Further Results on Generalised Communicating P Systems

Shankara Narayanan Krishna[a], Marian Gheorghe[b,*], Florentin Ipate[c], Erzsébet Csuhaj-Varjú[d], Rodica Ceterchi[c]

[a] *Department of Computer Science and Engineering*
Indian Institute of Technology Bombay, Powai
Mumbay 400076, India
[b] *School of Electrical Engineering and Computer Science, University of Bradford*
Bradford BD7 1DP, United Kingdom
[c] *Department of Computer Science, Faculty of Mathematics and Computer Science*
University of Bucharest
Str Academiei 14, 70109, Bucharest, Romania
[d] *Department of Algorithms and Their Applications*
Faculty of Informatics, Eötvös Loránd University
Budapest 1117, Hungary

Abstract

In this paper we consider four restricted cases of the generalised communicating P systems and study their computational power, by providing improved results, with respect to the number of compartments involved. We illustrate the expressive power of these devices by modelling several problems, such as producer/consumer, workflow patterns, broadcasting problem and comparative operations. We also present some relationships between generalised communicating P systems and P colonies, tissue-like P systems with very simple components.

Keywords: Generalised communicating P systems, register machine, computational power, producer/consumer, workflow patterns, broadcasting problem, comparative operations, P colony

1. Introduction

Membrane computing represents a branch of natural computing that brings a set of concepts and principles from cellular biology to computer science, with the aim of producing a family of coherent, powerful and efficient computational models, called *membrane systems* or (*P systems*), that are inspired by the behaviour of some cellular processes. These models include various computational paradigms like non-deterministic, parallel and distributed

*Corresponding author
Email addresses: krishnas@cse.iitb.ac.in (Shankara Narayanan Krishna), m.gheorghe@bradford.ac.uk (Marian Gheorghe), florentin.ipate@ifsoft.ro (Florentin Ipate), csuhaj@inf.elte.hu (Erzsébet Csuhaj-Varjú), rceterchi@gmail.com (Rodica Ceterchi)

calculus that mimic, through a set of evolution and communication rules applied in different compartments, the behaviour of various bio-chemical systems defining key functions of the living cells [21].

Professor Marcus develops in [14] a taxonomy of five characteristics of a living system, and concludes: "As we can see, membranes are involved in four of the above five steps. The scenario above calls attention to the fact that the closure of a membrane around some autocatalytic chemical reaction system is an attractive candidate for a first step towards the origin of a living system".

One of the most studied P system models is called *cell-like* P system, and consists of a *hierarchical structure of membranes* (tree structure) delimiting *compartments* (or *regions*); each compartment contains a *multiset of objects* that react according to a set of *rules* belonging to the compartment. The rules of the most basic cell-like P systems are of the form $u \to v$, where u and v are multisets of objects. The multiset v consists of objects that remain in this compartment and others that will go into the compartment that contains the current one (the parent compartment) or to compartments contained in it (child compartments). When such a rule is applied to a multiset it replaces u, if it is contained in the multiset, by v. Other types of objects (strings instead of multisets), rules (using states or activators/inhibitors, considering electrical charges, dissolving or creating compartments etc.) and connections between compartments (arbitrary or specific graphs) have been considered ([21], Chapters 4, 5, 7 - 11, 13, 14). Some of these models replace the hierarchical structure of membranes with a graph structure and the model is called *tissue-like*. Most of these systems are computationally complete and when membranes can be multiplied they are able to provide efficient solutions, with respect to time, to complex problems ([21], Chapters 12, 21). Membrane computing has introduced a plethora of variants of P systems and the above mentioned enumeration of such systems reflects only some of the many existing models.

A special type of P systems emphasises the communicating aspects of these models by using different rules to transport objects across membranes ([21], Chapter 5). One of these models uses the biological metaphor of exchanging pairs of bio-chemical elements between compartments, and has been called *symport/antiport* P systems [18]. A special case of such communicating mechanisms has been considered for *generalised communicating P systems* [26], where only communication rules are used, but in a very general way. They are simultaneously moving symbols from two compartments into other two. This model is inspired by both the symport/antiport paradigm and the way transitions of the Petri nets fire tokens coming from various input places and then send them out to other output places [23]. The model has been introduced in [26] and further investigated in [8, 9, 13, 12, 1].

The key contributions of the paper are: (a) the improvement of the computational power results presented in [9], with respect to the number of compartments; (b) a set of examples illustrating the modelling capabilities of these classes of P systems in specifying problems such as producer/consumer, workflow patterns, broadcasting problem and comparative operations; and (c) some relationships between generalised communicating P systems and P colonies, variants of tissue-like P systems with very simple cells acting and evolving in a

shared environment. Theorems 1 - 4 have been for the first time introduced in [12, 13], the proof of Theorem 1 in [13] and of the others in the technical report [12]. In this paper these proofs have been all revised and are presented in a more compact and clearer form than in the previous publications.

The paper consists of four key sections. Section 2 introduces the key concepts and definitions of the paper. Section 3 presents the computational power results of the generalised communicating P systems. Several applications of this computational model are described in Section 4 and relationships with P colonies are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. Definitions

In this section we introduce the definitions of the main concepts utilised in this paper, generalised communicating P systems, P colonies and register machines.

A *Generalised Communicating P System of degree* n (a GCPS *of degree* n) is a P system consisting of n compartments, called *cells*, linked in a tissue-like manner. These links are implicitly specified by the rules. Formally, a GCPS of degree n is a construct,

$$\Pi = (O, E, w_1, \ldots, w_n, R_\Pi, i_0),$$

where O is a *finite alphabet*; $E \subseteq O$ is the set of *environment symbols*; $w_i \in O^*$, $1 \leq i \leq n$, is the *initial multiset* associated with cell i; R_Π is a finite set of *interaction rules* of the form $r : (a, i \rightarrow k;\ b, j \rightarrow l)$ (also written as $r : (a, i)(b, j) \rightarrow (a, k)(b, l)$), with $a, b \in O$ and $0 \leq i, j, k, l \leq n$ (0 denotes the environment), and such that if $i = j = 0$ then at least one of a and b is not in E; and $i_0 \in \{1, \ldots, n\}$ is the *output cell*.

The P system Π consists of n cells, labelled $1, \ldots, n$, containing multisets of objects over O. The environment contains an unbounded number of copies of symbols from E. The cells, and the environment, are supposed to interact through rules from R_Π. A rule $r : (a, i)(b, j) \rightarrow (a, k)(b, l)$, moves a from cell i to k and b from cell j to l. The rules are applied in each step of the computation in the usual non-deterministic and maximally parallel manner. According to maximal parallelism principle, after associating objects to rules, no rule can be applied to any of the remaining objects, if any (see [21]). As usual in membrane systems, the multisets will be denoted by strings and the empty multiset denoted by λ (for further details see [21]).

A *configuration* of Π is an $n + 1$-tuple $(x_E, x_1, \ldots, x_n)$, where $x_E \in (O \setminus E)^*$ and x_i is the multiset from cell i, $1 \leq i \leq n$. If c is a configuration of Π then a *computation step* or *transition* is the process of obtaining a new configuration c' from c by applying the rules of R_Π in a maximally parallel manner. A *computation* is a sequence of transitions. Only halting computations are considered. The result of any computation, obtained in cell i_0, is given by the number of objects from this cell at the end of the computation. The set of non-negative integer numbers computed by the GCPS Π, as the number of symbols obtained in the output cell i_0, is denoted by $N(\Pi)$. The family of sets of numbers generated by GCPS with at most n cells is denoted by $NGCPS_n$.

The functionality described by each rule $r : (a, i \to k; b, j \to l)$, is similar to that of a transition in a Petri net model [23], in which tokens a and b from the input places are then moved to output places. Apart from moving objects between compartments, the rules can increase or decrease the number of objects in the system by involving the environment.

Some variants of the GCPS models have been defined in [26] and their computational power studied in [9]. They utilise a set of particular rules. In this paper we refer to: (a) *join rules* $(k = l, i \neq k, j \neq k, i \neq j)$; (b) *split rules* $(i = j, i \neq k, i \neq l, k \neq l)$; (c) *presence_move rules* $(i = k, i \neq l, i \neq j, j \neq l)$; and (d) *parallel_shift rules* $(i \neq k, i \neq l, i \neq j, j \neq l)$. We call these classes of systems, *GCPS with minimal interaction*.

If only rules of type (a), (b), (c) or (d) are used then the corresponding family of sets of numbers computed by these GCPS devices with minimal interaction is denoted by $NGCPS(t)_n$, $t \in \{join, split, presence\_move, parallel\_shift\}$.

P colony [7] represents a simple membrane system model with communities of cells communicating with a shared environment. Here we consider a restricted version of it, called *P colony without checking rules*. More on this model can be found in [11].

A *P colony* (without checking rules) is an $n + 3$-tuple, $n \geq 1$

$$\Pi = (O, e, F, C_1, \ldots, C_n),$$

where O is an alphabet (the alphabet of *objects*), $e \in O$ (the *environment object*), and $F \subseteq O$ (the set of *final objects*). Each pair $C_i = (O_i, P_i)$, $1 \leq i \leq n$, is called a cell of Π, where O_i is a multiset over $\{e\}$ having the same cardinality (called *capacity*) for every C_i, and P_i is a finite set of *programs*. Each program consists of a finite multiset of rules of the following forms: (a) $a \to b$ (internal point mutation or evolution), specifying that an object $a \in O$ inside the cell is changed to $b \in O$; (b) $c \leftrightarrow d$ (one object exchange with the environment or communication), specifying that if $c \in O$ is contained inside the cell and $d \in O$ is present in the environment, then c is sent out of the cell to the environment while d is brought inside the cell from the environment. The number of rules in each program of C_i coincides with the capacity of Π.

An $n + 1$-tuple $(x_E, x_1, \ldots, x_n)$, where $x_E \in (O \setminus \{e\})^*$, $x_i \in O_i^*$ are finite multisets, is called a *configuration* of Π. At the *initial configuration*, the environment contains arbitrarily many copies of e and each cell contains inside as many objects e as the capacity of Π.

