On Generalized Communicating P Systems with One Symbol*

Erzsébet Csuhaj-Varjú[1]**, György Vaszil[1], and Sergey Verlan[2]

[1] Computer and Automation Research Institute, Hungarian Academy of Sciences
Kende u. 13-17, H-1111 Budapest, Hungary
{csuhaj,vaszil}@sztaki.hu

[2] Laboratoire d'Algorithmique, Complexité et Logique,
Département Informatique,
Université Paris Est,
61, av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Abstract. Generalized communicating P systems (GCPSs) are purely communicating tissue-like membrane systems with rules for moving pairs of objects. Despite their simplicity, these systems are rather powerful, they are able to generate any recursively enumerable set of numbers even having restricted variants of communication rules. In this paper we show that GCPSs still remain computationally complete if they are given with a singleton alphabet of objects and with one of the restricted types of rules parallel-shift, join, presence-move, and chain.

1 Introduction

The notion of a *generalized communicating P system* was introduced in [14], with the aim of providing a common generalization of various purely communicating models in the framework of P systems. The model was inspired by and also captures important features of several other well-known distributed computational models.

A generalized communicating P system, or a *GCPS* for short, corresponds to a hypergraph where each node is represented by a cell and each edge is represented by a rule. Every cell contains a multiset of objects which – by communication rules – may move between the cells. The form of a *communication rule* is $(a, i)(b, j) \to (a, k)(b, l)$ where a and b are objects and natural numbers i, j, k, l are labels identifying 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 to

* Research supported in part by the Hungarian Scientific Research Fund, "OTKA", project K75952 and by Science and Technology Center in Ukraine, project 4032.

** Also affiliated with the Department of Algorithms and Their Applications, Faculty of Informatics, Eötvös Loránd University, Pázmány Péter sétány 1/c, 1117 Budapest, Hungary

cell k and cell l, respectively. Communication rules can also be interpreted as *interaction rules*.

Depending on their form, several *restrictions on communication rules* (modulo symmetry) can be introduced; we provide a detailed description of these variants in Subsection 2.2.

When a GCPS has only one type of these restricted rules, then we speak of a *generalized communicating P system with minimal interaction* or a *minimal interaction P system* (with the given type of rules), called also a GCPSMI, for short.

Due to the simplicity of their rules, the generative power of minimal interaction P systems is of particular interest and it has been studied in details. In [14, 4] it was proved that eight of the possible nine restricted variants (with respect to the form of rules) are able to generate any recursively enumerable set of numbers; in the ninth case only finite sets of singletons can be obtained. Furthermore, it was also shown that these systems even with relatively small numbers of cells and simple underlying (hypergraph) architectures are able to achieve this generative power.

In this paper, we add one more restriction, i.e., we study minimal interaction P systems where the *alphabet of objects is a singleton*. We show that these so-called *one-symbol minimal interaction P systems* given with any of the *parallel-shift, presence-move, chain*, and *join rules* are able to generate every recursively enumerable set of natural numbers. These results demonstrate that computational completeness can be achieved with (certain variants of) GCPSMIs defined over the simplest alphabets. We also examine the generative power of one-symbol minimal interaction P systems with *conditional-uniport-in rules* and prove that these constructs are less powerful than the previous one. We note that, by definition, the concept of the rest of the restricted communication rules is not applicable for GCPSs over a one-symbol object alphabet. Finally, we provide topics for future research.

2 Preliminaries

In this section we recall some basic notions and notations used in membrane computing, formal language theory and computability theory. For further details and information the reader is referred to [9–11].

2.1 Some basic notions

An alphabet is a finite non-empty set of symbols. For an alphabet V, we denote by V^* the set of all strings over V, including the empty string, λ.

A finite multiset over V is a mapping $M : V \longrightarrow \mathbb{N}$; $M(a)$ is said to be the multiplicity of a in M ($\mathbb{N}$ denotes the set of non-negative integers). A finite multiset M over an alphabet V can be represented by all permutations of a string $x = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \in V^*$, where $a_j \in V$, $M(a_j) \neq 0$, $1 \leq j \leq n$; x represents M in V^*. If no confusion arises, we also may use the customary set

notation for denoting multisets. The size of a finite multiset M, represented by $x \in V^*$ is defined as $\Sigma_{a \in V} |x|_a$.

Next we recall the notion of a *register machine*; for further details the reader is referred to [7].

A *register machine* is a 5-tuple $M = (Q, R, q_0, q_f, P)$, where Q is a finite non-empty set, called the set of *states*, $R = \{A_1, \ldots, A_k\}$, $k \geq 1$, is a set of *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 $p, q, s \in Q, p \neq q_f, A \in R$, called an *increment instruction*, or $(p, A-, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called a *decrement instruction*. Furthermore, for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q, s)$.

A *configuration* of a register machine M, defined above, is given by 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 corresponds to the current state of M and $m_1, \ldots, m_k$ are the current numbers stored in the registers (in other words, the current contents of the registers or the value of the registers) $A_1, \ldots A_k$, respectively.

A transition of the register machine consists in updating the number stored in a register and in changing the current state to another one, according to 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 $(p, A-, q, s) \in P$ 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, and if the number stored in A is 0, then the contents of A remains unchanged and M enters state s.

We say that a register machine $M = (Q, R, q_0, q_f, P)$, with k registers, given as above, *generates* a non-negative integer n if starting from the *initial configuration* $(q_0, \underbrace{0, 0, \ldots, 0}_{k})$ it enters the *final configuration* $(q_f, \underbrace{n, 0, \ldots, 0}_{k})$.

