



**HAL**  
open science

## Tissue P Systems with Vesicles of Multisets

Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan

► **To cite this version:**

Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan. Tissue P Systems with Vesicles of Multisets. *International Journal of Foundations of Computer Science*, 2022, 33 (3-4), pp.179–202. 10.1142/S0129054122410015 . hal-03672466

**HAL Id: hal-03672466**

**<https://hal.science/hal-03672466>**

Submitted on 3 Jan 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

International Journal of Foundations of Computer Science  
© World Scientific Publishing Company


## Tissue P Systems with Vesicles of Multisets\*

Artiom Alhazov 

*Vladimir Andrunachievici Institute of Mathematics and Computer Science  
Academiei 5, Chişinău, MD-2028, Moldova  
artiom@math.md*

Rudolf Freund 

*Faculty of Informatics, TU Wien  
Favoritenstraße 9–11, 1040 Vienna, Austria  
rudi@emcc.at*

Sergiu Ivanov 

*IBISC, Université Évry, Paris-Saclay  
23, boulevard de France 91034 Évry, France  
sergiu.ivanov@ibisc.univ-evry.fr*

Sergey Verlan 

*Univ. Paris Est Creteil, LACL  
94010, Creteil, France  
verlan@u-pec.fr*

We consider tissue P systems working on vesicles of multisets with the very simple operations of insertion, deletion, and substitution of single objects. With the whole multiset being enclosed in a vesicle, sending it to a target cell can be indicated in those simple rules working on the multiset. As derivation modes we consider the sequential mode, where exactly one rule is applied in a derivation step, and the set maximal mode, where in each derivation step a non-extendable set of rules is applied. With the set maximal mode, computational completeness can already be obtained with tissue P systems having a tree structure, whereas tissue P systems even with an arbitrary communication structure are not computationally complete when working in the sequential mode. Adding polarizations – only the three polarizations  $-1$ ,  $0$ ,  $1$  are sufficient – allows for obtaining computational completeness even for tissue P systems working in the sequential mode.

### 1. Introduction

Membrane systems were introduced at the end of last century by Gheorghe Păun, e.g., see [6] and [16], motivated by the biological interaction of molecules between cells and their surrounding environment. In the basic model, the membranes are organized in a hierarchical membrane structure (i.e., the connection structure be-

\*This paper is an extended and improved version of the paper presented at AFL 2017 in Debrecen.

2 Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan

tween the compartments/regions within the membranes being representable as a tree), and the multisets of objects in the membrane regions evolve in a maximally parallel way, with the resulting objects also being able to pass through the surrounding membrane to the parent membrane region or to enter an inner membrane. Since then, a lot of variants of membrane systems, for obvious reasons mostly called *P systems*, have been investigated, most of them being computationally complete, i.e., being able to simulate the computations of register machines. If an arbitrary graph is used as the connection structure between the cells/membranes, the systems are called *tissue P systems*, see [13].

Instead of multisets of plain symbols coming from a finite alphabet, P systems quite often operate on more complex objects (e.g., strings, arrays), too. A comprehensive overview of different flavors of (tissue) P systems and their expressive power is given in the handbook which appeared in 2010, see [17]. For a view on the state of the art of the domain, we refer the reader to the P systems website [20], as well as to the Bulletin series of the International Membrane Computing Society [19].

Very simple biologically motivated operations on strings are the so-called *point mutations*, i.e., *insertion*, *deletion*, and *substitution*, which mean inserting or deleting one symbol in a string or replacing one symbol by another one. For example, graph-controlled insertion-deletion systems have been investigated in [8], and P systems using these operations at the left or right end of string objects were introduced in [12], where also a short history of using these point mutations in formal language theory can be found.

When dealing with multisets of objects, the close relation of insertion and deletion with the increment and decrement instructions in register machines looks rather obvious. The power of changing states in connection with the increment and decrement instructions then has to be mimicked by moving the whole multiset representing the configuration of a register machine from one cell to another one in the corresponding tissue system. Yet usually moving the whole multiset of objects in a cell to another one, besides maximal parallelism, requires *target agreement* between all applied rules, i.e., that all results are moved to the same target cell, e.g., see [10].