The P colony works with direct changes of its configurations, called *transitions*. To obtain a new configuration by a transition, the programs of the cells are used in the non-deterministic maximally parallel manner, i.e., each cell which is able to use one of its programs should use one. The use of a program means the parallel application of the rule(s) of the program to the object(s) inside the cell. A sequence of transitions starting from the initial configuration is a *computation*. A computation is successful if it is *halting*, i.e., if a configuration is obtained where no cell can use any program. The *result* of a successful computation is the number of copies of objects from F present in the halting configuration. The set of numbers obtained as results of successful computations of a P colony Π is denoted by $N(\Pi)$.

A *register machine* with k registers is a 5-tuple,

$$M = (Q, R, q_0, q_f, P),$$

where Q is a finite non-empty 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 the *set of instructions* of the following forms: (a) (p, A_i^+, q, s) where $p, q, s \in Q$, $p \neq q_f$, $A_i \in R$ (an *increment instruction*, which increments register A_i by 1 and moves non-deterministically to either q or s); (b) (p, A_i^-, q, s) where $p, q, s \in Q$, $p \neq q_f$, $A_i \in R$ (a *decrement instruction*, which decrements register A_i by 1 and moves to q, if strictly positive, otherwise left it unchanged and jumps to state s). For every $p \in Q, (p \neq q_f)$, there is exactly one instruction of the form either (p, A_i^+, q, s) or (p, A_i^-, 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 $A_1, \ldots, A_k$, respectively. We say that a register machine M with k registers *generates* a non-negative integer u if starting from the initial configuration $(q_0, 0, 0, \ldots, 0)$ it enters the final configuration $(q_f, u, 0, \ldots, 0)$. The set of non-negative integers generated by M is denoted by $N(M)$. It is known that register machines are able to generate all recursively enumerable sets of non-negative integers [16], denoted by NRE.

3. Main results

In [9] it is proved that GCPS devices with minimal interaction achieve universality for systems using at most: (a) 7 cells and *join rules*; (b) 9 cells and *split rules*; (c) 19 cells and *parallel_shift rules*; and (d) 36 cells and *presence_move rules*. We improve in the sequel all these results, by showing that universality can be achieved for these systems when at most 4, 5, 5 and 6 cells are, respectively, used.

The proofs of these results have been initially provided in [12] and the proof for join rules appeared initially in [13]. In this paper they have been revised and are presented in a more compact and clearer form than in the previous publications.

3.1. GCPS with minimal interaction - join

The proof below uses for an arbitrary set in NRE a register machine which will be simulated by a GCPS with at most 4 cells and using only *join rules*, which are such that $k = l, i \neq k, j \neq k, i \neq j$, i.e., distinct source cells and the same destination cell.

Theorem 1. $NGCPS(join)_4 = NRE$.

Proof. Consider a register machine $M = (Q, R, q_0, q_f, P)$, as defined in Section 2, with Q a finite non-empty set of states, $R = \{A_1, \ldots, A_k\}$, $k \geq 1$, a set of registers, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state and P is the set of instructions. We construct a GCPS with minimal interaction, $\Pi = (O, E, w_1, w_2, w_3, w_4, R_\Pi, 2)$, using only *join rules*, such that $N(M) = N(\Pi)$. It will be shown that for any computation in M there is a halting computation of the constructed GCPS, Π, and no unexpected computations are allowed in Π.

We define the following two sets

- $Q^+ = \{p^+ \mid p \text{ is the state corresponding to an increment instruction}\}$,
- $Q^- = \{p^- \mid p \text{ is the state corresponding to a decrement instruction}\}$.

Given the register machine M, the GCPS with minimal interaction, Π, is constructed with the elements below.

- $O = Q^+ \cup Q^- \cup \{\overline{q} \mid q \in Q\} \cup \{q_f, c_1, \ldots, c_k\} \cup \{f, g, a, b, e, Y_1, Y_2, Y_3, \dagger, \chi\}$ is the alphabet of the system; the number of symbols c_i, $1 \leq i \leq k$, in Π represents the content of register A_i at any given point in time; $f, g, a, b, e, Y_1, Y_2, Y_3, \dagger, \chi$ are auxiliary symbols and help the computation.
- $E = Q^+ \cup Q^- \cup \{\overline{q} \mid q \in Q\} \cup \{\dagger, g, q_f\} \cup \{c_i \mid 1 \leq i \leq k\}$ is the set of symbols present in the environment in infinitely many copies.
- $w_1 = fab\chi, w_2 = \lambda, w_3 = Y_2Y_3, w_4 = eY_1$ are the initial multisets of Π.
- Cell 2 is the output cell.
- The set of rules, R_Π, consists of
  - **I.** *Initialisation rules*:

    I.1 $(a,1)(Y_2,3) \to (a,0)(Y_2,0)$, $(b,1)(Y_3,3) \to (b,0)(Y_3,0)$, $(q_0^+,0)(e,4) \to (q_0^+,1)(e,1)$.
  - **II.** *For each increment instruction, (p, A_i^+, q, s)*, if $r \in \{q, s\}$, then in the rules below $r^\star$ stands for r^+ or r^-, depending on whether r is the state corresponding to an increment instruction or a decrement instruction:

    II.1 $(p^+,1)(c_i,0) \to (p^+,2)(c_i,2)$;

    II.2 $(p^+,2)(\overline{r},0) \to (p^+,3)(\overline{r},3)$;

    II.3.1 $(p^+,3)(Y_2,2) \to (p^+,0)(Y_2,0)$; **II.3.2** $(\overline{r},3)(r^\star,0) \to (\overline{r},1)(r^\star,1)$.
  - **III.** *For each decrement instruction, (p, A_i^-, q, s)*, the following rules are added to R_Π:

    III.1 $(p^-,1)(c_i,2) \to (p^-,4)(c_i,4)$;

    III.2 $(p^-,4)(\overline{q},0) \to (p^-,3)(\overline{q},3)$;

    III.3.1 $(p^-,3)(Y_2,2) \to (p^-,0)(Y_2,0)$, **III.3.2** $(\overline{q},3)(q^\star,0) \to (\overline{q},1)(q^\star,1)$, where $q^\star$ stands for q^+ or q^-;

    III.4 $(p^-,1)(Y_2,2) \to (p^-,3)(Y_2,3)$;

    III.5 $(p^-,3)(f,1) \to (p^-,2)(f,2)$;

    III.6 $(p^-,2)(\overline{s},0) \to (p^-,3)(\overline{s},3)$;

    III.7 $(p^-,3)(Y_2,4) \to (p^-,0)(Y_2,0)$, $(\overline{s},3)(s^\star,0) \to (\overline{s},1)(s^\star,1)$, where $s^\star$ stands for s^+ or s^-.

– **IV.** *For the halting instruction corresponding to state q_f*, the following rule is added to R_Π:

IV.1 $(q_f, 1)(Y_1, 4) \to (q_f, 0)(Y_1, 0)$.

– **V.** *Auxiliary rules*:

V.1 $(Y_1, 4)(Y_2, 0) \to (Y_1, 2)(Y_2, 2)$;
V.2 $(Y_1, 2)(Y_3, 0) \to (Y_1, 4)(Y_3, 4)$;
V.3 $(f, 2)(Y_2, 3) \to (f, 4)(Y_2, 4)$;
V.4 $(g, 0)(f, 4) \to (g, 1)(f, 1)$;
V.5 $(f, 2)(\dagger, 0) \to (f, 3)(\dagger, 3)$;
V.6 $(\chi, 1)(\dagger, 3) \to (\chi, 2)(\dagger, 2)$;
V.7 $(\chi, 2)(\dagger, 0) \to (\chi, 3)(\dagger, 3)$;
V.8 $(\chi, 3)(\dagger, 0) \to (\chi, 2)(\dagger, 2)$;
IV.9 $(\overline{p}, 1)(Y_3, 4) \to (\overline{p}, 0)(Y_3, 0)$.

In the sequel we describe successful computations in Π corresponding to successful generation of non-negative numbers in M.

Initialisation step.
The computation in Π starts from the initial configuration, given by $(x_E, fab\chi, \lambda, Y_2Y_3, eY_1)$, with $x_E = \lambda$, by applying the *initialisation* rules, (I.1), in one step. The objects Y_2, Y_3 from cell 3 and a, b from cell 1 go to the environment, while the state of the initial instruction, q_0^+, from the environment and e from cell 4 come to membrane 1. Formally, the new configuration of Π is $(abY_2Y_3, efq_0^+\chi, \lambda, \lambda, Y_1)$. This configuration appears before any of the instructions of the register machine is simulated. The symbols a and b from the environment will be included in the current environment contents $x_E \in (O \setminus E)^*$.

Before we begin the simulation of the register machine instructions, let us observe that the current configuration of Π, corresponding to the start of the simulation of a register machine instruction, $(p, A_i^\star, q, s)$, where $A_i^\star$ is either A_i^+ or A_i^-, has the form $(x_EY_2, eg^\alpha f\overline{p}p^\star\chi, w, \lambda, Y_1Y_3w')$, where $p^\star$ is either p^+ or p^-, $w = c_1^{l_1} \dots c_k^{l_k}$, $l_i \geq 0$, $1 \leq i \leq k$, $w' = c_1^{l'_1} \dots c_k^{l'_k}$, $l'_i \geq 0$, $1 \leq i \leq k$, $\alpha \geq 0$. Please note that objects g appear in cell 1 when a decrement instruction cannot be applied and objects of w' appear in cell 4 when a decrement instruction is applied. Also, $\overline{p}$ appears after at least one instruction simulation takes place.

Simulation of an increment instruction (p, A_i^+, q, s).
Initially, when $p = q_0$, the system configuration is given by the result of the initialisation step, i.e., $(x_EY_2Y_3, efq_0^+\chi, \lambda, \lambda, Y_1)$. In general, when $p \neq q_0$, then the system configuration is $(x_EY_2, eg^\alpha f\overline{p}p^+\chi, w, \lambda, Y_1Y_3w')$.

