1)(1, q)(2, p_2p_3ZL_1)(3, pp_3L_2) \Rightarrow_{p.7.1, p.7.2} \\
&(0, [A^+]A_iP_2)(1, q)(2, p_3ZL_1)(3, pp_1p_3L_2) \Rightarrow_{p.8.1} \\
&(0, [A^+]A_i)(1, q)(2, p_3ZL_1)(3, pp_1p_2p_3L_2)
\end{aligned}
$$

A detailed description of the transition tree corresponding to the decrement instruction, as figures, can be found in the Appendix (figures `pm_minus_z` and `pm_minus_nz` corresponding to the cases when the register is zero, so the computation fails, or non-zero), where dashed nodes correspond to configurations where the loop symbol is activated.

Finally, we deal with the zero-check instruction. Recall that a zero-check instruction $(p, A_i0, q) \in P$ is performed if M is in state p, and if the number stored in register A_i is 0, then A_is will store zero in the counter and M enters state q. If the contents of counter A_i is not zero, then the computation blocks and the corresponding branch of the computation fails.

For any instruction (p, A_i0, q) of M, the set of rules in P consists of the following rules.

$p.1.1 : (1, p)(3, p_1) \to (1, p)(2, p_1)$

$p.2.1 : (1, p)(3, p_2) \to (1, p)(0, p_2)$ $\quad p.2.2 : (1, A_i)(2, p_1) \to (1, A_i)(0, p_1)$

$p.3.1 : (0, p_2)(2, p_1) \to (0, p_2)(1, p_1)$

$p.4.1 : (1, p_1)(0, p_2) \to (1, p_1)(2, p_2)$

$p.5.1 : (2, p_2)(1, p) \to (2, p_2)(3, p)$ $\quad p.5.2 : (1, p_1)(3, q) \to (1, p_1)(2, q)$

$p.6.1 : (2, q)(1, p_1) \to (2, q)(3, p_1)$ $\quad p.6.2 : (3, p)(2, p_2) \to (3, p)(0, p_2)$

$p.7.1 : (0, p_2)(2, q) \to (0, p_2)(1, q)$

$p.8.1 : (1, q)(0, p_2) \to (1, q)(3, p_2)$

$p.L.1 : (0, p_1)(2, L_1) \to (0, p_1)(1, L_1)$ $\quad p.L.2 : (0, p_2)(2, L_1) \to (0, p_2)(1, L_1)$

$p.L.3 : (2, p_2)(0, L_1) \to (2, p_2)(1, L_1)$ $\quad p.L.4 : (1, p_1)(3, L_2) \to (1, p_1)(2, L_2)$

The simulation starts in a configuration where cell 1 contains p and a multiset of symbols A_j, $1 \leq j \leq k$, which represents the numbers stored in the ith counter of M. In cell 2 there are auxiliary (loop-generating) symbols $\{Z, L_1, L_2\}$ and in cell 3 there are symbols of $Q \setminus \{q_0\} \cup \{p_1, p_2, p_3 \mid p \in Q\} \cup \{Z\}$, where each symbol is present in one copy. We note that the execution of the instruction in M and thus the simulation of the execution of this instruction is successful if and only if M stores zero in the ith counter and thus there is no occurrence of A_i in cell 1. At the end of the simulation of the instruction, cell 1 has no occurrence of A_i, and either p and q or p and s swap their location, depending on whether or not the execution of he instruction was successful. All the other symbols are in the same location (in the same cell) as they were at the beginning of the computation.