The set of non-negative integers generated by M is denoted by $N(M)$.

It is known that register machines generate all recursively enumerable sets of non-negative integers [7]; the family of these sets of numbers is denoted by NRE and the family of finite sets of non-negative integers by $NFIN$.

2.2 Generalized communicating P systems

Next we recall the basic definitions concerning generalized communicating P systems [14].

Definition 1. *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 $(a,i)(b,j) \rightarrow (a,k)(b,l)$, 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., $a \notin E$ and/or $b \notin E$;*
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). If no confusion arises, we may distinguish the cells by using another labeling.

We distinguish an additional special cell, labeled by 0, called the *environment*. The environment contains objects of E in an *infinite number of copies*. Since the initial configuration is given and the rules only move but not increase the objects among the cells, we need a supply of objects in the environment.

The cells interact with each other by means of the rules $(a,i)(b,j) \rightarrow (a,k)(b,l)$, with $a, b \in O$ and $0 \leq i,j,k,l \leq n$. Such an interaction rule may be applied if there is an object a in cell i and an object b in cell j. As the result of the application of the rule, the object a moves from cell i to cell k and b moves from cell j to cell l. If two objects from the environment are moved to some other cell or cells, then at least one of them must not appear in the environment in an infinite number of copies. Otherwise, an infinite number of objects can be imported in the system in one step.

Let $\Pi = (O, E, w_1, \ldots, w_n, R, h)$, $n \geq 1$, be a GCPS. A *configuration* of Π 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 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 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. Then, for a configuration $u = (z_0, z_1, \ldots, z_n)$ of Π, a new configuration $u' = (z'_0, z'_1, \ldots, z'_n)$ is obtained by applying the rules of R in a non-deterministic maximally parallel manner: taking an applicable multiset of rules $\mathcal{R}$ over R such that the application of $\mathcal{R}$ results in configuration $u' = (z'_0, z'_1, \ldots, z'_n)$ and there is no other applicable multiset of rules $\mathcal{R}'$ over $\mathcal{R}$ which properly contains $\mathcal{R}$.

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 configurations*, i.e., with a configuration where no further transition step can be performed.

We say that *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 $N(\Pi)$.

In the following we recall the notions of the possible restrictions on the interaction rules (modulo symmetry). Let O be an alphabet and let us consider an interaction rule $(a,i)(b,j) \to (a,k)(b,l)$ with $a,b \in O$, $i,j,k,l \geq 0$. Then we distinguish the following cases:

1. $i = j = k \neq l$: the *conditional-uniport-out rule*
   (the *uout* rule, for short) sends b to cell l provided that a and b are in cell i;

2. $i = k = l \neq j$: the *conditional-uniport-in rule*
   (the *uin* rule, for short) brings b to cell i provided that a is in that cell;

3. $i = j$, $k = l$, $i \neq k$: the *symport2 rule*
   (the $sym2$ rule, for short) corresponds to the minimal symport rule [10], 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, for short) corresponds to the minimal antiport rule [10], 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, for short) 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*
   (the *split* rule, for short) 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*
   (the *join* rule, for short) brings a and b together to cell i;

8. $i = l$, $i \neq j$, $i \neq k$ and $j \neq k$: the *chain rule*
   (the *chain* rule, for short) 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 was previously ;

9. i, j, k, l are pairwise different numbers: the *parallel-shift rule*
   (the $shift$ rule, for short) moves a and b from two different cells to another two different cells.

A generalized communicating P system may have rules of several types as defined above. When only one of them is considered, then we call the corresponding GCPS a *minimal interaction P system* (with the given type of rules), or a GCPSMI, for short.

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

If the number of objects in the alphabet of objects in the GCPSMI is l, then the previous notations are changed to $NOtP_{k,l}(x)$ and $NOtP_{*,l}(x)$, respectively. We call these GCPSMIs *l-symbol minimal interaction P systems* (with a given type of rules). For simplicity, we use terms 1-symbol GCPSMI and *one-symbol GCPSMI* as equivalent.

Throughout the paper, we also refer to the symbol in the one-symbol P systems as token (denoted by $\bullet$). Furthermore, without any loss of the generality we may assume that one-symbol GCPSMIs are given over alphabet $O = \{\bullet\}$.

Notice that in the case of one-symbol minimal interaction P systems, the concepts of rule types conditional-uniport-out, symport2, antiport1, and split are not applicable. Since $O = E$, in these cases the rules $(\bullet, i)(\bullet, j) \to (\bullet, k)(\bullet, l)$ do not satisfy condition 4. of Definition 1, namely, that if $i = 0$ and $j = 0$, then $\{\bullet\} \cap (O \setminus E) \neq \emptyset$.

Due to their simplicity, the generative power of minimal interaction P systems is of particular interest. In [14] and [4] it was shown that $NOtP_*(anti1) \subset NFIN$ and

$$NRE = NOtP_{30}(uin) = NOtP_{30}(uot) = NOtP_{10}(sym2) = NOtP_9(split) = \\ = NOtP_7(join) = NOtP_{36}(presence) = NOtP_{19}(shift) = NOtP_*(chain).$$

These results are valid if no restriction is introduced on the size of the object alphabet of the minimal interaction P systems. In the following we restrict the size of the alphabet of objects to one, and will investigate the generative power of one-symbol minimal interaction P systems.