Now we show the sequence of steps in Π while simulating (p, A_i^+, q, s) from the above generic configuration of the system, with r denoting one of q or s and $r^\star$ standing for either r^+ or r^-. The configuration $(x_EY_2, eg^\alpha f\overline{p}p^+\chi, w, \lambda, Y_1Y_3w')$ evolves into $(x_EY_3, eg^\alpha f\chi,$ $p^+c_iwY_1Y_2,\ \lambda, w')$ by using rules II.1, V.9 and rule V.1 (when $q = q_0$, the rule V.9 is not

applicable, as $\overline{p}$ does not appear in cell 1). From the last configuration using rules II.2 and V.2 we obtain the configuration $(x_E, eg^\alpha f\chi, c_i w Y_2, p^+\overline{r}, Y_1Y_3w')$ and this one in turn evolves into $(x_E Y_2, eg^\alpha f\overline{r}r^\star\chi, c_i w, \lambda, Y_1Y_3w')$ by applying rules II.3.1 and II.3.2. This configuration shows that the increment instruction applied to register A_i has been simulated by bringing a c_i into cell 2. The other cells and the environment have returned to the values they had at the start of the increment instruction simulation, now with $\overline{r}r^\star$ in cell 1.

Simulation of a decrement instruction (p, A_i^-, q, s).

We start from a configuration $(x_E Y_2, eg^\alpha f\overline{p}p^-\chi, c_i^l w, \lambda, Y_1Y_3w')$, where $l \geq 0$ (i.e., symbol c_i appears in zero or more copies in cell 2).

Case 1. The content of register A_i is non-zero ($l > 0$). In this case, we apply rules III.1 and V.9 and a rule V.1 which lead to p^- from cell 1 and a copy of c_i from cell 2 moving together to cell 4 (III.1), Y_1 from cell 4 with Y_2 from environment move to cell 2 (V.1) and $\overline{p}$ from cell 1 and Y_3 from cell 4 move to environment (V.9). This step leads to a configuration $(x_E Y_3, eg^\alpha f\chi, wY_1Y_2, \lambda, p^- c_i w')$. Then p^- from cell 4 and symbol $\overline{q}$ from the environment go to cell 3 (rule III.2) and in parallel, the symbol Y_1 from cell 2 and Y_3 from environment go to cell 4 (V.2), leading to a configuration $(x_E, eg^\alpha f\chi, wY_2, p^-\overline{q}, Y_1Y_3w'')$, where $w'' = c_i w'$. Next, the symbols p^-, Y_2 return to the environment (III.3.1), while $\overline{q}$ from cell 3 and $q^\star$ from the environment go to membrane 1 (III.3.2), producing a new configuration $(x_E Y_2, eg^\alpha f\overline{q}q^\star\chi, w, \lambda, Y_1Y_3w'')$, which is ready for the simulation of another register machine instruction.

Case 2. The content of register A_i is zero ($l = 0$). In this case the rule III.1 can no longer be applied as c_i does not appear in cell 2, although p^- is in cell 1. However, symbols Y_1, Y_2 move to cell 2 using the rule V.1 and $\overline{p}$ from cell 1 and Y_3 from cell 4 move to environment (V.9), hence we get to the configuration $(x_E Y_3, eg^\alpha fp^-\chi, wY_1Y_2, \lambda, w')$. In this case, we have the symbol p^- in cell 1, and Y_2 in cell 2. Then we use rule III.4, moving both p^-, Y_2 to cell 3 and, in parallel, symbols Y_1, Y_3 move to cell 4 by rule V.2, leading to the configuration $(x_E, eg^\alpha f\chi, w, p^-Y_2, Y_1Y_3w')$. This is followed by p^- moving to cell 2 along with f from cell 1 using rule III.5. In the next step p^- in cell 2 and $\overline{s}$ from the environment move to cell 3 (III.6) and, in parallel, symbols f, Y_2 move to cell 4 (rule V.3). After these two steps the configuration $(x_E, eg^\alpha\chi, w, p^-\overline{s}, Y_1Y_2Y_3fw')$ is obtained. Next, p^-, Y_2 return to the environment from cells 3 and 4, respectively, while $\overline{s}$ from cell 3 and $s^\star$ from the environment move to cell 1 (rules III.7) and the symbol f from cell 4 goes back to cell 1 together with g from the environment (rules V.4). The newly obtained configuration is $(x_E Y_2, eg^{\alpha+1} f\overline{s}s^\star\chi, w, \lambda, Y_1Y_3w')$, which is ready for the simulation of the register machine instruction corresponding to state s.

Exception Handling.

Note that in Case 1 in the configuration $(x_E, eg^\alpha f\chi, wY_2, p^-\overline{q}, Y_1Y_3w')$, instead of using both rules III.3.1 and III.3.2, one could use the rules III.5 and III.3.2. If this is done we get the configuration $(x_E, eg^\alpha\overline{q}q^\star\chi, fp^-wY_2, \lambda, Y_1Y_3w')$. Now, rules V.9 and III.6 can be applied, but as we have f and Y_2 both in cell 2, the rule V.3 is no longer applicable and rule V.5 is used instead, producing the configuration $(x_E Y_3, eg^\alpha q^\star\chi, wY_2, p^-\overline{s}f\dagger, Y_1w')$. Now, using V.6, the

symbol χ is moved to cell 2, and this produces a non-halting computation, where rules V.7 and V.8 are indefinitely applied.

Halting step.
Once we obtain q_f in cell 1, the symbol Y_1 is removed from the system to the environment using the rule IV.1. This way, the chain of actions $(Y_1, 4)(Y_2, 0) \to (Y_1, 2)(Y_2, 2)$, $(Y_1, 2)(Y_3, 0) \to (Y_1, 4)(Y_3, 4)$ can be stopped. The number of symbols $c_i, 1 \leq i \leq k$, in cell 2 is the output.

It is clear that every set of numbers generated by a register machine can be simulated by a GCPS using only *join rules*. □

3.2. GCPS with minimal interaction - split

We show now that using only *split rules* ($i = j, i \neq k, i \neq l, k \neq l$), i.e., the same source cell and distinct destination cells, and at most 5 compartments, register machines are simulated.

Theorem 2. $NGCPS(split)_5 = NRE$.

Proof. As in the proof of Theorem 1, we construct a GCPS with minimal interaction, $\Pi = (O, E, w_1, w_2, w_3, w_4, w_5, R_\Pi, 2)$, this one using only *split rules* and simulating a register machine M, such that $N(M) = N(\Pi)$. It will be shown that for any computation in M there is a halting computation of the constructed GCPS, Π, and no unexpected computations are allowed in Π.

Given a register machine, M, with k registers and the set of states $Q = \{p_1, \ldots, p_m\}$, we define Q^+, Q^- as in the proof of Theorem 1. The minimal interaction GCPS is constructed.

- $O = Q^+ \cup Q^- \cup \{c_i \mid 1 \leq i \leq k\} \cup \{p', p'', p''' \mid p \in Q \backslash \{q_f\}\} \cup \{q_f, X, X', \overline{X}\}$, is the alphabet of the system.
- $E = Q^+ \cup Q^- \cup \{c_i \mid 1 \leq i \leq k\}$, is the set of symbols present in the environment.
- $w_1 = q_0^+ X$, $w_2 = \lambda$, $w_3 = p_1' \ldots p_m' \overline{X}$, $w_4 = p_1'' \ldots p_m'' X'$, $w_5 = p_1''' \ldots p_m'''$.
- Cell 2 is the output cell.
- The rules R_Π are as follows
  - **I.** *For each increment instruction* (p, A_i^+, q, s), if $r \in \{q, s\}$, the rules below are introduced into R_Π, where $r^\star$ stands for r^+ or r^-, depending on whether r corresponds to an increment instruction or a decrement instruction.
    I.1 $(p^+, 1)(X, 1) \to (p^+, 3)(X, 4)$;
    I.2 $(p^+, 3)(p', 3) \to (p^+, 1)(p', 0)$;
    I.3 $(p^+, 1)(X', 1) \to (p^+, 0)(X', 5)$, $(p', 0)(c_i, 0) \to (p', 4)(c_i, 2)$;
    I.4 $(p', 4)(p'', 4) \to (p', 3)(p'', 0)$;
    I.5 $(p'', 0)(r^\star, 0) \to (p'', 4)(r^\star, 1)$.

– **II.** *For each decrement instruction* (p, A_i^-, q, s), the following rules are added to R_Π, where $q^\star \in \{q^+, q^-\}$ and $s^\star \in \{s^+, s^-\}$.

II.1 $(p^-, 1)(X, 1) \to (p^-, 2)(X, 3)$;
II.2 $(p^-, 2)(c_i, 2) \to (p^-, 4)(c_i, 0)$;
II.3 $(p'', 4)(p^-, 4) \to (p'', 0)(p^-, 1)$;
II.4 $(p^-, 1)(\overline{X}, 1) \to (p^-, 0)(\overline{X}, 2)$, $(p'', 0)(q^\star, 0) \to (p'', 4)(q^\star, 1)$;
II.5 $(X, 2)(\overline{X}, 2) \to (X, 1)(\overline{X}, 3)$;
II.6 $(p^-, 2)(X, 2) \to (p^-, 3)(X, 1)$;
II.7 $(X, 1)(\overline{X}, 1) \to (X, 2)(\overline{X}, 3)$;
II.8 $(p^-, 3)(\overline{X}, 3) \to (p^-, 5)(\overline{X}, 1)$;
II.9 $(p^-, 5)(p''', 5) \to (p^-, 1)(p''', 0)$;
II.10 $(p''', 0)(s^\star, 0) \to (p''', 5)(s^\star, 1)$, $(p^-, 1)(\overline{X}, 1) \to (p^-, 0)(\overline{X}, 2)$.

– **III.** *Auxiliary rules*

III.1 $(X, 4)(X', 4) \to (X, 5)(X', 1)$;
III.2 $(X, 5)(X', 5) \to (X, 1)(X', 4)$;
III.3 $(X, 3)(\overline{X}, 3) \to (X, 2)(\overline{X}, 1)$.

For any instruction of the register machine, a state $p \in Q$ is associated with it. For each state p, we associate a symbol p' in cell 3, a symbol p'' in cell 4 and a symbol p''' in cell 5. Note that the symbols p', p'', p''', for $p \in Q$, and $X, X', \overline{X}$ are present in the system only in one copy. Cell 2, the output cell, contains the contents of the registers.