We describe the simulation of a successful execution of the instruction. It starts with the application of rule $p.1.1$, resulting in a new configuration where p_1 moves from cell 3 to cell 2. In the next step there are two options: If no A_i is in cell 1, then rule $p.2.1$ is applied and p_2 from cell 3 moves to the environment. If at least one A_i is present in cell 1, then rules $p.2.1$ and $p.2.2$ are applied in parallel and after that both p_1 and p_2 move to the environment. In this case, in the next step rule $L.1$ is applied that leads to a loop generation. If there was no occurrence of A_i in cell 1, then, the computation continues with rule $p.3.1$, where in the presence of p_2 in the environment, p_1 moves to cell 1, and in the next step, by rule $p.4.1$ symbol p_2 returns from the environment to cell 2. This can only be done if p_1 is in cell 1. Now, there are two tasks to complete this phase of the computation: to swap the locations of p and q and to send back symbols p_1 and p_2 to their original locations. Rules $p.5.1$ and $p.5.2$ applied in parallel, result in a configuration where p_1 is in cell 1, p_2 is in cell 2, p is in cell 3, and q is in cell 2. After that, by rule $p.6.1$, symbol p_1 returns to cell 3. The next three rules, within the next two steps, $p.6.2$, $p.7.1$ and $p.7.2$ move q to cell 1 and rule $p.8.1$ moves p_2 to cell 3. Thus, we obtain the required configuration and the simulation in correct. Let us examine the roles of rules L_1, $L.2$, $L.3$ and

$L.4$. Rule $L.1$ can be applied after $p.2.2$, when an occurrence of A_i is moved to the environment. These rules move L_1 to cell 1 that will lead to an infinite loop. Similarly, if rule $L.2$ is applied whenever p_2 is in the environment, L_1 moves to cell 1 and the simulation fails. Rules $L.3$ and $L.4$ work analogously: in the presence of p_2 in cell 2 and in the presence of p_1 in cell 1, they move L_1 to cell 1 and L_2 to cell 2, which actions imply an infinite loop. The reader may see that the rule sets above were constructed in such a way that they result in a correct derivation if and only if they are used in the above described order.

As in the case of the simulation of the increment and the decrement instructions of M, we present a simulation of the zero-check instruction when no A_i is present in cell 1. As in the previous cases, we indicate only those symbols in the configurations which are necessary to follow the changes. Recall that $[A^+]$ denotes that in cell 0, i.e., in the environment, there are arbitrarily many copies of A_j, for every j, $1 \leq j \leq k$. In the case described below, one copy on A_i leaves cell 1 and symbol p changes to q.

$$
\begin{aligned}
&(0,[A^+])(1,p)(2,ZL_1)(3,p_1p_2p_3qL_2) \Rightarrow_{p.1.1}\\
&(0,[A^+])(1,p)(2,p_1ZL_1)(3,p_2p_3qL_2) \Rightarrow_{p.2.1}\\
&(0,[A^+]p_2)(1,p)(2,p_1ZL_1)(3,p_3qL_2) \Rightarrow_{p.3.1}\\
&(0,[A^+]p_2)(1,pp_1)(2,ZL_1)(3,p_3qL_2) \Rightarrow_{p.4.1}\\
&(0,[A^+])(1,pp_1)(2,p_2ZL_1)(3,p_3qL_2) \Rightarrow_{p.5.1,p.5.2}\\
&(0,[A^+])(1,p_1)(2,qp_2ZL_1)(3,pp_3L_2) \Rightarrow_{p.6.1,p.6.2}\\
&(0,[A^+]p_2)(1,)(2,qZL_1)(3,pp_1p_3L_2) \Rightarrow_{p.7.1}\\
&(0,[A^+]p_2)(1,q)(2,ZL_1)(3,pp_1p_3L_2) \Rightarrow_{p.8.1}\\
&(0,[A^+])(1,q)(2,ZL_1)(3,pp_1p_2p_3L_2)
\end{aligned}
$$

A detailed description of the transition tree for the zero instruction, as figures, can be found in the Appendix (figures `pm_zero_z` and `pm_zero_nz` corresponding to the cases when the register is zero, or non-zero, so the computation fails), where dashed nodes correspond to configurations where the loop symbol is activated.

Notice that those rules which involve only auxiliary symbols Z, L_1,L_2, can be used in simulating any instruction, thus belong to the whole rule set of Π.

The above rule sets are defined so that after a successful simulation of one instruction of the counter M, another instruction can be simulated immediately or Π halts.