We observe that such systems are very similar to Petri Nets having a restricted topology. This is especially visible if a graphical notation is used. However, the maximal parallelism and the concept of the environment are more specific to P systems, so we place this study in the latter framework. We also remark that due to the similarity, some of the used terms are borrowed from the area of Petri Nets. A converse study of P systems from the point of view of Petri Nets can be found in [5]. For more details on Petri Nets and membrane computing we refer to [10].

3 Main results

In this section we show that one-symbol minimal interaction P systems with any of rule types parallel-shift, presence-move, chain, and join are able to generate every recursively enumerable set of numbers. If the one-symbol minimal interaction P system Π uses conditional-uniport-in rules, then it either generates a finite set of non-negative numbers or there exists a natural number K such that $N(\Pi)$ contains any natural number $l \geq K$.

We first start with one-symbol minimal interaction P systems with parallel-shift rules and show their computational completeness. A very similar result is given in Theorem 4.4 from [5], however the definition used there is slightly different, but that construction can be adapted to the case of GCPSMI with parallel shift rules. We give below a different proof and use it in the rest of the sequel.

Theorem 1. $NOtP_{*,1}(shift) = NRE$.

Proof. Let $S \in NRE$ generated by a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \dots A_n\}$, $n \geq 1$ (M is given as in Subsection 2.1). To prove the statement, we construct a one-symbol minimal interaction P system $\Pi = (O, E, w_1, \dots, w_r, R_1, h)$, $r \geq 1$, with parallel-shift rules such that $N(M) = N(\Pi)$ holds. The proof idea is based on simulation of the application of the instructions of M by applications of rule sets of Π.

The components of Π are defined as follows. Let $O = E = \{\bullet\}$, and let Π have for any $p \in Q$ cells labeled by elements of $\{p, 1_p, 2_p, 3_p, 4_p, 5_p\}$ in Π such that the sets of cells representing different states of M are pairwise disjoint. (Note that for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q, s)$.) Furthermore, let Π have for any register A_i, $1 \leq i \leq n$, in M a dedicated cell labeled by A_i; the output cell is the one which corresponds to the output register of M. Let these cells be pairwise different and also different from the cells defined previously to the instructions. Furthermore, let Π have no more cells. For the sake of simplicity, throughout the paper, we use terms "cell p" , "cell A_i" and "cell labeled by p", "cell labeled by A_i" as equivalent, respectively.

We define the rule set, R_1, of Π as follows.

For any increment instruction (p, A_i+, q, s) of M, $1 \leq i \leq n$, R_1 contains the following rules.

$$(1^{(p)}) : (\bullet, p)(\bullet, 0) \to (\bullet, q)(\bullet, A_i) \qquad (2^{(p)}) : (\bullet, p)(\bullet, 0) \to (\bullet, s)(\bullet, A_i)$$

For any decrement instruction (p, A_i-, q, s) of M, $1 \leq i \leq n$, R_1 contains the following rules.

$$(1^{(p)}) : (\bullet, p)(\bullet, 0) \to (\bullet, 1_p)(\bullet, 2_p) \qquad (2^{(p)}) : (\bullet, 1_p)(\bullet, A_i) \to (\bullet, 3_p)(\bullet, 0)$$
$$(3^{(p)}) : (\bullet, 2_p)(\bullet, 0) \to (\bullet, 4_p)(\bullet, 5_p) \qquad (4^{(p)}) : (\bullet, 1_p)(\bullet, 4_p) \to (\bullet, s)(\bullet, 0)$$
$$(5^{(p)}) : (\bullet, 3_p)(\bullet, 4_p) \to (\bullet, q)(\bullet, 0)$$

Furthermore, R_1 consists only of the rules defined above.

The initial contents of the cells are given as by $w_{q_0} = \bullet$, and $w_x = \lambda$ for $x \in Q \cup \{k_y \mid 1 \leq k \leq 5, y \in Q\} \cup \{A_i \mid 1 \leq i \leq n\}$. It can easily be observed that this configuration corresponds to the initial configuration of M.

In the following we show that the application of an increment instruction or a decrement instruction of M can be simulated by the application of the corresponding rule set of Π defined above.

The reader may easily notice that $(\bullet, p)$ means that the current state of M is p, therefore the fact that rules $(1^{(p)})$ and rule $(2^{(p)})$ describe the application of an increment instruction of M is obvious.

Suppose now that the instruction of M to be simulated is (p, A_i-, q, s). Then cell p contains a token. (We note that no cell p, where $p \in Q$ has more than one

token during any step of the generation). At the first step of the simulation of the instruction, only rule $(1^{(p)})$ is applicable which moves the token from cell p and one token from the environment to cells 1_p and 2_p, respectively. If cell A_i contains at least one token, then, by rule $(2^{(p)})$, a token from cell 1_p moves to cell 3_p and one token from cell A_i exits to the environment. Meantime, the token from cell 2_p and one token from the environment are transported to cells 4_p and 5_p, respectively. At the next step, the token in cell 3_p moves to cell q and the token in cell 4_p exits to the environment, thus, the obtained configuration of Π corresponds to the configuration of M after performing the decrement instruction if register A_i contained at least one symbol. If cell A_i does not contain any token, then after performing rule $(1^{(p)})$, the only applicable rule is $(3^{(p)})$, which sends one-one tokens to cell 4_p and cell 5_p. After then, rule $(4^{(p)})$ is applied, which moves the token from cell 1_p to cell s and sends the token in cell 4_p to the environment. Thus, the simulation of the instruction is correct in this case as well.

Examining the proof above, the reader may observe that the generation process in Π is governed by the token arriving in a cell labeled by a state of M and no simulation of simultaneous instructions of M is possible.