Let us observe that the configuration of the system corresponding to the start of the simulation of an instruction $(p, A_i^\star, q, s)$, where $A_i^\star$ is either A_i^+ or A_i^-, is given by the contents of the environment, which is empty, and the five cells, i.e., $(\lambda, p^\star X, w, p't'\overline{X}, p''t''X', p'''t''')$, where $p^\star$ is either p^+ or p^-, t' denotes $p'_1 \dots p'_m$ without p' and similarly for t'' and t'''; $w = c_1^{l_1} \dots c_k^{l_k}$, $l_i \geq 0$, $1 \leq i \leq k$.

Simulation of an increment instruction (p, A_i^+, q, s).

The computation proceeds as follows: one starts from the configuration $(\lambda, p^+X, w, p't'\overline{X}, p''t''X', p'''t''')$ and the computation evolves into $(\lambda, \lambda, w, p't'\overline{X}p^+, p''t''X'X, p'''t''')$, by using the rule I.1. This in turn evolves into $(p', p^+X', w, t'\overline{X}, p''t'', p'''t'''X)$, by using in parallel the rules I.2 and III.1. Using in parallel the rules I.3, we get $(\lambda, \lambda, wc_i, t'\overline{X}, p''t''p', p'''t'''XX')$. Since the environment has infinitely many copies of the symbols from Q^+, we do not depict that p^+ has just joined the environment. Next, using I.4 and III.2 in parallel, we obtain $(p'', X, wc_i, p't'\overline{X}, t''X', p'''t''')$. Finally, using I.5, we get $(\lambda, r^\star X, wc_i, p't'\overline{X}, p''t''X', p'''t''')$. A new symbol c_i has been added to cell 2 and cells 3, 4 and 5 have restored the multisets they started with at the beginning of simulating the increment instruction. We are now ready to begin simulating the next instruction corresponding to state $r \in \{q, s\}$.

Simulation of a decrement instruction (p, A_i^-, q, s).

The computation starts from the configuration $(\lambda, p^-X, w, p't'\overline{X}, p''t''X', p'''t''')$. We distinguish two cases.

Case 1: The content of register A_i is non-zero, i.e., $w = c_iw'$. Starting from $(\lambda, p^-X, c_iw', p't'\overline{X}, p''t''X', p'''t''')$, we obtain $(\lambda, \lambda, c_iw'p^-, p't'\overline{X}X, p''t''X', p'''t''')$, by using the rule II.1. Further, using rules II.2 and III.3, we get $(\lambda, \overline{X}, w'X, p't', p''t''X'p^-, p'''t''')$. A copy of c_i is sent to the environment. Next, using II.3, we have $(p'', p^-\overline{X}, w'X, p't', t''X', p'''t''')$. Further, using rules II.4 in parallel, we obtain $(\lambda, q^\star, w'X\overline{X}, p't', p''t''X', p'''t''')$. Finally, II.5 restores X back to cell 1 and $\overline{X}$ back to cell 3, giving $(\lambda, q^\star X, w', p't'\overline{X}, p''t''X', p'''t''')$, and we are ready to simulate the next instruction.

Case 2: The content of register A_i is zero, i.e., w does not contain any c_i. In this case, we start again with II.1 obtaining $(\lambda, \lambda, wp^-, p't'\overline{X}X, p''t''X', p'''t''')$. As it has been noticed, w does not contain any c_i. The rule II.2 can no longer be applied, but III.3 is used and we get $(\lambda, \overline{X}, wp^-X, p't', p''t''X', p'''t''')$. Rule II.3 is not applicable now, but we use rule II.6 instead, obtaining $(\lambda, \overline{X}X, w, p't'p^-, p''t''X', p'''t''')$. Next, using II.7, we get $(\lambda, \lambda, wX, p't'p^-\overline{X}, p''t''X', p'''t''')$. Further, II.8 gives us $(\lambda, \overline{X}, wX, p't', p''t''X', p'''t'''p^-)$. Next, we obtain $(p''', \overline{X}p^-, wX, p't', p''t''X', t''')$, using II.9. This evolves into $(\lambda, s^\star, wX\overline{X}, p't', p''t''X', p'''t''')$, using II.10. Now, using II.5, $X, \overline{X}$ are restored to their usual positions and we obtain $(\lambda, s^\star X, w, p't'\overline{X}, p''t''X', p'''t''')$.

When we obtain the halting state q_f, the system halts. The contents of cell 2 is then the output of Π. □

3.3. GCPS with minimal interaction – parallel_shift

We recall that *parallel_shift* rules are such that $i \neq k, i \neq l, i \neq j, j \neq l$. We show below that 5 compartments are enough for simulating a register machine with such rules.

Theorem 3. $NGCPS(parallel\_shift)_5 = NRE$.

Proof. As in the earlier two results, for a given register machine, M, we construct a GCPS with minimal interaction, $\Pi = (O, E, w_1, w_2, w_3, w_4, w_5, R_\Pi, 2)$, using only *parallel shift rules* and simulating M. It will be shown that for any computation in M there is a halting computation of the constructed GCPS, Π, and no unexpected computations are allowed in Π. Define Q^+ and Q^- as in the proofs of the previous theorems.

- $O = Q^+ \cup Q^- \cup \{\overline{q}, \widehat{q} \mid q \in Q \backslash \{q_f\}\} \cup \{q_f, c_1, \ldots, c_k\} \cup \{a, b, e, Y_1, Y_2, Y_3\}$, is the alphabet of the system.
- $E = Q^+ \cup Q^- \cup \{c_1, \ldots, c_n\} \cup \{\overline{q}, \widehat{q} \mid q \in Q \backslash \{q_f\}\}$, is the set of symbols present in the environment.
- $w_1 = ab$, $w_2 = \lambda$, $w_3 = Y_2Y_3$, $w_4 = Y_1e$, $w_5 = \lambda$.
- Cel 2 is the output cell.

- The rules R_Π are as follows
  - **I.** *Initialisation rules*:

    I.1 $(a,1)(Y_2,3) \to (a,5)(Y_2,0)$, $(b,1)(Y_3,3) \to (b,5)(Y_3,0)$, $(q_0^+,0)(e,4) \to (q_0^+,1)(e,5)$.
  - **II.** *For each increment instruction* (p, A_i^+, q, s), the rules below are introduced into R_Π. As in the previous proofs for $r \in \{q, s\}$, $r^\star$ stands for r^+ or r^-.

    II.1 $(p^+,1)(c_i,0) \to (p^+,3)(c_i,2)$;

    II.2 $(p^+,3)(Y_2,2) \to (p^+,4)(Y_2,0)$;

    II.3 $(p^+,4)(r^\star,0) \to (p^+,5)(r^\star,1)$.
  - **III.** *For each decrement instruction* (p, A_i^-, q, s), the following rules are added to R_Π; $q^\star$ and $s^\star$ are used with the meaning from the previous proofs:

    III.1 $(p^-,1)(c_i,2) \to (p^-,3)(c_i,0)$;

    III.2 $(p^-,3)(\widehat{q},0) \to (p^-,5)(\widehat{q},1)$;

    III.3 $(\widehat{q},1)(\overline{q},0) \to (\widehat{q},3)(\overline{q},2)$;

    III.4 $(\overline{q},2)(q^\star,0) \to (\overline{q},5)(q^\star,1)$, $(\widehat{q},3)(Y_2,2) \to (\widehat{q},5)(Y_2,0)$;

    III.5 $(p^-,1)(Y_2,2) \to (p^-,4)(Y_2,0)$;

    III.6 $(p^-,4)(s^\star,0) \to (p^-,5)(s^\star,1)$.
  - **IV.** *For the halting instruction corresponding to state* q_f, the following rules are added to R_Π:

    IV.1 $(q_f,1)(Y_2,2) \to (q_f,3)(Y_2,5)$;

    IV.2 $(q_f,3)(Y_1,1) \to (q_f,0)(Y_1,5)$.
  - **V.** *Auxiliary rules*:

    V.1 $(Y_1,4)(Y_2,0) \to (Y_1,3)(Y_2,2)$;

    V.2 $(Y_1,3)(Y_3,0) \to (Y_1,1)(Y_3,5)$;

    V.3 $(Y_1,1)(Y_3,5) \to (Y_1,4)(Y_3,0)$.

In the sequel we describe successful computations in Π corresponding to successful generation of non-negative numbers in M.

Initialisation step.
Before the simulation of M begins, starting from the initial configuration $(\lambda, ab, \lambda, Y_2Y_3, Y_1e, \lambda)$, we attain the configuration $(Y_2Y_3, q_0^+, \lambda, \lambda, Y_1, abe)$.

As in the proofs of the previous results we depict the configuration of the system at the start of the simulation of an instruction $(p, A_i^\star, q, s)$, where $A_i^\star$ is either A_i^+ or A_i^-. This is given by $(Y_2Y_3, p^\star, w, \lambda, Y_1, abex)$, where $p^\star$ is either p^+ or p^-, $w = c_1^{l_1} \dots c_k^{l_k}$, $l_i \geq 0$, $1 \leq i \leq k$, and x is a multiset over $Q^+ \cup Q^- \cup \{\overline{q}, \widehat{q} \mid q \in Q \backslash \{q_f\}\}$.

Simulation of an increment instruction (p, A_i^+, q, s).
We start the simulation of the increment instruction from the configuration $(Y_2Y_3, p^+, w, \lambda, Y,$

$abex$) (when $p = q_0$, $w = \lambda$). We use rules II.1 and V.1 in parallel obtaining $(Y_3, \lambda, wc_iY_2, p^+Y_1, \lambda, abex)$. A copy of c_i is brought into cell 2. This is followed by the use of II.2 and V.2 giving $(Y_2, Y_1, wc_i, \lambda, p^+, abexY_3)$, where symbol p^+ is moved into cell 4. This is followed by the use of II.3 and V.3 producing $(Y_2Y_3, r^\star, wc_i, \lambda, Y_1, abexp^+)$, by moving p^+ from cell 4 to cell 5 and $r^\star$ from the environment to cell 1. This copy of p^+ stays forever in cell 5; hence cell 5 can be considered as a "garbage collector" component. We are ready for the simulation of the next register machine instruction corresponding to state r, where $r \in \{q, s\}$.

Simulation of a decrement instruction (p, A_i^-, q, s).

As in the previous proofs, $q^\star \in \{q^+, q^-\}$ and $s^\star \in \{s^+, s^-\}$. To begin the simulation of a decrement instruction, we start with the configuration $(Y_2Y_3, p^-, w, \lambda, Y_1, abex)$. Two cases are considered.