Recall that the computation ends in M in the final configuration $(q_f, n, 0, \ldots, 0)$ where q_f is the final state of M. This corresponds to the case when Π has q_f in cell 1 together with as many occurrences of elements of A_1 as the number stored in the first counter of M. The other auxiliary symbols are in the cells they were located at the beginning. This implies that $N(M) = N(\Pi)$, and thus the statement holds.

As in the previous case, we obtain the following corollary.

Corollary 2. $NOtP_3(pm) = NRE$.

4 Conclusion

In this paper we considered GCPSs using either only parallel-shift or only presence-move rules. We proved that GCPSs with only parallel-shift rules and only four cells, and GCPSs with only presence-move rules and only three cells are computationally complete as. With these results, we contribute to the research goal of providing a very small lower bound on the number of cells needed to achieve computational completeness for all families of GCPSs where GCPSs use only one type of interaction rules.

In a previous paper [8], we presented a conjecture based on the following observation. For GCPSs using only one of the nine types of interaction rules, it holds that if the rule applies to k cells ($k = 2$, or $k =3$, or $k = 4$), then the minimum number of cells required for its maximal computational power is k (without the environment as a cell). Such results are obtained for conditional-uniport-in, split, join, chain rules [7] and now for parallel-shift and presence move rules. A similar result holds for symport2 rules [2,1]. We guess that fewer than this number of cells is not sufficient to achieve computational completeness. This interesting problem is waiting for future research.

We also note that the pictures in the Appendix were generated by a specially designed simulator that computed all possible rule applications for each step. The simulator was implemented in Java and the pictures were generated using the Graphviz library. The simulator is available upon request.

References

1. Artiom Alhazov, Maurice Margenstern, Vladimir Rogozhin, Yurii Rogozhin, and Sergey Verlan. Communicative P systems with minimal cooperation. In Giancarlo Mauri, Gheorghe Păun, Mario J. Pérez-Jiménez, Grzegorz Rozenberg, and Arto Salomaa, editors, *International Workshop WMC5, Milano, Italy, 2004, LNCS, Springer, 2005*, volume 3365 of *Lecture Notes in Computer Science*, pages 161–177. Springer, 2005.
2. Artiom Alhazov, Yurii Rogozhin, and Sergey Verlan. Minimal cooperation in symport/antiport tissue P systems. *International Journal of Foundations of Computer Science*, 18(1):163–180, 2007.
3. Francesco Bernardini, Marian Gheorghe, Maurice Margenstern, and Sergey Verlan. Producer/consumer in membrane systems and petri nets. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18-23, 2007, Proceedings*, volume 4497 of *Lecture Notes in Computer Science*, pages 43–52. Springer, 2007.
4. Erzsébet Csuhaj-Varjú, György Vaszil, and Sergey Verlan. On generalized communicating P systems with one symbol. In Marian Gheorghe, Thomas Hinze, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*