In this paper we choose a different approach to guarantee that the whole multiset is moved even if only some point mutations are applied – the multiset is enclosed in a vesicle, and this vesicle is moved from one cell to another one as a whole, no matter how many rules have been applied. One constraint, of course, is that a common target has been selected by all rules to be applied; in the sequential derivation mode, this is no restriction at all, whereas in the set maximally parallel derivation mode this means that the set of rules to be applied must be non-extendable, but only with respect to all rules having to indicate the same target cell. We also consider the variants of the set maximally parallel derivation mode where the maximal number of rules has to be applied or where the maximal number of objects has to be affected by the applied rules. As we will show, with all the variants of the set

maximally parallel derivation mode computational completeness can be obtained, whereas with the sequential mode we achieve a characterization of the family of sets of (vectors of) natural numbers defined by partially blind register machines, which itself corresponds with the family of sets of (vectors of) natural numbers obtained as number (Parikh) sets of string languages generated by matrix grammars without appearance checking.

The idea of using vesicles of multisets has already been used in variants of P systems using the operations *drip* and *mate*, corresponding with the operations *cut* and *paste* well-known from the area of DNA computing, see [9]. Yet in that case, always two vesicles (one of them possibly an axiom available in an unbounded number) have to interact. In this paper, the rules (bounded in number) are always applied to the same vesicle.

The *point mutations*, i.e., *insertion*, *deletion*, and *substitution*, well-known from biology as operations on DNA, have also widely been used in the variants of *networks of evolutionary processors (NEPs)*, which consist of cells (processors) each of them allowing for specific operations on strings. *Networks of Evolutionary Processors (NEPs)* were introduced in [5] as a model of string processing devices distributed over a graph, with the processors carrying out these point mutations. Computations in such a network consist of alternatingly performing two steps – an *evolution step* where in each cell all possible operations on all strings currently present in the cell are performed, and a *communication step* in which strings are sent from one cell to another cell provided specific conditions are fulfilled. Examples of such conditions are (output and input) filters which have to be passed, and these (output and input) filters can be specific types of regular languages or permitting and forbidden context conditions. The set of strings obtained as results of computations by the NEP is defined as the set of objects which appear in some distinguished node in the course of a computation.

In *hybrid networks of evolutionary processors (HNEPs)*, each language processor performs only one of these operations at a certain position of the strings. Furthermore, the filters are defined by some variants of random-context conditions, i.e., they check the presence and the absence of certain symbols in the strings. For an overview on HNEPs and the best results known so far, we refer the reader to [1].

In *networks of evolutionary processors with polarizations*, each symbol has assigned a fixed integer value; the polarization of a string is computed according to a given evaluation function, and in the communication step the obtained string is moved to any of the connected cells having the same polarization. Networks of polarized evolutionary processors were considered in [4] and [3]), and networks of evolutionary processors only using the elementary polarizations  $-1, 0, 1$  were investigated in [15]. The number of processors (cells) needed to obtain computational completeness has been improved in a considerable way in [11] making these results already comparable with those obtained in [1] for hybrid networks of evolutionary processors using permitting and forbidden contexts as filters for the communication

of strings between cells.

Seen from a biological point of view, networks of evolutionary processors are a collection of cells communicating via membrane channels, which makes them closely related to tissue-like P systems considered in the area of membrane computing. Hence, in this paper we will also take over the idea of polarizations; as in [15] and in [11], we will only consider the elementary polarizations  $-1, 0, 1$  for the symbols as well as for the cells. Using this variant of tissue P systems, we are going to show computational completeness even with the sequential derivation mode.

The rest of the paper is structured as follows: In Section 2 we recall some well-known definitions from formal language theory, and in Section 3 we give the definitions of the model of tissue P systems with vesicles of multisets as well as its variants to be considered in this paper, especially the variant with elementary polarizations  $-1, 0, 1$ . In Section 4 we show our main results for tissue P systems with vesicles of multisets using all three operations insertion, deletion, and substitution, but without using polarizations, i.e., that computational completeness can be achieved by using the set maximally parallel derivation mode, whereas with the sequential mode we get a characterization of the families of sets of natural numbers and Parikh sets of natural numbers generated by partially blind register machines. In Section 5 we show that even with the sequential derivation mode we obtain computational completeness when using polarizations (only  $-1, 0, 1$  are needed). A summary of the results and an outlook to future research conclude the paper.

## 2. Prerequisites

We start by recalling some basic notions of formal language theory. An alphabet is a non-empty finite set. A finite sequence of symbols from an alphabet  $V$  is called a *string* over  $V$ . The set of all strings over  $V$  is denoted by  $V^*$ ; the *empty string* is denoted by  $\lambda$ ; moreover, we define  $V^+ = V^* \setminus \{\lambda\}$ . The *length* of a string  $x$  is denoted by  $|x|$ , and by  $|x|_a$  we denote the number of occurrences of a letter  $a$  in a string  $x$ . For a string  $x$ ,  $alph(x)$  denotes the smallest alphabet  $\Sigma$  such that  $x \in \Sigma^*$ . For a finite set  $M$ , its cardinality is denoted by  $card(M)$  or  $|M|$ .