Case 1: The content of register A_i is non-zero, i.e., $w = c_iw'$. We start by applying the rules III.1 and V.1 and obtaining $(Y_3, \lambda, w'Y_2, p^-Y_1, \lambda, abex)$. Next, using III.2 and V.2, we get $(\lambda, \widehat{q}Y_1, w'Y_2, \lambda, \lambda, Y_3abexp^-)$. Next, using III.3 and V.3, we have $(Y_3, \lambda, \overline{q}w'Y_2, \widehat{q}, Y_1, abexp^-)$. This is followed by III.4, obtaining $(Y_2Y_3, q^\star, w', \lambda, Y_1, abexp^-\overline{q}\widehat{q})$. The auxiliary symbols $\widehat{q}, \overline{q}$ are pushed into the "garbage collector" component (cell 5), and we are ready for the simulation of the next instruction corresponding to state q.

Case 2: Content of register A_i is zero; in this case w does not contain any c_i. In this case, we cannot use the rule III.1, since there is no c_i in cell 2. We therefore start with the rule of V.1, obtaining Y_1 in cell 3 and Y_2 in cell 2, and p^- staying in cell 1. So, the following configuration is obtained: $(Y_3, p^-, wY_2, Y_1, \lambda, abex)$. This is followed by rules III.5 and V.2, obtaining $(Y_2, Y_1, w, \lambda, p^-, abexY_3)$. Then, rules III.6 and V.3 are used which gives $(Y_2Y_3, s^\star, w, \lambda, Y_1, abexp^-)$. Y_2, Y_3 go back to the environment, and $s^\star$ comes to cell 1. We are now ready to simulate the instruction corresponding to state s.

Halting step.

To halt the computation, we move the auxiliary symbols Y_1, Y_2 into the "garbage collector" component, cell 5. When we obtain q_f in cell 1, the configuration is $(Y_2Y_3, q_f, w, \lambda, Y_1, abex)$. Applying rule V.1 to this configuration one gets $(Y_3, q_f, wY_2, Y_1, \lambda, abex)$. Then using IV.1 and V.2 in parallel we end up with $(\lambda, Y_1, w, q_f, \lambda, abexY_2Y_3)$. Now, rule IV.2 is used, and q_f goes to the environment, Y_1 goes to the "garbage collector" obtaining $(\lambda, \lambda, w, \lambda, \lambda, abexY_1Y_2Y_3)$. There are no more rules to apply, and the system halts. The content of membrane 2 is the output. □

3.4. GCPS with minimal interaction – presence_move

We recal that *presence_move* rules are such that $i = k, i \neq l, i \neq j, j \neq l$. We show below that 6 compartments are enough for simulating a register machine with such rules.

Theorem 4. $NGCPS(presence\_move)_6 = NRE$.

Proof. As in the proofs of the earlier results, we construct a minimal interaction GCPS Π with *presence rules*, having 6 cells which simulates a register machine, $M = (Q, R, q_0, q_f, P)$,

with k registers and m states ($Q = \{q_0, q_f, p_1, \ldots, p_{m-2}\}$), such that $N(\Pi) = N(M)$. It will be shown that for any computation in M there is a halting computation of the constructed GCPS, Π, and no unexpected computations are allowed in Π.

Given $Q' = Q \setminus \{q_f\}$, the sets Q'^+, Q'^- are defined as in Theorems 1, 2 and 3. We construct $\Pi = (O, E, w_1, \ldots, w_6, R_\Pi, 2)$ as it is described below.

- $O = Q'^+ \cup Q'^- \cup \{c_i \mid 1 \leq i \leq k\} \cup \{q_f, \dagger, \gamma, \delta, \eta_1, \eta_2, \chi\}$, is the alphabet of the system.
- $E = \{c_i \mid 1 \leq i \leq k\} \cup \{\dagger\}$, is the alphabet of the environment.
- $w_1 = q_0^+\chi$, $w_2 = w_3 = \lambda$, $w_4 = \eta_2$, $w_5 = \gamma$, $w_6 = \delta\eta_1 p_1^+ \ldots p_{m-2}^+ p_1^- \ldots p_{m-2}^- q_f$, are the initial multisets.
- Cell 2 is the output cell, storing the result of the computation.
- The rules R_Π are as follows:
  - **I.** *For each increment instruction* (p, A_i^+, q, s), if $r \in \{q, s\}$, the rules below are introduced into R_Π, where $r^\star$ stands for r^+ or r^-, when $r \in Q'$ or $r^\star = q_f$.
    I.1 $(p^+, 1)(c_i, 0) \to (p^+, 1)(c_i, 6)$;
    I.2 $(c_i, 6)(p^+, 1) \to (c_i, 6)(p^+, 3)$;
    I.3 $(c_i, 6)(\dagger, 0) \to (c_i, 6)(\dagger, 3)$;
    I.4 $(p^+, 3)(c_i, 6) \to (p^+, 3)(c_i, 2)$;
    I.5 $(\delta, 4)(p^+, 3) \to (\delta, 4)(p^+, 5)$;
    I.6 $(p^+, 5)(r^\star, 6) \to (p^+, 5)(r^\star, 1)$;
    I.7 $(\eta_2, 4)(p^+, 5) \to (\eta_2, 4)(p^+, 6)$.
  - **II.** *For each decrement instruction* (p, A_i^-, q, s), the following rules are added to R_Π; with $q^\star \in \{q^+, q^-\}$ or $q^\star = q_f$ and $s^\star \in \{s^+, s^-\}$ or $s^\star = q_f$:
    II.1 $(p^-, 1)(c_i, 2) \to (p^-, 1)(c_i, 3)$;
    II.2 $(c_i, 3)(p^-, 1) \to (c_i, 3)(p^-, 4)$;
    II.3 $(c_i, 3)(\dagger, 0) \to (c_i, 3)(\dagger, 6)$;
    II.4 $(p^-, 4)(c_i, 3) \to (p^-, 4)(c_i, 5)$;
    II.5 $(c_i, 5)(p^-, 4) \to (c_i, 5)(p^-, 3)$;
    II.6 $(\eta_1, 6)(c_i, 5) \to (\eta_1, 6)(c_i, 0)$, $(p^-, 3)(q^\star, 6) \to (p^-, 3)(q^\star, 1)$;
    II.7 $(q^\star, 1)(p^-, 3) \to (q^\star, 1)(p^-, 6)$;
    II.8 $(p^-, 3)(\dagger, 0) \to (p^-, 3)(\dagger, 6)$;
    II.9 $(\delta, 4)(p^-, 1) \to (\delta, 4)(p^-, 2)$;
    II.10 $(p^-, 2)(s^\star, 6) \to (p^-, 2)(s^\star, 1)$;
    II.11 $(\eta_2, 4)(p^-, 2) \to (\eta_2, 4)(p^-, 6)$.

– **III.** *For the halting instruction corresponding to state q_f*, the following rule is added to R_Π:

  III.1 $(q_f, 1)(\gamma, 5) \to (q_f, 1)(\gamma, 0)$.

– **IV.** *Auxiliary rules*:

  IV.1 $(\gamma, 5)(\delta, 6) \to (\gamma, 5)(\delta, 4)$;

  IV.2 $(\gamma, 5)(\delta, 4) \to (\gamma, 5)(\delta, 6)$;

  IV.3 $(\dagger, 3)(\chi, 1) \to (\dagger, 3)(\chi, 4)$, $(\dagger, 3)(\chi, 4) \to (\dagger, 3)(\chi, 1)$;

  IV.4 $(\dagger, 6)(\chi, 1) \to (\dagger, 6)(\chi, 4)$, $(\dagger, 6)(\chi, 4) \to (\dagger, 6)(\chi, 1)$.

The only objects available in infinitely many copies (in the environment) are the symbols c_i and $\dagger$; cell 6 has a copy of p^+, p^- for all $p \in Q' \setminus \{q_0^+, q_0^-\}$ as well as one copy of each of the auxiliary symbols δ, η_1. Symbols in $Q'^+ \cup Q'^-$ as well as $\{q_f, \gamma, \delta, \eta_1, \eta_2, \chi\}$ are available only in a single copy in Π.

As in the proofs of the previous results we start by describing the generic configuration of the system at the start of the simulation of an instruction $(p, A_i^\star, q, s)$, where $A_i^\star$ is either A_i^+ or A_i^-. This is given by $(\lambda, p^\star\chi, w, \lambda, \eta_2, \gamma, \delta\eta_1 x)$, where $p^\star \in \{p^+, p^-\}$, $w = c_1^{l_1} \dots c_k^{l_k}$, $l_i \geq 0$, $1 \leq i \leq k$, and x is a multiset over $Q'^+ \cup Q'^- \cup \{q_f\}$. After the simulation of an increment instruction one can have in cell 5 the multiset γr^+, $r \in Q'$, but r^+ will be moved to cell 6 immediately afterwards.

Simulation of an increment instruction (p, A_i^+, q, s).

The configuration of Π before the simulation of the increment instruction (p, A_i^+, q, s) is derived from the generic one described above; in this case the multiset in cell 1 is $p^+\chi$ and the rest remains unchanged. We start with rules I.1 and IV.1 obtaining $(\lambda, p^+\chi, w, \lambda, \eta_2\delta, \gamma, \eta_1 x c_i)$, i.e., a copy of c_i from the environment is moved into cell 6 and δ from cell 6 goes to cell 4, respectively. Also, we should not get more copies of c_i from the environment. For this, we shift p^+ from cell 1 to cell 3, in the presence of c_i in cell 6 and get δ back to cell 6, by using rules I.2 and IV.2, respectively. Consequently, $(\lambda, \chi, w, p^+, \eta_2, \gamma, \delta\eta_1 x c_i)$ is obtained. Note that δ keeps shuttling between cells 4 and 6 in the presence of γ in cell 5 (rules IV.1 and IV.2). Rules I.3 and IV.3 are to ensure that p^+ is shifted to cell 3; using I.3 and IV.3 will lead to an infinite computation. We need to move c_i to cell 2, where the contents of all the registers are stored (rule I.4). Rules I.4 and IV.1, used in parallel, produce $(\lambda, \chi, wc_i, p^+, \eta_2\delta, \gamma, \eta_1 x)$. Next, using I.5, p^+ is shifted to cell 5, in the presence of δ in cell 4. Using rule I.6, p^+ in cell 5 begets from cell 6, the symbol $r^\star \in \{q^+, q^-, s^+, s^-\}$ and in parallel, rule IV.2 will move δ from cell 4 to 6; $r^\star$, placed in cell 1, corresponds to the state of the new instruction, i.e., $(\lambda, r^\star\chi, wc_i, \lambda, \eta_2, \gamma p^+, \delta\eta_1 x)$. The symbol p^+, corresponding to the old instruction, is moved to cell 6; where symbols from Q'^+ are stored; this is done using rule I.7. Note that if I.7 is used before I.6, then we will not beget the symbol $r^\star$ corresponding to the state of the new instruction, in cell 1; in this case, δ will keep on shuttling between cells 6 and 4, using rules IV.1 and IV.2, respectively, generating an infinite computation. Note also that I.7 happens

in parallel with the beginning of the simulation of the next instruction; δ is in cell 6 at the start of a new simulation.