- 11th International Conference, CMC 2010, Jena, Germany, August 24-27, 2010. Revised Selected Papers, volume 6501 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2010.
5. Erzsébet Csuhaj-Varjú and Sergey Verlan. On generalized communicating P systems with minimal interaction rules. *Theoretical Computer Science*, 412(1-2):124–135, 2011.
6. Erzsébet Csuhaj-Varjú and Sergey Verlan. Bi-simulation between P colonies and P systems with multi-stable catalysts. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing - 18th International Conference, CMC 2017, Bradford, UK, July 25-28, 2017, Revised Selected Papers*, volume 10725 of *Lecture Notes in Computer Science*, pages 105–117. Springer, 2017.
7. Erzsébet Csuhaj-Varjú and Sergey Verlan. Computationally complete generalized communicating P systems with three cells. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, and Claudio Zandron, editors, *Membrane Computing - 18th International Conference, CMC 2017, Bradford, UK, July 25-28, 2017, Revised Selected Papers*, volume 10725 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2017.
8. Erzsébet Csuhaj-Varjú and Sergey Verlan. Conditional uniport P systems with two cells. In Lucie Ciencialová, editor, *Proceedings of the Twenty-fourth International Conference on Membrane Computing (CMC2023), 28-31 August, 2023, Opava, Czech Republic*, pages 97–126. Silesian Unversity at Opava, 2023.
9. Rudolf Freund, Artiom Alhazov, Yurii Rogozhin, and Sergey Verlan. Communication P systems. In Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *The Oxford Handbook of Membrane Computing*, pages 118–143. Oxford University Press, 2009.
10. Rudolf Freund, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez, and Sergey Verlan. A formalization of membrane systems with dynamically evolving structures. *International Journal of Computer Mathematics*, 90(4):801–815, 2013.
11. Rudolf Freund and Sergey Verlan. A formal framework for static (tissue) P systems. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2007.
12. Shankara Narayanan Krishna, Marian Gheorghe, Florentin Ipate, Erzsébet Csuhaj-Varjú, and Rodica Ceterchi. Further results on generalised communicating P systems. *Theor. Comput. Sci.*, 701:146–160, 2017.
13. Marvin Minsky. *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
14. Andrei Paun and Gheorghe Păun. The power of communication: P systems with symport/antiport. *New Gener. Comput.*, 20(3):295–306, 2002.
15. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England, 2010.
16. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.
17. Sergey Verlan, Francesco Bernardini, Marian Gheorghe, and Maurice Margenstern. Computational completeness of tissue P systems with conditional uniport. In Hendrik Jan Hoogeboom, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers*, volume 4361 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2006.

18. Sergey Verlan, Francesco Bernardini, Marian Gheorghe, and Maurice Margenstern. Generalized communicating P systems. *Theoretical Computer Science*, 404(1-2):170–184, 2008.
19. Sergey Verlan, Rudolf Freund, Artiom Alhazov, Sergiu Ivanov, and Linqiang Pan. A formal framework for spiking neural P systems. *Journal of Membrane Computing*, 2:355–368, 2020.
20. Sergey Verlan and Gexiang Zhang. A tutorial on the formal framework for spiking neural P systems. *Natural Computing*, 22(1):181–194, 2023.

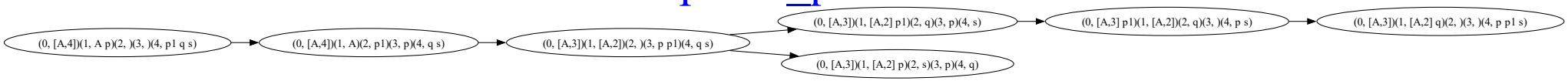
A Appendix

The appendix contains pictures corresponding to configuration graphs where a state corresponds to a configuration and there are transitions to all possible next configurations. These pictures were automatically computed and generated using a specially designed simulator. Since a single instruction is considered, the corresponding graphs are trees. The initial configuration is the root of the tree and it is depicted on the left. There are two types of leaves. The dashed leaves correspond to configurations where the loop symbol is activated. To simplify the pictures we do not make the distinction between L_1 and L_2 loop symbols and use L for both of them. Obviously, these configurations further perform an infinite loop, so the corresponding computation fails. The solid leaves correspond to configurations where the computation is successful.

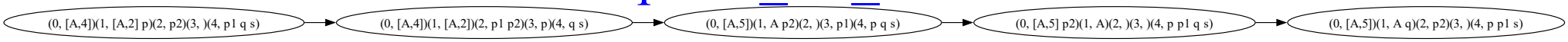
The pictures are named as follows:

- `pshift_plus.pdf` — the configuration graph for the increment instruction of the parallel shift increment instruction construction;
- `pshift_zm_nz.pdf` — the configuration graph for the decrement instruction of the parallel shift decrement instruction construction when the register is non-zero;
- `pshift_zm_z.pdf` — the configuration graph for the decrement instruction of the parallel shift decrement instruction construction when the register is zero;
- `pm_plus.pdf` — the configuration graph for the increment instruction of the presence move increment instruction construction;
- `pm_minus_z.pdf` — the configuration graph for the decrement instruction of the presence move decrement instruction construction when the register is zero;
- `pm_minus_nz.pdf` — the configuration graph for the decrement instruction of the presence move decrement instruction construction when the register is non-zero;
- `pm_zero_z.pdf` — the configuration graph for the zero-check instruction of the presence move zero-check instruction construction when the register is zero;
- `pm_zero_nz.pdf` — the configuration graph for the zero-check instruction of the presence move zero-check instruction construction when the register is non-zero.