A *multiset*  $M$  with underlying set  $A$  is a pair  $(A, f)$  where  $f : A \rightarrow \mathbb{N}$  is a mapping, with  $\mathbb{N}$  denoting the set of natural numbers (non-negative integers). If  $M = (A, f)$  is a multiset then its *support* is defined as  $supp(M) = \{x \in A \mid f(x) > 0\}$ . A multiset is empty (respectively finite) if its support is the empty set (respectively a finite set). If  $M = (A, f)$  is a finite multiset over  $A$  and  $supp(M) = \{a_1, \dots, a_k\}$ , then it can also be represented by the string  $a_1^{f(a_1)} \dots a_k^{f(a_k)}$  over the alphabet  $\{a_1, \dots, a_k\}$ ; the corresponding vector  $(f(a_1), \dots, f(a_k))$  of natural numbers is called Parikh vector of the string  $a_1^{f(a_1)} \dots a_k^{f(a_k)}$ . Moreover, all permutations of this string  $a_1^{f(a_1)} \dots a_k^{f(a_k)}$  precisely identify the same multiset  $M$  (they have the same Parikh vector). The set of all multisets over the alphabet  $V$  is denoted by  $V^\circ$ .

The family of all recursively enumerable sets of strings is denoted by  $RE$ , the corresponding family of recursively enumerable sets of Parikh sets (vectors of natural

numbers) is denoted by  $PsRE$ . In general, for any family of string languages  $\mathcal{F}$ , the corresponding family of Parikh sets is denoted by  $Ps\mathcal{F}$ .

For more details of formal language theory the reader is referred to the monographs and handbooks in this area, such as [18].

### 2.1. Insertion, deletion, and substitution

For an alphabet  $V$ , let  $a \rightarrow b$  be a rewriting rule with  $a, b \in V \cup \{\lambda\}$ , and  $ab \neq \lambda$ ; we call such a rule a *substitution rule* if both  $a$  and  $b$  are different from  $\lambda$ ; such a rule is called a *deletion rule* if  $a \neq \lambda$  and  $b = \lambda$ , and it is called an *insertion rule* if  $a = \lambda$  and  $b \neq \lambda$ . The set of all insertion rules, deletion rules, and substitution rules over an alphabet  $V$  is denoted by  $Ins_V$ ,  $Del_V$ , and  $Sub_V$ , respectively. Whereas an insertion rule is always applicable, the applicability of a deletion and a substitution rules depends on the presence of the symbol  $a$ . We remark that insertion rules, deletion rules, and substitution rules can be applied to strings as well as to multisets, too. Whereas in the string case, the position of the inserted, deleted, and substituted symbol matters, in the case of a multiset this only means incrementing the number of symbols  $b$ , decrementing the number of symbols  $a$ , or decrementing the number of symbols  $a$  and at the same time incrementing the number of symbols  $b$ .

### 2.2. Register machines

**Definition 1.** A register machine is a construct  $M = (m, B, l_0, l_h, P)$  where

- $m$  is the number of registers,
- $B$  is a set of labels,
- $l_0 \in B$  is the initial label, and
- $l_h \in B$  is the final label.

The labeled instructions of  $M$  in  $P$  can be of the following forms:

- $p : (ADD(r), q, s)$ , with  $p \in B \setminus \{l_h\}$ ,  $q, s \in B$ ,  $1 \leq r \leq m$ .  
Increase the value of register  $r$  by one, and non-deterministically jump to instruction  $q$  or  $s$ .
- $p : (SUB(r), q, s)$ , with  $p \in B \setminus \{l_h\}$ ,  $q, s \in B$ ,  $1 \leq r \leq m$ .  
If the value of register  $r$  is not zero then decrease the value of register  $r$  by one (decrement case) and jump to instruction  $q$ , otherwise jump to instruction  $s$  (zero-test case).
- $l_h : HALT$ . Stop the execution of the register machine.

A configuration of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed.