By the construction of R_1, if M enters state q_f, then Π halts, since there is no applicable rule if cell q_f contains a token. It also can easily be seen that the number of tokens at the output cell is equal to the number stored in the output register of M by halting. Thus, $N(M) = N(\Pi$.

Using in part the construction in the proof of Theorem 1, we show that one-symbol minimal interaction P systems with join rules are also computationally complete.

Theorem 2. $NOtP_{*,1}(join) = NRE$.

Proof. The proof, as the previous one, is based on simulation of the work of a register machine. Let $S \in N(RE)$ generated by a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \dots A_n\}$, $n \geq 1$. (M is given as in Subsection 2.1.) We construct a one-symbol minimal interaction P system $\Pi = (O, E, w_1, \dots, w_r, R_1, h)$, $r \geq 1$, with join rules such that $N(M) = N(\Pi)$ holds. The construction of Π is done in several steps.

We first note that by Theorem 1 there exists a one-symbol minimal interaction P system $\Pi' = (\{\bullet\}, \{\bullet\}, w'_1, \dots, w'_r, R'_1, h')$, $r \geq 1$, with parallel-shift rules which generates S. Suppose that Π' is given as the GCPSMI in the proof of Theorem 1. It is easy to see that the application of any parallel-shift rule $(t) : (\bullet, i)(\bullet, j) \to (\bullet, k)(\bullet, l)$, where i, j, k, l are labels of cells and (t) is the label of the rule, can be simulated by the application of a join rule $(t') : (\bullet, i)(\bullet, j) \to (\bullet, c_t)(\bullet, c_t)$ followed by a split rule $(t'') : (\bullet, c_t)(\bullet, c_t) \to (\bullet, k)(\bullet, l)$, where c_t is a new cell introduced to rule $(t) : (\bullet, i)(\bullet, j) \to (\bullet, k)(\bullet, l)$. The new cells are pairwise different and different from the already existing ones. By this observation, we can construct a one-symbol minimal interaction P system Π'' which is equivalent to Π', i.e., $N(\Pi'') = N(\Pi')$ and the rule set of Π'' consists of

the join and split rules constructed to the rules of Π' in the previously described manner.

Then, starting from Π'', we will construct Π. To do this, for any split rule in Π'' we design a set of rules in Π which rule set simulates the application of the split rule and only that. For this reason, we first define a block of cells, a so-called pseudo-split block. The term "pseudo-split" refers to that the rule set realizes a split if some conditions hold.

A pseudo-split block is given as follows (see Figure 1).

The block aims to split two tokens from cell 1 to cells 2 and 3. Cell 1 is supposed to have at least two tokens. One of them is sent to cell 2 and the other one to cell 3. Furthermore, one of the following conditions must hold:

1. In one of the nodes 2 and 3 only one token can leave the cell to outside the block in the following steps of the generation.
2. In one of the nodes 2 and 3 no token can leave the cell to outside the block for at least the next step of the generation.

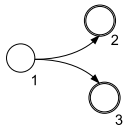

Fig. 1. A pseudo-split block

The pseudo-split block is implemented by join rules as follows (initially cells $4, 4'$ have two tokens and cell $\#$ is a so-called trap cell):

$$(1) : (\bullet, 1)(\bullet, 4) \to (\bullet, 5)(\bullet, 5) \qquad (2) : (\bullet, 1)(\bullet, 4') \to (\bullet, 5')(\bullet, 5')$$
$$(3) : (\bullet, 4)(\bullet, 5) \to (\bullet, 3)(\bullet, 3) \qquad (4) : (\bullet, 4')(\bullet, 5') \to (\bullet, 2)(\bullet, 2)$$
$$(5) : (\bullet, 4)(\bullet, 5') \to (\bullet, \#)(\bullet, \#) \qquad (6) : (\bullet, 4')(\bullet, 5) \to (\bullet, \#)(\bullet, \#)$$
$$(7) : (\bullet, 5)(\bullet, 3) \to (\bullet, 4)(\bullet, 4) \qquad (8) : (\bullet, 5')(\bullet, 2) \to (\bullet, 4')(\bullet, 4')$$

The rules corresponding to the trap cell are the following:

$$(\#_1) : (\bullet, \#)(\bullet, 0) \to (\bullet, \bar{\#})(\bullet, \bar{\#}) \qquad (\#_2) : (\bullet, \bar{\#})(\bullet, 0) \to (\bullet, \#)(\bullet, \#)$$

We explain how the block functions. At starting, there are three possibilities: moving both tokens to cell 5, moving both tokens to cell $5'$ or distributing one token to each of cells 5 and $5'$. In the first two cases, the corresponding cells 4 or $4'$ will contain no token, so the only applicable rule will be rule (5) or (6),

leading to an infinite generation. So the only possibility remains the third case where the tokens found at cell 1 are equally distributed between cells 5 and $5'$ (rules (1) and (2)). After that, two tokens will arrive at cell 2 and two tokens at cell 3 (from cells $4'$ and $5'$ and from cells 4 and 5, respectively). Suppose now that no token can leave cell 3 in the next step (i.e., condition 2 holds). Then, one token from cell 3 and one token from cell 5 move to cell 4. This implies that at the same time one token from cell 2 and one token from cell $5'$ will move to cell $4'$, because if this does not take place, then at the next step a token from cell 4 and the one remained in $5'$ will move to the trap cell. Since the construction is symmetric, a similar generation phase will be performed if it is cell 2 that cannot evolve for one step. The block functions similarly if condition (1) holds.