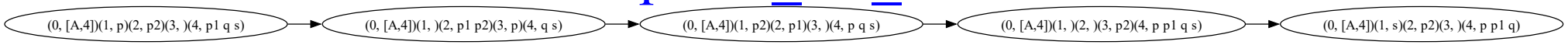
pshift_plus


(0, [A,4])(1, A p)(2,)(3,)(4, p1 q s)
(0, [A,4])(1, A)(2, p1)(3, p)(4, q s)
(0, [A,3])(1, [A,2])(2,)(3, p p1)(4, q s)
(0, [A,3])(1, [A,2] p1)(2, q)(3, p)(4, s)
(0, [A,3])(1, [A,2] p)(2, s)(3, p)(4, q)
(0, [A,3] p1)(1, [A,2])(2, q)(3,)(4, p s)
(0, [A,3])(1, [A,2] q)(2,)(3,)(4, p p1 s)

pshift_zm_nz


(0, [A,4])(1, [A,2] p)(2, p2)(3,)(4, p1 q s)
(0, [A,4])(1, [A,2])(2, p1 p2)(3, p)(4, q s)
(0, [A,5])(1, A p2)(2,)(3, p1)(4, p q s)
(0, [A,5] p2)(1, A)(2,)(3,)(4, p p1 q s)
(0, [A,5])(1, A q)(2, p2)(3,)(4, p p1 s)

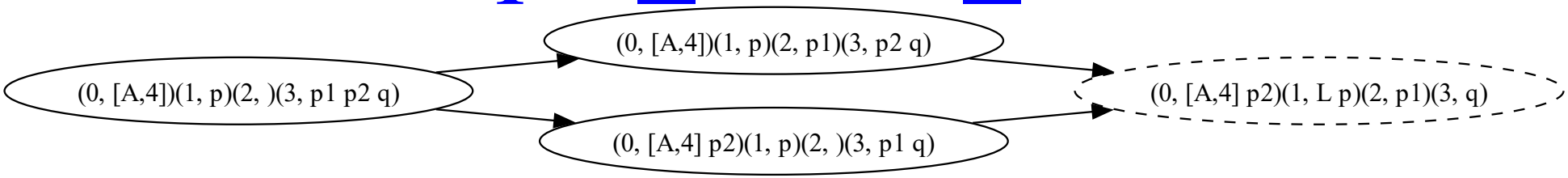
pshift_zm_z


(0, [A,4])(1, p)(2, p2)(3,)(4, p1 q s)
(0, [A,4])(1,)(2, p1 p2)(3, p)(4, q s)
(0, [A,4])(1, p2)(2, p1)(3,)(4, p q s)
(0, [A,4])(1,)(2,)(3, p2)(4, p p1 q s)
(0, [A,4])(1, s)(2, p2)(3,)(4, p p1 q)