Simulation of a decrement instruction (p, A_i^-, q, s).

At the start of a decrement instruction (p, A_i^-, q, s), we have the configuration derived from the generic one, whereby cell 1 contains the multiset $p^-\chi$ and the rest stays unchanged. We consider two cases here:

Case 1: The content of register A_i is non-zero, i.e., $w = c_i w'$. In this case, we start with II.1 and IV.1. A copy of c_i is moved from cell 2 to cell 3 and δ moves from cell 6 to cell 3, respectively. The resultant configuration is $(\lambda, p^-, w', c_i, \eta_2\delta, \gamma, \eta_1 x)$. To prevent the removal of more than one copy of c_i from cell 2, we use rule II.2, which moves p^- to cell 4; in parallel δ is moved from cell 4 to 6 (IV.2). If rule II.2 is not executed (assume p^- stays in cell 1, and removes another copy of c_i, or uses rule II.9) then rule II.3 is used, and the symbol † is brought inside Π, cell 6, resulting in an infinite computation by using IV.4. This is followed by rule II.4, and the copy of c_i is shifted from cell 3 to cell 5; the rule IV.1 is used in parallel, resulting the configuration $(\lambda, \lambda, w', \lambda, \eta_2 p^-\delta, \gamma c_i, \eta_1 x)$. Rule II.5 moves p^- to cell 3 and then rules II.6 release c_i from cell 5 into the environment, and bring $q^\star$, the symbol corresponding to the state of the next instruction, from cell 6 to 1. The configuration then is $(\lambda, q^\star, w', p^-, \eta_2\delta, \gamma, \eta_1 x')$, where x' is x without $q^\star$. This is followed by II.7, which moves p^- to cell 6; note that if $q^\star$ gets involved in any rule other than II.7, an infinite computation is obtained (rules IV.4) using rule II.8. If II.7 is used, then we obtain $(\lambda, q^\star, w', \lambda, \eta_2, \gamma, \delta\eta_1 x)$, and we are ready for the next simulation.

Case 2: The content of register A_i is zero, i.e., there is no c_i in cell 2. In this case we cannot use the rule II.1. Then we have p^- in cell 1, and δ in cell 4. We then use rule II.9 and p^- is moved to cell 2. From cell 2, p^- obtains $s^\star$ in cell 1 using rule II.10, and δ comes to cell 6. This $s^\star$ can start the next simulation, while in parallel, p^- is moved to cell 6 using II.11. It must be noted that if II.11 is executed before II.10, we would obtain an infinite computation due to δ moving between cells 4 and 6.

Halting step.

To halt the computation, we remove γ from cell 5, once we obtain q_f in cell 1. The output of Π is given by the contents of cell 2. □

4. Applications

In this section we will present some problems that are modelled with GCPS. They consist of a producer/consumer problem, some examples of workflow patterns, two broadcasting problems and comparative operations.

For these applications we use a GCPS model given by

$$\Pi = (O, E, w_1, \ldots, w_n, R_\Pi)$$

where its elements will be defined in each of the cases below. Please note that we do not need output cells as the result is not captured in a distinguished cell.

4.1. Producer/consumer models

The **producer/consumer** paradigm consists of a system with two processes, a producer and a consumer, which synchronise through a buffer of one item. This has been modelled in different formalisms, including Petri nets [22] and membrane systems [2]. The last model uses generalised membrane systems with rewriting rules taking objects from various compartments and placing the results in other compartments. A slightly different version of this problem is investigated in [19, 20] by using numerical P systems. In the sequel we will use GCPS models with the same number of rules as those in [2].

The GCPS, $\Pi_{p/c}$ of degree 6 will consists of cells identified by P ("ready to produce") and D ("ready to deliver") – for the producer, R ("ready to remove") and C ("ready to consume") – the consumer, and F ("filled") and E ("empty") – the buffer. The alphabet $O = \{t\}$ (where t stands for a token available in certain cells). We consider that the environment will also contain t. The initial multisets are $w_P = w_E = w_C = t$ and the others are empty. The rule set $R_{p/c}$ contains the following rules:
$r_1 : (t, P)(t, 0) \to (t, D)(t, 0)$;
$r_2 : (t, C)(t, 0) \to (t, R)(t, 0)$;
$r_3 : (t, E)(t, D) \to (t, F)(t, P)$;
$r_4 : (t, F)(t, R) \to (t, E)(t, C)$.

The producer starts producing by moving the token t from P to D (rule r_1) and the consumer is preparing for consuming what the producer is sending, by moving t from C to R (rule r_2). At this moment the buffer moves t from E to F, signalling that the buffer is full, and the producer is returning it from D to P (rule r_3). Then the consumer consumes the object sent by the producer and stored by the buffer by using rule r_4. The process described by the GCPS $\Pi_{p/c}$ is very similar to the process described by the Petri nets model [22] and by the membrane systems [2].

4.2. Workflow patterns

Workflow patterns represent the basis of building various workflow processes and are modelled with different formalisms, the most utilised being Petri nets [25, 24]. In the sequel we present GCPS models for the sequence, parallel split, synchronisation and mutual exclusion workflow patterns.

A *Sequence* pattern is used to model consecutive steps in a workflow process. The control pattern cannot be started again until it has completed the execution of the preceding thread of control. We represent this pattern with a GCPS consisting of five cells, in, aux_1, aux_2, p and out; two symbols c, X and two rules:
$r_1 : (c, in)(X, aux_1) \to (c, p)(X, aux_2)$;
$r_2 : (c, p)(X, aux_2) \to (c, out)(X, aux_1)$.

This model describes precisely the behaviour of a sequence pattern where a new pattern is executed only after processing the last rule.

A *Parallel split* workflow pattern describes the situation of separating a workflow into two workflows running in parallel, each of them ending in different states. The GCPS uses seven

cells denoted in, p_1, p_2, aux_1, aux_2, out_1, out_2; two symbols c, X and five rules:
$r_1 : (c, in)(c, in) \to (c, p_1)(c, p_2)$;
$r_{1+i} : (c, p_i)(X, aux_i) \to (c, out_i)(X, aux_i)$; $1 \leq i \leq 2$;
$r_{3+i} : (c, p_i)(X, aux_i) \to (c, p_i)(X, aux_i)$, $1 \leq i \leq 2$.

After $c's$ are distributed, each one for a different workflow, then the two workflows are executed in parallel, in any order. The rule r_1 is a *split* operation, r_2, r_3 are *presence_move rules* and r_4, r_5 are *identity rules* (the same left and right side).

The *Synchronisation* pattern describes the problem of a parallel merge. Two parallel workflows are synchronised such that their final results are processed at the same time. A GCPS for this problem consists of eight cells, $in_1, in_2, p_1, p_2, aux_1, aux_2, aux, out$; two symbols c, X and the rules:
$r_i : (c, in_i)(X, aux_i) \to (c, p_i)(X, aux_i)$; $1 \leq i \leq 2$;
$r_{2+i} : (c, in_i)(X, aux_i) \to (c, in_i)(X, aux_i)$; $1 \leq i \leq 2$;
$r_5 : (c, p_1)(c, p_2) \to (c, out)(c, aux)$.

The two c's that start from in_1 and in_2 arrive at p_1 and p_2, respectively; they might not proceed at the same time, due to jkjr_3 and r_4 that delay there move. The synchronisation is achieved through rule r_5.

The *Mutual exclusion* problem with a common resource consists of two processes A and B which are executed in parallel, but only one is executed at a given time when a shared resource is requested [22]. The GCPS consists of the following cells: p_1, p_2, p_3 and a for A and q_1, q_2, q_3 and b for B and a_1, b_1, s for the common resource; symbols c, X, x are used. The rules are:
for the process A:
$r_1 : (c, p_1)(X, a) \to (c, p_2)(X, a)$;
$r_2 : (c, p_2)(x, s) \to (c, p_3)(x, a_1)$;
$r_3 : (c, p_3)(x, a_1) \to (c, p_1)(x, s)$;
for the process B:
$r_4 : (c, q_1)(X, b) \to (c, q_2)(X, b)$;
$r_5 : (c, q_2)(x, s) \to (c, q_3)(x, b_1)$;
$r_6 : (c, q_3)(x, b_1) \to (c, q_1)(x, s)$.

The processes A and B will start executing when a c enters p_1 and q_1, respectively. So, A and B can be executed in any order or at the same time and c moves to p_2 and q_2, respectively, by using r_1 and r_4, respectively, which are presence_move rules. Now they are ready to enter the critical region, i.e., an x available from s will trigger the execution of either r_2 or r_5, but not both (parallel_shift rules); this means that only one process enters the critical region. Once a process finishes then x is restored back to s and c moves to the initial state for a new iteration.

4.3. Broadcasting

Broadcasting a signal in a network has been investigated for various types of P systems

using tree-like structures [4, 5] or hyperdags [17]. The hyperdag structure has been used in the case of bounded fanout broadcast problem [10]. In these cases the network is modelled as a P system structure with nodes being cells of the P system. We will investigate this problem by using a GCPS model. Our model will be implicitly a bounded fanout broadcast.

The brodcasting problem considered here can be formulated as sending a signal in a tree-like network from the root to every other node. We also present the case when an acknowledgement is received from each child of a node.