Notice that the pseudo-split block corresponds to a split rule only if at least one of the conditions (1) and (2) holds. In the following we construct a block arrangement consisting of pseudo-split blocks such that the above criterion is satisfied (see Figure 2).

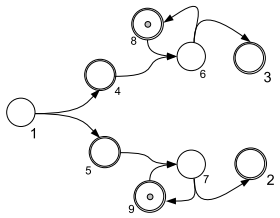

Fig. 2. Pseudo-split blocks simulating a split rule

The block arrangement functions as follows. Initially, each cell 8 and 9 contains one token. Firstly, a pseudo-split block is used to separate two tokens from cell 1 to cells 4 and 5. After then, a join rule is applied to send a token from cell 4 and cell 8 to cell 6 and from cell 5 and cell 9 to cell 7, respectively. This implies that the destination cells of the previously performed pseudo-split, i.e., cells 4 and 5 satisfy condition (1), since only one token can leave these cells. This phase of the generation is followed by two pseudo-splits, where the destination cells satisfy condition (2), i.e., no token will be able to leave cell 8 and 9, respectively, at the next step. This implies that no simulation of another split instruction can start before the tokens coming from cell 1 arrive at cell 2 and 3. Hence, a split instruction is performed.

Combining join and split operations as described above, the rule set of Π can be constructed. We leave the details of the construction to the reader. It can easily be seen that Π generates the same set of numbers as Π', thus $N(\Pi) = N(M)$ holds.

Next we show that one-symbol minimal interaction P systems with presence-move rules are also computationally complete.

Theorem 3. $NOtP_{*,1}(presence) = NRE$.

Proof. Let $S \in NRE$ and let S be generated by a register machine $M = (Q, R, q_0, q_f, P)$ with $R = \{A_1, \dots A_n\}$, $n \geq 1$. (M is given as in Subsection 2.1.) As in the case of the previous statements, we show that a one-symbol minimal interaction P system Π with presence-move rules can be constructed such that $N(M) = N(\Pi)$ holds. GCPSMI Π is defined in several steps.

Instead of direct simulations of the increment and decrement instructions of M, we define sets of rules, called (primitive) blocks, as it was done in [14, 13, 4] and then we show how a set of rules simulating the application of an increment instruction or that of a decrement instruction can be constructed from these blocks.

We will use three types of blocks: the *uniport block*, the *main block*, and the *zero block*.

To help the reader in the easier understanding of the constructions, we add figures.

The *uniport block* is denoted by an arrow between circles labeled by 1 and 2 with a token on the top of it. It corresponds to the move of a token from cell 1 to cell 2. This action is simulated by the following presence-move rule: (we suppose that a token is present initially in cell $1'$): $(\bullet, 1')(\bullet, 1) \to (\bullet, 1')(\bullet, 2)$.

The *main block*, see Figure 3, permits to move synchronously a token from cell i to cell j and a token from cell k to cell m. If no token is present in cell k, then an infinite loop occurs. The arrows show the direction of the move of the objects and the circles corresponds to the cells. Since the semantics of the block is not symmetric, the double circle indicates the place of the symbol that triggers the generation and for which the infinite loop can occur.

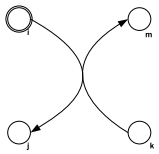

Fig. 3. The main block

The *zero block*, see Figure 4, moves a token from cell i to cell j providing that there is no token in cell k. If this is not the case, then the generation enters an infinite loop. The notations are analogous to the ones used in Figure 3, namely, the arrow denotes the direction of the movement of the object, the circles denote cells, the double line and the circle labeled with k refer to the condition that no token is present in cell k.

In the following we show how the main block and the zero block is implemented in Π. The simulation of the main block is done by the following rule set. We suppose that initially cells 5, 8 and 13 contain a token.

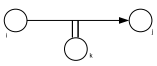

Fig. 4. The zero block

$(1) : (\bullet, 1)(\bullet, 5) \rightarrow (\bullet, 1)(\bullet, 6)$ $(2) : (\bullet, 6)(\bullet, 1) \rightarrow (\bullet, 6)(\bullet, 7)$
$(3) : (\bullet, 7)(\bullet, 3) \rightarrow (\bullet, 7)(\bullet, 4)$ $(4) : (\bullet, 8)(\bullet, 6) \rightarrow (\bullet, 8)(\bullet, 9)$
$(5) : (\bullet, 9)(\bullet, 7) \rightarrow (\bullet, 9)(\bullet, 12)$ $(6) : (\bullet, 12)(\bullet, 9) \rightarrow (\bullet, 12)(\bullet, 5)$
$(7) : (\bullet, 5)(\bullet, 12) \rightarrow (\bullet, 5)(\bullet, 2)$ $(8) : (\bullet, 8)(\bullet, 9) \rightarrow (\bullet, 8)(\bullet, 10)$
$(9) : (\bullet, 8)(\bullet, 10) \rightarrow (\bullet, 8)(\bullet, 11)$ $(10) : (\bullet, 8)(\bullet, 11) \rightarrow (\bullet, 8)(\bullet, 10)$
$(11) : (\bullet, 13)(\bullet, 7) \rightarrow (\bullet, 13)(\bullet, 10)$

These rules are also depicted on Figure 5 where the arrow represents the movement direction and the dashed line the controlling cell.