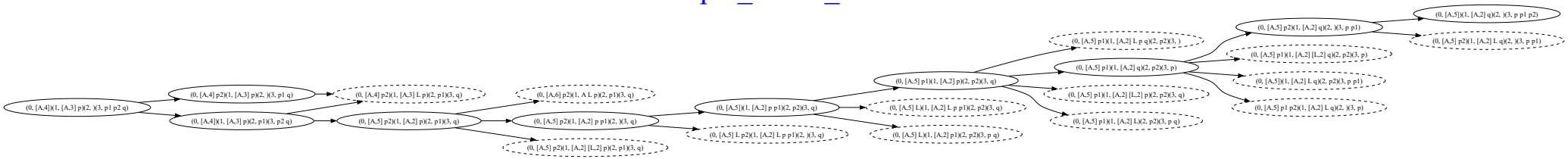
pm_plus


(0, [A,4]p2)(1, A)(2,)(3, p p1 q s)
(0, [A,4] p2)(1, A)(2, p3)(3, p p1 q s)
(0, [A,4] p2)(1, A p)(2,)(3, p1 p3 q s)
(0, [A,4]p2)(1, A p)(2, p1)(3, p2 p3 q s)
(0, [A,4]p2)(1, A p)(2, p1)(3, p2 p3 q s)
(0, [A,4] p2)(1, A p)(2, p1 p3)(3, q s)
(0, [A,4] p2)(1, A L)(2, p1)(3, p3 q s)
(0, [A,4] p2)(1, A L)(2,)(3, p p1 p3 q s)
(0, [A,4] p2)(1, A [L,2])(2, p3)(3, p p1 q s)
(0, [A,4] p2 p3)(1, A L)(2,)(3, p p1 q s)
(0, [A,4] p2)(1, A [L,2])(2, p1 p3)(3, p q s)
(0, [A,4] p2)(1, A [L,3] p)(2, p1 p3)(3, q s)
(0, [A,3] p2)(1, A L)(2, p1 p3)(3, A p q s)
(0, [A,3] p2)(1, A [L,2] p)(2, p1 p3)(3, A q s)
(0, [A,3] p1 p2)(1, A p)(2, p3)(3, A q s)
(0, [A,3] p1 p2)(1, A [L,3])(2, p3)(3, A p q s)
(0, [A,3] p1 p2)(1, A [L,4] p)(2, p3)(3, A q s)
(0, [A,3] p1 p2)(1, A [L,2])(2, p3 s)(3, A p q)
(0, [A,3] p1 p2)(1, A [L,2])(2, p3 q)(3, A p s)
(0, [A,3] p2)(1, A [L,2] p)(2, p3)(3, A p1 q s)
(0, [A,3] p1 p2)(1, A [L,2] L)(2, p3)(3, p q s)
(0, [A,3] p2)(1, A L)(2, p3)(3, A p p1 q s)
(0, [A,3] p1 p2)(1, A [L,2] [L,2] p)(2, p3)(3, q s)
(0, [A,3] p1 p2)(1, A [L,2] L p)(2, p3 q)(3, s)
(0, [A,3] p1 p2)(1, A [L,2] L p)(2, p3 s)(3, q)
(0, [A,3] p1 p2)(1, A [L,3] p)(2, p3 s)(3, A q)
(0, [A,3] p1 p2)(1, A [L,3] p)(2, p3 q)(3, A s)
(0, [A,3] p1 p2)(1, A)(2,)(3, A p p3 q s)
(0, [A,3] p1 p2)(1, A [L,3])(2,)(3, A p p3 q s)
(0, [A,3] p1 p2)(1, A [L,2])(2, p3)(3, A p q s)
(0, [A,3] p1 p2)(1, A L)(2, p3 s)(3, A p q)
(0, [A,3] p1 p2)(1, A [L,2])(2, s)(3, A p p3 q)