The tree will be defined by using a graph notation where the set of nodes is V and the edges are E. For a set V with m elements this will be $\{1, \ldots, m\}$, with 1 being the root. For any edge (i, j), i denotes the parent and j one of its children. In order to use a GCPS model we have to add some auxilliary cells and links to "wire" them with those of the tree-like structure. This model is a $parallel\_shift$ GCPS.

The GCPS is no longer relying on an underlying tree structure, but we will use the graph terminology, like parent-child, when cells corresponding to the nodes of the initial tree structure are utilised.

Broadcasting. The GCPS Π_{p_s} of degree n has, apart from the cells labelled i, for $i \in V$, where 1 stands for the root, the additional cells labelled $[i, j]$ for $(i, j) \in E$ and x, a new symbol not in V. Hence, $n = 2m$, where m is the number of nodes of the tree. Π_{p_s} consists of

- $O = \{s, t\}$, where s is the signal to be broadcast and t is used in selecting a child where s will be sent to; the system will contain one s located initially in the root, and a t for each non-leaf node, as explained below; the environment is considered to contain copies of s;
- $w_1 = s, w_{[i,j_1]} = t$, j_1 indicates the first of the p children, $j_1, \ldots, j_p$ of i, $(i, j_k) \in E$, $1 \leq k \leq p$; all the other initial multisets are λ;
- the rule set R_{p_s} contains for every non-leaf node i, such that $(i, j_k) \in E$, $1 \leq k \leq p$, the rules
  - $p = 1$
    - $r_{(i,j_1)} : (s, i)(t, [i, j_1]) \to (s, j_1)(t, x)$;
  - $p > 1$
    - $r_{(i,j_1),1} : (s, i)(t, [i, j_1]) \to (s, j_1)(t, [i, j_2])$;
    - $r_{(i,j_k),2} : (s, 0)(t, [i, j_k]) \to (s, j_k)(t, \alpha)$, where $\alpha = [i, j_{k+1}]$, $1 < k < p$ and $\alpha = x$, when $k = p$.

When signal s is in cell i as $[i, j_1]$ has a t, then we have the following situations: when $p = 1$, s goes to cell j_1 – rule $r_{(i,j_1)}$ is enabled and t goes to x; if $p > 1$ then s goes to j_1, according to rule $r_{(i,j_1),1}$ which is enabled in this case, and t moves to $[i, j_2]$. Signal s arrives in the other cells j_k, $1 < k \leq p$, from the environment (cell 0) by using rules $r_{(i,j_k),2}$, as t

becomes available in $[i, j_k]$. So, it takes p steps for the signal to get to all p children of i. The process will iterate until s arrives in leaf compartments.

The number of rules of the GCPS R_{p_s} is equal to $m-1$, the number of edges in the tree. The maximum number of steps of the computation is also equal to the number of edges, $m-1$; this maximum is attained when every non-leaf node has precisely one child. Typically, the number of steps required is less than $m-1$ since, in maximal parallelism mode, rules at different levels of the tree are applied simultaneously. We have an upper bound on the number of steps given by the height of the tree plus the maximum degree of a node.

Broadcasting with acknowledgement. The GCPS $\Pi_{p\_s,ac}$ is similar to Π_{p_s} introduced above, but has some more cells and rules necessary to handle the acknowledgement process. $\Pi_{p\_s,ac}$ has cells labelled i, $i \in V$, x – as above, and $[i, j, 1]$ and $[i, j, 2]$ for each $(i, j) \in E$. Hence, the total number of cells is $n = 3m - 1$. $\Pi_{p\_s,ac}$ consists of

- $O = \{s, a, t\}$, where s and t are as in Π_{p_s} and a is the acknowledgement symbol; the system will contain one s located initially in the root, a t for each non-leaf node and an a for each node of the tree different from the root, as explained below; the environment is considered to contain copies of s;
- the initial multisets are $w_1 = s$, $w_k = a$, $1 < k \leq m$, $w_x = \lambda$; $w_{[i,j_1,1]} = t$, $w_{[i,j_k,1]} = \lambda$, $1 < k \leq p$ and $w_{[i,j_k,2]} = \lambda$, $1 \leq k \leq p$, for all $(i, j_k) \in E$, $1 \leq k \leq p$;
- the rule set $R_{p\_s,ac}$ contains for every non-leaf node i, such that $(i, j_k) \in E$, $1 \leq k \leq p$, the rules
  - $p = 1$
    - $r_{(i,j_1,1)} : (s, i)(t, [i, j_1, 1]) \to (s, j_1)(t, [i, j_1, 2])$;
    - $r_{(i,j_1,2)} : (a, j_1)(t, [i, j_1, 2]) \to (a, i)(t, x)$;
  - $p > 1$
    - $r_{(i,j_1,1),1} : (s, i)(t, [i, j_1, 1]) \to (s, j_1)(t, [i, j_1, 2])$;
    - $r_{(i,j_1,2),1} : (a, j_1)(t, [i, j_1, 2]) \to (a, i)(t, [i, j_2, 1])$;
    - $r_{(i,j_k,1),2} : (s, 0)(t, [i, j_k, 1]) \to (s, j_k)(t, [i, j_k, 2])$;
    - $r_{(i,j_k,2),2} : (a, j_k)(t, [i, j_k, 2]) \to (a, i)(t, \alpha)$, where $\alpha = [i, j_{k+1}, 1]$, $1 < k < p$ and $\alpha = x$, when $k = p$.

The behaviour of $\Pi_{p\_s,ac}$ is similar to Π_{p_s}, but additionally an acknowledgement is sent from each child cell to its parent. For this reason an additional cell appears associated with each $(i, j) \in E$. In one step the signal is sent from the parent to the first child (either of the rules $r_{(i,j_1,1)}$ or $r_{(i,j_1,1),1}$), or from the environment to each of the other children when more than a child exists (rule $r_{(i,j_k,1),2}$). In the second step an acknowledgement is received by the parent from each child (one of the rules $r_{(i,j_1,2)}$, $r_{(i,j_1,2),1}$ or $r_{(i,j_k,2),2}$, depending on the number of children).

The number of rules of GCPS $\Pi_{p\_s,ac}$ is $2(m-1)$, twice the number of edges. The maximum number of steps of the computation is also equal to $2(m-1)$. This maximum is attained when every non-leaf node has one child, otherwise, due to the maximal parallelism mode, less steps are needed. We have an upper bound on the number of steps given by the height of the tree plus the maximum degree of a node.

The model presented is bounded fanout broadcast [10], with only one single communication per each step.

4.4. Comparative operations

Comparative operations between two integers have been implemented with a great variety of P systems, and have been used to develop sorting methods with such systems.

In the context of bringing together membrane computing and the human genome project, Professor Marcus in [15] says: "The rules identifying the genes are a mixture of chemistry, biology and combinatorial and **comparative** operations." This emphasises the importance of comparative operations, which are in turn key operations for sorting methods.

We present in the sequel a comparator implemented with minimal interaction rules. Unlike in the previous applications, two different types of rules are used.

We codify two integers x_1 and x_2 with the number of apparitions of one symbol, a, in two different cells, labelled respectively 1 and 2. We have $w_1 = a^{x_1}$ and $w_2 = a^{x_2}$, all the other cells are empty. We collect the result of the comparison in other two cells, labelled $1'$ and $2'$. In $1'$ we collect $min\{x_1, x_2\}$, and in $2'$ we collect $max\{x_1, x_2\}$, both codified as number of occurrences of symbol a.

We have the following set of rules:
$(a,1)(a,2) \rightarrow (a,1')(a,2')$;
$(a,1')(a,1) \rightarrow (a,1')(a,2')$;
$(a,1')(a,2) \rightarrow (a,1')(a,2')$.

The first rule is a parallel_shift rule, and it places $min\{x_1, x_2\}$ occurrences of a in membranes $1'$ and $2'$. The next two rules are presence_move rules. Their role is to place $max\{x_1, x_2\}$ occurrences of a in cell $2'$. They do so by moving remaining objects a from cell 1 or from cell 2 into $2'$.

This comparator works very similar to the comparator devised in [3] based on symport/antiport rules, with priorities. The comparator in [3] achieves the result in one step, but it uses, apart from communication rules, priorities in triggering the rules.

5. Relations between GCPS models and P colonies

GCPSs and P colonies demonstrate formal and functional similarities. Both of them are networks of cells where the cells and their shared environment are represented by multisets of objects and the cells may interact with the environment via exchange of objects. However, differences between the two models can also be noticed: while the number of objects in each cell of a P colony is constant during functioning, the cells of a GCPS may contain an arbitrarily

large number of objects in the course of computation. Furthermore, when a program of a cell of a P colony is used, then a rule is applied to each object in the cell, while in the case of GCPSs a cell may contain objects which are not affected by any of the rules applied during the transition.

P colonies, as generalised communicating P systems, are computationally complete computing devices [7]; even those of capacity 1 [6]. In the following we show how these systems can be simulated with GCPSs. For easier reading, we slightly deviate from the standard labeling of cells.

Theorem 5. *For every P colony* $\Pi = (O, e, F, C_1, \ldots, C_n)$, $n \geq 1$, *of capacity* 1, *we can construct a GCPS* $\Pi' = (O, E, w_{0_E}, w_{0_F}, w_1, \ldots, w_n, R_{\Pi'}, 0_F)$, *such that* $N(\Pi) = N(\Pi')$ *holds.*

Proof. Let us consider $\Pi = (O, e, F, C_1, \ldots, C_n)$, $n \geq 1$. Components of $\Pi' = (O, E, w_{0_E}, w_{0_F}, w_1, \ldots, w_n, R_{\Pi'}, 0_F)$, simulating Π are defined as follows. Let $O = E$, $w_{0_E} = w_{0_F} = \lambda$ and let $w_i = e$ for every i, $1 \leq i \leq n$.