First, by applying rule (1), a token from cell 1 and the token from cell 5 (notice that cell 1 may contain more than one tokens) moves to cell 6. Then, there are two possibilities. If the rule (4) is used, then a token will arrive in cell 9 and the only possible rule to be applied is rule (8) leading to an infinite generation. The other possibility is using rule (2). At the next step, two possibilities may occur depending on whether cell 3 contains a token or not. Suppose that there exists a token in cell 3. Then rules (3) and (4) are applied in parallel, thus a token in cell 3 moves to cell 4 and the token in cell 6 moves to cell 9. After that, by rule (5), the token in cell 7 is transported to cell 12, and then, by rule (6), the token leaves cell 9 and arrives in cell 5. Then, the token in cell 12 moves to cell 2 (rule (7)), which means that the operations of the main block are performed, i.e., a token from cell 1 moved to cell 2 and a token from cell 3 moved to cell 4. Furthermore, the conditions of the initial configuration hold as well, i.e., each of cells 5, 8, 13 contains one token. Suppose now that cell 3 does not contain a token. Then, after performing rule (2), only rules (4) and (11) can be applied leading to an infinite generation (the token moves infinitely many times between cells 10 and 11). The reader may easily see that the above rules can properly function only in the previously described manner. Notice that if we want to ensure that only one symbol in cell 1 is processed by this block, then we add rule (12):$(\bullet, 14)(\bullet, 1) \rightarrow (\bullet, 14)(\bullet, 10)$ to the above rule set.

The simulation of the zero block can be obtained from the rule set above, by eliminating rule (11) and replacing rule $(3) : (\bullet, 7)(\bullet, 3) \rightarrow (\bullet, 7)(\bullet, 4)$ with $(3) : (\bullet, 3)(\bullet, 7) \rightarrow (\bullet, 3)(\bullet, 10)$. In this case, if rules (3) and (4) can be simultaneously applied, i.e., if there is a token in cell 3, then a token will appear in cell 10 leading to an infinite generation by rules (9) and (10). If rule (3) is not applicable, then applying the sequence of rules (4), (5), (6), (7), a token from cell 1 is successfully moved to cell 2 on the condition that cell 3 does not contain any token. The reader

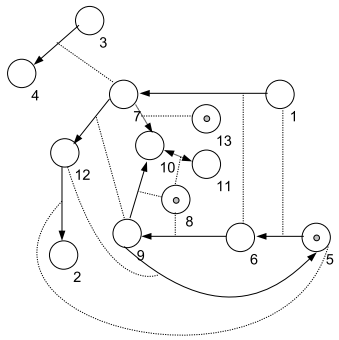

Fig. 5. A set of presence-move rules implementing a main block

may notice that the rules cannot be applied in any other manner as described previously.

Now we explain how the blocks can be used in constructing a one-symbol minimal interaction P system Π generating the same set of numbers as register machine M. To do this, we construct block arrangements of rules of Π which simulate the increment instructions and the decrement instructions of M.

The block arrangement for simulating an increment instruction is illustrated by Figure 6.

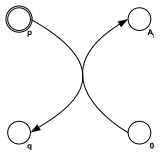

Fig. 6. Block arrangement for simulating an increment instruction

For any increment instruction (p, A_i+, q, s) of M, R_1 contains a rule set, which corresponds to a main block with the following modifications: cell 1 is replaced by cell p, cell 3 by the environment, and cell 4 by cell A_i. Cells i are replaced by cells $i^{(p)}$ for $5 \leq i \leq 12$, respectively. (The cells belonging to the same instruction and to different instructions (states) of M are pairwise different.) Since M may enter from state p either state q or state s, which both represent cell 2 in Π, therefore, instead of rule (7) : $(\bullet, 5)(\bullet, 12) \rightarrow (\bullet, 5)(\bullet, 2)$, we consider rules $(\bullet, 5^{(p)})(\bullet, 12^{(p)}) \rightarrow (\bullet, 5^{(p)})(\bullet, q)$ and $(\bullet, 5^{(p)})(\bullet, 12^{(p)}) \rightarrow (\bullet, 5^{(p)})(\bullet, s)$.

The reader may immediately see that through the main blocks, described above, any increment instruction of M can be simulated, therefore the corresponding part of the rule set of Π can be constructed.

Next we show how to construct a block arrangement for simulating a decrement increment instruction. To help the reader in the easier understanding, we illustrate the construction by Figure 7.

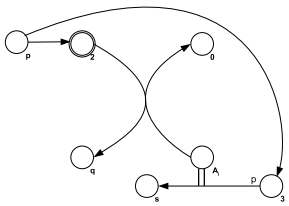

Fig. 7. Block arrangement for simulating a decrement instruction

For a decrement instruction (p, A_i-, q, s) of M, R_1 contains two uniport blocks $(\bullet, 1')(\bullet, p) \to (\bullet, 1')(\bullet, 2)$ and $(\bullet, 1')(\bullet, 2) \to (\bullet, 1')(\bullet, 3)$, i.e., we guess whether or not cell A_i contains a token. Furthermore, it contains a block arrangement which is a combination of a main block and a zero block. The main block moves a token from cell 2 to cell q and one token from cell A_i to the environment (if A_i contains at least one token and a token from cell p was sent to cell 2), or the token sent from cell p to cell 3 is forwarded to cell s (if A_i does not contain any token). In any other case, the constructed block arrangement implies the occurrence of an infinite loop. It is easy to see that the block arrangement described above simulates the application of the decrement instruction of M. Thus, any decrement instruction of M can be simulated, therefore the corresponding part of the rule set of Π can be constructed.

M halts in the final state q_f and the result of the generation is the number stored in its output register, A_h. To simulate the halting in M, we do not associate any rule to cells q_f, thus no further generation steps in Π can be performed.