(0, [A,3] p1 p2)(1, A [L,2])(2, q)(3, A p p3 s)
(0, [A,3] p1 p2)(1, A L)(2, p3 q)(3, A p s)
(0, [A,3] p2)(1, A p)(2, p1)(3, A p3 q s)
(0, [A,3] p2)(1, A [L,3] p)(2, p1)(3, A p3 q s)
(0, [A,3] p1 p2)(1, A L p)(2,)(3, A p3 q s)
(0, [A,2] p2)(1, A L p)(2, p1 p3)(3, [A,2] q s)
(0, [A,3] p2)(1, A [L,2])(2, p1)(3, A p p3 q s)
(0, [A,2] p2)(1, A [L,2] p)(2, p1)(3, [A,2] p3 q s)
(0, [A,2] p2)(1, A L)(2, p1)(3, [A,2] p p3 q s)
(0, [A,3] p1 p2)(1, A [L,2])(2, p3 s)(3, p q)
(0, [A,3] p1 p2)(1, A [L,2])(2, p3 q)(3, p s)
(0, [A,3] p1 p2)(1, A [L,2] L)(2, p3 s)(3, p q)
(0, [A,3] p2)(1, A [L,2] L)(2, p3 s)(3, p p1 q)
(0, [A,3] p1 p2)(1, A [L,2] [L,2])(2, p3 s)(3, p)
(0, [A,3] p1 p2)(1, A [L,2] [L,3])(2, p3 s)(3, p s)
(0, [A,3] p1 p2)(1, A [L,2] L q)(2, p3 s)(3, p)
(0, [A,3] p2)(1, A [L,2] L)(2, p3 q)(3, p p1 s)
(0, [A,3] p1 p2)(1, A [L,2] L s)(2, p3 q)(3, p)
(0, [A,3] p2)(1, A [L,2] s)(2, p3)(3, p p1 q)
(0, [A,3] p1 p2)(1, A [L,2] [L,2] s)(2, p3)(3, p p1 q)
(0, [A,3])(1, A [L,2] L s)(2, p3)(3, p p1 p2 q)
(0, [A,3] p3)(1, [A,2] s)(2,)(3, p p1 p2 q)
(0, [A,3] p3)(1, [A,2] L s)(2,)(3, p p1 p2 q)
(0, [A,3])(1, [A,2] s)(2,)(3, p p1 p2 p3 q)
(0, [A,3] p2 p3)(1, [A,2] L s)(2,)(3, p p1 q)
(0, [A,3] p3)(1, [A,2] q)(2,)(3, p p1 p2 s)
(0, [A,3] p2)(1, [A,2] [L,2] q)(2, p3)(3, p p1 s)
(0, [A,3])(1, [A,2] L q)(2, p3)(3, p p1 p2 s)
(0, [A,3] p2 p3)(1, [A,2] L q)(2,)(3, p p1 s)
(0, [A,3])(1, [A,2] q)(2,)(3, p p1 p2 p3 s)
(0, [A,3] p3)(1, [A,2] L q)(2,)(3, p p1 p2 s)