The rule set $R_{\Pi'}$ of Π' is given as follows:

1. For any program $p : a \to b$ in P_i, $1 \leq i \leq n$, where $a, b \in O$, $R_{\Pi'}$ has a rule $r : (a, i)(b, 0) \to (b, i)(a, 0)$.
2. For any program of the form $p : a \leftrightarrow b$ in P_i, $1 \leq i \leq n$, where $a, b \in (O \setminus (F \cup \{e\}))$, $R_{\Pi'}$ has a rule $r : (a, i)(b, 0_E) \to (b, i)(a, 0_E)$.
3. For any program of the form $p : a \leftrightarrow e$ in P_i, $1 \leq i \leq n$, where $a \in (O \setminus (F \cup \{e\}))$, $R_{\Pi'}$ has a rule $r : (a, i)(e, 0) \to (e, i)(a, 0_E)$.
4. For any program of the form $p : e \leftrightarrow a$ in P_i, $1 \leq i \leq n$, where $a \in (O \setminus (F \cup \{e\}))$, $R_{\Pi'}$ has a rule $r : (e, i)(a, 0_E) \to (a, i)(e, 0)$.
5. For any program of the form $p : f \leftrightarrow a$ in P_i, $1 \leq i \leq n$, where $a \in (O \setminus (F \cup \{e\}))$, $f \in F$, $R_{\Pi'}$ has a rule $r : (f, i)(a, 0_E) \to (a, i)(f, 0_F)$.
6. For any program of the form $p : a \leftrightarrow f$ in P_i, $1 \leq i \leq n$, where $a \in (O \setminus (F \cup \{e\}))$, $f \in F$, $R_{\Pi'}$ has a rule $r : (a, i)(f, 0_F) \to (f, i)(a, 0_E)$.
7. For any program of the form $p : e \leftrightarrow f$ in P_i, $1 \leq i \leq n$, where $f \in F$, $R_{\Pi'}$ has a rule $r : (e, i)(f, 0_F) \to (f, i)(e, 0)$.
8. For any program of the form $p : f \leftrightarrow e$ in P_i, $1 \leq i \leq n$, where $f \in F$, $R_{\Pi'}$ has a rule $r : (f, i)(e, 0) \to (e, i)(f, 0_F)$.

$R_{\Pi'}$ consists of the previously given rules.

Next we show that $N(\Pi) = N(\Pi')$. Our idea is based on the following considerations. Firstly, we consider $E = O$, thus ensuring the availability of object b for simulating an evolution $a \to b$ of Π at any step of the computation in Π'. Secondly, since a cell of Π can import an object $a \neq e$ from the environment only if a is present there, Π' should have at least one cell for storing the objects different from e which are present in the environment of Π in the current computation step. Thus, Π' needs to have $n + 2$ cells, out of which n cells

simulate the cells of Π, cell 0_E stores the objects which are different from elements of $F \cup \{e\}$ and are present at that step of the computation of Π in the environment. Cell 0_F contains as many copies of objects $f \in F$ as are present in the environment at the current computation step and it has no other elements. Cell 0 denotes the environment of GCPS Π'. The given rules of Π' correspond to programs of Π and the initial configurations of Π and Π' correspond to each other, by the definition of Π'.

We show that if configuration $(y_E, y_1, \ldots, y_n)$ can be obtained from configuration $(x_E, x_1, \ldots, x_n)$ by a transition in Π, then there exist configurations $(\lambda, y_{0_E}, y_{0_F}, y_1, \ldots, y_n)$ and $(\lambda, x_{0_E}, x_{0_F}, x_1, \ldots, x_n)$ where $y_E = y_{0_E} y_{0_F}$, $x_E = x_{0_E} x_{0_F}$ such that $(\lambda, y_{0_E}, y_{0_F}, y_1, \ldots, y_n)$ can directly be obtained from configuration $(\lambda, x_{0_E}, x_{0_F}, x_1, \ldots, x_n)$ in Π'. If the transition from $(x_E, x_1, \ldots, x_n)$ to $(y_E, y_1, \ldots, y_n)$ is performed in Π, then for every object x_j, $1 \leq j \leq n$, one of the following cases holds: (1) no program (rule) is applicable to x_j; (2) y_j is obtained from x_j by evolution; (3) y_j is obtained from x_j by communication. Then, by definition of $R_{\Pi'}$, the corresponding rules can be applied to $(\lambda, x_{0_E}, x_{0_F}, x_1, \ldots, x_n)$ resulting in configuration $(\lambda, y_{0_E}, y_{0_F}, y_1, \ldots, y_n)$. The reverse statement can be proved by using analogous reasoning. We obtain that all computations in Π correspond to computations in Π' and reversely, which implies that $N(\Pi) = N(\Pi')$.

6. Conclusions

In this paper we have further improved the completeness results for GCPS models with minimal interaction, which was formulated as an open problem in [26]. Finally, some examples illustrating the modelling capabilities of GCPS are presented. We briefly investigated relations between GCPSs and P colonies; in the future, we plan to investigate this relation for P colonies of capacity k, $k > 1$, for GCPSs with minimal interaction, with dynamically changing environment, and with different functioning modes. We also plan to compare the synchronisation mechanisms in these models. There are some open problems related to the minimal number of cells for the GCPS models with minimal interactions as well as the generic ones that require further investigations.

Acknowledgements. MG and FI were supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-II-ID-PCE-2011-3-0688, CSVE by grant No. 120558 of the National Research, Development, and Innovation Office, Hungary. The authors would like to thank Acad. Gh. Păun for reading a first draft of this paper and making valuable comments. Also, they would like to thank the anonymous reviewers for carefully reading a previous version of the paper and for their valuable comments made and improvements suggested.

References

[1] A. Balaskó, E. Csuhaj-Varjú and Gy. Vaszil, Dynamically changing environment for generalized communicating P systems. In G. Rozenberg et al., eds., *16th International*

Conference on Membrane Computing, 2015, Valencia, Spain, Lecture Notes in Computer Science, **9504**, 2015, 92 – 105.

[2] F. Bernardini, M. Gheorghe, M. Mergenstern and S. Verlan, Producer/consumer in membrane systems and Petri nets. In S. B. Cooper at al., eds., *Computation and Logic in the Real World - 3rd Conference on Computability in Europe, CiE 2007, Siena, Italy, 2007,* Lecture Notes in Computer Science, **4497**, 2007, 43 – 52.

[3] R. Ceterchi, C. Martín-Vide, P Systems with communication for static sorting. In M. Cavaliere et al., eds., *Pre-Proc. 1st Brainstorming Week on Membrane Computing*, Technical Report, no 26, Rovira i Virgili Univ., Tarragona, 2003, 101 – 117.

[4] G. Ciobanu, Distributed algorithms over communicating membrane systems. *Biosystems*, **70**, 2003, 123 – 133.

[5] G. Ciobanu, R. Desai and A. Kumar, Membrane systems and distributed computing. In Gh. Păun et al., eds., *3rd International Workshop on Membrane Computing, 2002, Curtea de Argeş, Romania*, Lecture Notes in Computer Sience, **2597**, 2003, 187 – 202.

[6] L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová and Gy. Vaszil, Variants of P colonies with very simple cell structure. *Int. J. of Computers, Communications & Control*, **IV(3)**, 2009, 224 – 233.

[7] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun and Gy. Vaszil, Computing with cells in environment: P colonies. *Multiple-Valued Logic and Soft Computing*, **12(3-4)**, 2006, 201 – 215.

[8] E. Csuhaj-Varjú, Gy. Vaszil and S. Verlan, On generalized communicating P systems with one symbol. In M. Gheorghe et al., eds., *11th International Conference on Membrane Computing, 2010, Jena, Germany*, Lecture Notes in Computer Science, **6501**, 2010, 160 – 174.

[9] E. Csuhaj-Varjú and S. Verlan, On generalized communicating P systems with minimal interaction rules. *Theoretical Computer Science*, **412**, 2011, 124 – 135.

[10] M.J. Dinneen and Y.-B. Kim, Using membrane systems to solve the bounded fanout broadcast problem. *Romanian Journal of Information Science and Technology*, (accepted).

[11] A. Kelemenová, P colonies. In [21], 2010, 584 – 594.

[12] S.N. Krishna, M. Gheorghe and C. Dragomir, Some classes of generalised communicating P systems and simple kernel P systems. Technical report, CS-12-03, University of Sheffield, 2013; available at http://staffwww.dcs.shef.ac.uk/people/M.Gheorghe/research/paperlist.html

[13] S.N. Krishna, M. Gheorghe and C. Dragomir, Some classes of generalised communicating P systems and simple kernel P systems. In P. Bonizzoni et al., eds., *The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, 2013,* Lecture Notes in Computer Science, **7921**, 2013, 284 – 293.

[14] S. Marcus, Membranes versus DNA. *Fundamenta Informaticae*, **49**, 1–3, 2002, 223 – 227.

[15] S. Marcus, Bridging P systems and genomics: a preliminary approach. In Gh. Păun et al., eds., *3rd International Workshop on Membrane Computing, 2002, Curtea de Argeş, Romania*, Lecture Notes in Computer Sience, **2597**, 2003, 371 – 376.

[16] M. Minsky, *Computation: finite and infinite machines*, Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[17] R. Nicolescu, M.J. Dinneen and Y.-B. Kim, Discovering the membrane topology of hyperdag P systems. In Gh. Păun et al, eds., *10th International Workshop on Membrane Computing, 2009, Curtea de Argeş, Romania*, Lecture Notes in Computer Sience, **5957**, 2010, 410 – 435.

[18] A. Păun and Gh. Păun, The power of communication: P systems with symport/antiport. *New Generation Computing,* **20**, 2002, 295 – 306.

[19] Gh. Păun and R. Păun, Membrane computing as a framework for modeling economic processes. In Proc. SYNASC 05, Timisoara, Romania, IEEE Press, 2005, 11 - 18.

[20] Gh. Păun and R. Păun, Membrane Computing and Economics. In [21], 632 – 644.

[21] Gh. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford handbook of membrane computing*, Oxford University Press, 2010.

[22] W. Reisig, *Elements of distributed algorithms: modeling and analysis with Petri nets* Springer-Verlag, 1998.

[23] W. Reisig, *Petri nets: an introduction,* Springer-Verlag, 1985.

[24] N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst and N. Mulyar, Workflow control-flow patterns: a revised view. *BPM Center Report BPM-06-22*, 2006.

[25] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, Workflow patterns. *Distributed and Parallel Databases*, **14**, 5, 2003, 5 – 51.

[26] S. Verlan, F. Bernardini, M. Gheorghe and M. Margenstern, Generalized communicating P systems. *Theoretical Computer Science*, **404**, 1–2, 2008, 170 – 184.