At the beginning of the generation Π contains a token in cell q_0, $1'$, and in all the cells in the main blocks used for simulating the increment instructions and in the block arrangements used for simulating the decrement instructions which should initially contain at least one token. Due to the construction of R_1, the generation in Π is governed by cells p which correspond to the instructions of M to be performed, therefore any successful generation in Π corresponds to a successful generation in M and conversely. This implies that $N(M) = N(\Pi)$.

Over a one-symbol alphabet of objects, any chain rule corresponds to a presence-move rule, therefore the following statement is a direct consequence of Theorem 3.

Corollary 1. $NOtP_{*,1}(chain) = NRE$.

Next we describe the generative power of minimal interaction P systems with conditional-uniport-in rules.

Let $\Pi = (O, E, w_1, \ldots, w_r, R_1, h)$, $r \geq 1$, be a one-symbol minimal interaction P system with conditional-uniport-in rules. Since the conditional-uniport-in rules involve only two cells, to any configuration c of Π, we can assign a directed graph $G(c)$ where node i represents the cell i of Π, $1 \leq i \leq r$. The edges of G are determined by the rules of Π, i.e., if there is a rule of the form $p : (\bullet, i)(\bullet, j) \to (\bullet, i)(\bullet, i)$ in R_1, and the cell i is not empty, then there is a directed edge from node j to node representing i. We call this graph the *communication graph* of Π in configuration c. Note that if cell i contains no token, then, despite that $p \in R_1$, $G(c)$ has no edge from node j to node i. In this case we say that the edge from node j to node i has been broken.

In what follows, if no confusion arises, we use Π and graph G, in particular, the terms node/cell and edge/rule as equivalent.

Theorem 4. *For any GCPSMI Π with conditional-uniport-in rules either $N(\Pi)$ is finite or there is a natural number K such that $l \in N(\Pi)$ for every $l \geq K$.*

Proof. The proof is organized as follows. First, we introduce the notion of an *alive* path which informally corresponds to a path from the environment to the output cell in the communication graph of the initial configuration. When no such path exists, obviously $N(\Pi)$ is finite. In the other case we show that it is possible to bring any number of tokens from the environment to the output cell. We also show that any alive path can be broken, hence this process can be stopped. If the tokens can possibly go out from the output cell then depending on whether it is possible to break all outgoing edges or not, $N(\Pi)$ will be unbounded or empty.

Now we give some technical details. Let $\Pi = (O, E, w_1, \ldots, w_r, R_1, h)$, $r \geq 1$, be a one-symbol minimal interaction P system with conditional-uniport-in rules. Without the loss of any generality we may assume that at the initial configuration every cell contains at least one token, since due to the forms of the rules, if the cell is empty, then it will remain empty during the generation.

We first show that if $N(\Pi)$ is an infinite set, then Π should have an alive path. We call a sequence of cells $p = i_0, i_1, \ldots, i_{l_p}$, where i_0 denotes the environment and $i_{l_p} = h$, an *alive* path (of length l_p) to output cell h if a token $(\bullet)$ from the environment can move to cell h in l_p steps via the cells of p in the given order. This means that after l_p steps, Π will reach a configuration c where any cell in p contains at least two tokens and there is a directed edge (i_j, i_{j+1}), in the graph G representing c, i.e., any cell i_{j+1} can import a token from cell i_j, $0 \leq j \leq l_p - 1$. It is easy to see that using the two tokens and the corresponding rules in the cells, an arbitrary number of tokens can move to cell h via the path. Obviously, if Π has no alive path, then the set of numbers generated by Π is finite, since during the generation only a finite number of tokens can move to h.

However, the tokens arriving at the output cell may leave it, therefore the existence of an alive path does not imply the infinity of $N(\Pi)$. For this reason, we examine the possible movement of the tokens from the output cell.

A node k for which there is an edge (h, k) in the representing graph G of Π in some configuration c is called a *neighbor* of node h in c. We denote by $NBR(h, c)$

the set of all neighbors of a node h in configuration c and by $NBR(h)$ the set of all nodes which are neighbors of h during any generation.

We also recall that a knot in a graph is a subset of nodes X such that for every edge (i, j), $i \in X$ it holds $j \in X$, i.e., it is not possible to leave X.

The reader may observe that $L(\Pi) = \{0\}$ or $L(\Pi) = \emptyset$ if the following condition holds:

1. In any configuration c, there exists a set of nodes X with $X \subseteq (NBR(h, c) \cup NBR(i_0, c) \setminus \{i_1\})$, such that X forms a knot.

In particular, if X is a set of neighbors of the environment which does not contain i_1, the first node from the alive path p, that is, $X \subseteq (NBR(i_0, c) \setminus \{i_1\})$, then $L(\Pi) = \emptyset$, otherwise $L(\Pi) = \{0\}$.

This means that if $L(\Pi)$ is an infinite set, than there is an alive path from i_0 to h, such that the condition above does not hold.

Now we show that the existence of such an alive path implies that there exists a constant K such that $l \in L(\Pi)$ for any $l \geq K$. Suppose that there exists an alive path p in Π which does not satisfy condition 1 above. Then there is a generation in Π which permits to bring any number of new tokens in the output cell as follows: a token that was imported at the first time from the environment by cell i_1 (at the first configuration change in cell i_1) at the next step will be used for bringing in one other token from the environment. Then, after $|p|$ steps the token that first entered from the environment arrives at cell h and at the same time each internal cell in the path, i.e., cells $i_1, \ldots, i_{l_p-1}$ will contain at least two tokens.