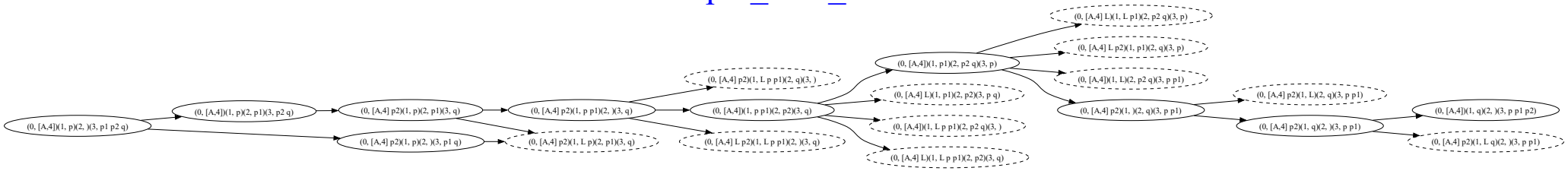
pm_minus_z



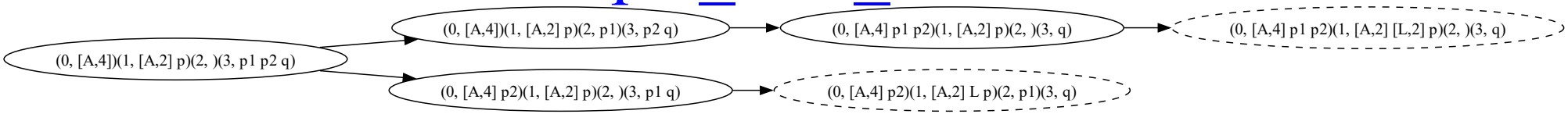
pm_minus_nz


(0, [A.4](1, [A.3] p)(2,)(3, p1 p2 q)
(0, [A.4] p2)(1, [A.3] p)(2,)(3, p1 q)
(0, [A.4] p2)(1, [A.3] L p)(2, p1)(3, q)
(0, [A.4](1, [A.3] p)(2, p1)(3, p2 q)
(0, [A.5] p2)(1, [A.2] p)(2, p1)(3, q)
(0, [A.6] p2)(1, A L p)(2, p1)(3, q)
(0, [A.5] p2)(1, [A.2] p p1)(2,)(3, q)
(0, [A.5] p2)(1, [A.2] [L.2] p)(2, p1)(3, q)
(0, [A.5](1, [A.2] p p1)(2, p2)(3, q)
(0, [A.5] L p2)(1, [A.2] L p p1)(2,)(3, q)
(0, [A.5] p1)(1, [A.2] p)(2, p2)(3, q)
(0, [A.5] L)(1, [A.2] L p p1)(2, p2)(3, q)
(0, [A.5] L)(1, [A.2] p1)(2, p2)(3, p q)
(0, [A.5] p1)(1, [A.2] L p q)(2, p2)(3,)
(0, [A.5] p1)(1, [A.2] q)(2, p2)(3, p)
(0, [A.5] p1)(1, [A.2] [L.2] p)(2, p2)(3, q)
(0, [A.5] p1)(1, [A.2] L)(2, p2)(3, p q)
(0, [A.5] p2)(1, [A.2] q)(2,)(3, p p1)
(0, [A.5] p1)(1, [A.2] [L.2] q)(2, p2)(3, p)
(0, [A.5](1, [A.2] L q)(2, p2)(3, p p1)
(0, [A.5] p1 p2)(1, [A.2] L q)(2,)(3, p)
(0, [A.5](1, [A.2] q)(2,)(3, p p1 p2)
(0, [A.5] p2)(1, [A.2] L q)(2,)(3, p p1)

pm_zero_z


(0, [A,4])(1, p)(2,)(3, p1 p2 q)
(0, [A,4])(1, p)(2, p1)(3, p2 q)
(0, [A,4] p2)(1, p)(2, p1)(3, q)
(0, [A,4] p2)(1, p)(2,)(3, p1 q)
(0, [A,4] p2)(1, p p1)(2,)(3, q)
(0, [A,4] p2)(1, L p p)(2, p1)(3, q)
(0, [A,4] p2)(1, p p1)(2, p2)(3, q)
(0, [A,4] p2)(1, L p p1)(2,)(3, q)
(0, [A,4])(1, p1)(2, p2 q)(3, p)
(0, [A,4] L)(1, p1)(2, p2)(3, p q)
(0, [A,4])(1, L p p1)(2, p2 q)(3,)
(0, [A,4] L)(1, L p p1)(2, p2)(3, q)
(0, [A,4] L)(1, L p1)(2, p2 q)(3, p)
(0, [A,4] L p2)(1, p1)(2, q)(3, p)
(0, [A,4])(1, L)(2, p2 q)(3, p p1)
(0, [A,4] p2)(1,)(2, q)(3, p p1)
(0, [A,4] p2)(1, L)(2, q)(3, p p1)
(0, [A,4] p2)(1, q)(2,)(3, p p1)
(0, [A,4])(1, q)(2,)(3, p p1 p2)
(0, [A,4] p2)(1, L q)(2,)(3, p p1)

pm_zero_nz


(0, [A,4])(1, [A,2] p)(2,)(3, p1 p2 q)
(0, [A,4])(1, [A,2] p)(2, p1)(3, p2 q)
(0, [A,4] p2)(1, [A,2] p)(2,)(3, p1 q)
(0, [A,4] p1 p2)(1, [A,2] p)(2,)(3, q)
(0, [A,4] p2)(1, [A,2] L p)(2, p1)(3, q)
(0, [A,4] p1 p2)(1, [A,2] [L,2] p)(2,)(3, q)