Now we should prove that after some point the process can stop at any time. Our assertion is based on the following observation. If there are two cells a (with n tokens) and b with one token, and there is a rule $(\bullet, b)(\bullet, a) \rightarrow (\bullet, b)(\bullet, b)$ (i.e. an edge (a, b)), then by using all tokens present in b at every step, after at most n steps all tokens in cell a can be transported to cell b. This observation comes from the fact that the number of tokens in cell b after k steps is 2^k, while the number of tokens in cell a can be at most $2^k(n - k)$. Using this procedure, it is possible to break an edge in the graph $G(c)$ representing Π in some configuration c, i.e., to obtain a configuration where the communication graph has not this edge anymore.

This observation implies that for an alive path $p = i_0 \ldots i_{l_p} = h$ of Π it is possible to break all edges in the graph which are different from the edges $(i, i + 1)$, $1 \leq i \leq l_p - 1$. (We also break the edges of all other possible alive paths: edge by edge, starting from the one linked to the environment.) After that, using a similar procedure for every node in p, we may obtain a configuration that all cells belonging to the path p, except i_0 and h, contain exactly two tokens.

Finally, since condition 1 does not hold, there is no subset of neighbor nodes of h which form a knot, for any configuration c and every node $k \in NBR(h, c)$ either there is a node $k' \in NBR(k, c)$ but $k' \notin NBR(h, c)$, or there is a path from k consisting of neighbors of h to some node $k' \in NBR(h, c)$ such that it has a neighbor $k'' \in NBR(k', c)$ which is not neighbor of h ($k'' \notin NBR(h, c)$).

We observe that in the first case node k can be emptied, thus the edge (h, k) can be broken, while in the second case all tokens can move from k to k' (for all possible k) which reduces to the first case.

In a similar way, since condition 1 does not hold in any configuration c, it is possible to break all edges (i_0, k), $k \in (NBR(i_0, c) \setminus \{i_1\})$. Then, it is possible to break at the same time the edges going out from h as well as edges going out from the environment (i_0).

Hence, at this moment there have remained only one alive path p, where each cell contains exactly two tokens. Moreover, in this configuration h has no neighbor, and the only neighbor of the environment is i_1. Let $K - 1$ be the number of tokens at cell h at this moment. Take any $l \geq K$. In order to obtain l it is enough to perform $l - K + 1$ generation steps where every cell i_k belonging to p brings one token from i_{k-1} and sends one token to i_{k+1} and after that bring two tokens to cell h, thus breaking the last edge. Thus, h will contain l tokens. The generation will stop some steps later as follows. Firstly, the path p is broken edge by edge, starting from the edge linking to the environment. This concludes the proof.

4 Conclusions

It is a well-known fact that register machines are able to generate any recursively enumerable set of natural numbers. These machines operate over a one letter alphabet, i.e., use only the symbol stored in several copies in the registers. Our results provide computational complete computing devices over the simplest alphabet with other types of simple architectures. Since the concept of four types of the restricted communication rules were not applicable for one-symbol minimal interaction P systems, but GCPSMIs with three of them were computational complete if no restriction was put on the size of the object alphabet, the following problem arises. What is the minimal number k such that GCPSMIs with a given type of rule (except antiport1) with object alphabet of size at most k are computationally complete? In [4] constant upper bounds were presented on the number of cells in the GCPSMIs which were computationally complete. It is as interesting question what can we say on the generative power of one-symbol GCPSMI-s having a number of cells limited by a constant.

Another important topic for future research is the relation between one-symbol minimal interaction P systems and Petri nets. Similar questions have already been studied in [3].

Acknowledgment

We express our gratitude to the referees for their useful comments and suggestions.

References

1. Alhazov, A., Freund, R., Oswald, M., Verlan, S.: Partial versus total halting in P systems. In: Gutiérrez-Naranjo, M.A., Păun, Gh., Romero-Jiménez, A., Riscos-Núnez, A. (eds.) Proceedings of the Fifth Brainstorming Week on Membrane Computing, pp. 1–20, Sevilla (2007)
2. Alhazov, A., Freund, R., Rogozhin, Y.: Computational power of symport/antiport: History, advances, and open problems. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers. LNCS vol. 3850, pp. 1–30, Springer (2006)
3. Bernardini, F., Gheorghe, M., Margenstern, M., Verlan, S.: Producer/consumer in membrane systems and Petri nets. In: Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18-23, 2007, Proceedings. LNCS vol. 4497, pp. 43–52, Springer (2008)
4. Csuhaj-Varjú, E., Verlan, S.: On generalized communicating P systems with minimal interaction rules. (Submitted, 2009)
5. Frisco, P.: Computing with Cells. Oxford University Press (2009)
6. Martín-Vide, C., Păun, Gh., Pazos, J., Rodríguez-Patón, A.: Tissue P systems. Theoretical Computer Science 296(2), 295–326 (2003)
7. Minsky, M.: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, New Jersey (1967)
8. Păun, A., Păun, Gh.: The power of communication: P systems with symport/antiport. New Generation Computing 20, 295–305 (2002)
9. Păun, Gh.: Membrane Computing. An Introduction. Springer, Berlin (2002)
10. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)
11. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, volume 1–3, Springer (1997)
12. The P systems webpage:http://ppage.psystems.eu
13. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: On communication in tissue P systems: conditional uniport. In: Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers. LNCS vol. 4361, pp. 521–535, Springer (2006)
14. Verlan, S., Bernardini, F., Gheorghe, M., Margenstern, M.: Generalized communicating P systems. Theoretical Computer Science 404(1-2), 170–184 (2008)