In the accepting case, a computation starts with the input of a  $k$ -vector of natural numbers in its first  $k$  registers and by executing the first instruction of  $P$  (labeled

6 *Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan*

with  $l_0$ ); it terminates with reaching the *HALT*-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the generating case, a computation starts with all registers being empty and by executing the first instruction of  $P$  (labeled with  $l_0$ ); it terminates with reaching the *HALT*-instruction and the output of a  $k$ -vector of natural numbers in its first  $k$  registers. Without loss of generality, we may assume all registers  $> k$  to be empty at the end of the computation. The set of vectors of natural numbers computed by  $M$  in this way is denoted by  $Ps(M)$ . If we want to generate only numbers (1-dimensional vectors), then we have the result of a computation in register 1 and the set of numbers computed by  $M$  in this way is denoted by  $N(M)$ . By *NRM* and *PsRM* we denote the families of sets of natural numbers and of sets of vectors of natural numbers, respectively, generated by register machines.

Register machines are well-known universal devices for computing (generating or accepting) sets of vectors of natural numbers. It is folklore (e.g., see [14]) that  $PsRE = PsRM$  and  $NRE = NRM$  (actually, three registers are sufficient in order to generate any set from the family *NRE*, and, in general,  $k + 2$  registers needed to generate any set of from the family *NRE*).

### 2.2.1. *Partially blind register machines*

In the case when a register machine cannot check whether a register is empty we say that it is partially blind: the registers are increased and decreased by one as usual, but if the machine tries to subtract from an empty register, then the computation aborts without producing any result (that is we may say that the subtract instructions are of the form  $p : (SUB(r), q, abort)$ ; instead, we simply will write  $p : (SUB(r), q)$ . Moreover, acceptance or generation now by definition also requires all registers, except the first  $k$  output registers, to be empty (which means all registers  $k + 1, \dots, m$  have to be empty at the end of the computation), i.e., there is an implicit test for zero at the end of a (successful) computation, that is why we say that the device is partially blind. By *NPBRM* and *PsPBRM* we denote the families of sets of natural numbers and of sets of vectors of natural numbers, respectively, computed by partially blind register machines. It is known (e.g., see [7]) that partially blind register machines are strictly less powerful than general register machines (hence than Turing machines); moreover, *NPBRM* and *PsPBRM* characterize the number sets and Parikh sets, respectively, obtained by matrix grammars without appearance checking.

## 3. Tissue P Systems Working on Vesicles of Multisets

We first define our basic model of tissue P systems working on vesicles of multisets in the maximally parallel set derivation mode:

**Definition 2.** A tissue P systems working on vesicles of multisets (*a tPV system for short*) is a tuple  $\Pi = (L, V, T, R, (i_0, w_0), h)$  where

- $L$  is a set of labels identifying in a one-to-one manner the  $|L|$  cells of the tissue P system  $\Pi$ ;
- $V$  is the alphabet of the system,
- $T$  is the terminal alphabet of the system,
- $R$  is a set of rules of the form  $(i, p, j)$  where  $i, j \in L$  and  $p \in \text{Ins}_V \cup \text{Del}_V \cup \text{Sub}_V$ , i.e.,  $p$  is an insertion, deletion or substitution rule over the alphabet  $V$ ; we may collect all rules from cell  $i$  in one set and then write  $R_i = \{(i, p, j) \mid (i, p, j) \in R\}$ , so that  $R = \bigcup_{i \in L} R_i$ ; moreover, for the sake of conciseness, we may simply write  $R_i = \{(p, j) \mid (i, p, j) \in R\}$ , too;
- $(i_0, w_0)$  describes the initial vesicle containing the multiset  $w_0$  in cell  $i_0$
- $h$  is the (label of the) output cell.

As in the case of NEPs and HNEPs, we call  $\Pi$  a *hybrid tPV* system if every cell is “specialized” in one type of evolution rules from (at most) one of the sets  $\text{Ins}_V$ ,  $\text{Del}_V$ , and  $\text{Sub}_V$ , respectively.

The tPV system can work with different derivation modes for applying the rules in  $R$ . The simplest case is the sequential mode (abbreviated *sequ*), where in each derivation step, with the vesicle enclosing the multiset  $w$  being in cell  $i$ , exactly one rule  $(i, p, j)$  from  $R_i$  is applied, which in fact means that  $p$  is applied to  $w$  and the resulting multiset in its vesicle is moved to cell  $j$ . Using the set maximally parallel derivation mode (abbreviated *smax*), with the vesicle enclosing the multiset  $w$  being in cell  $i$ , we apply a set of rules from  $R_i$  which has to obey the condition that all the evolution rules  $(i, p, j)$  in this multiset of rules specify the same target cell  $j$  and is non-extendable with respect to this condition. We also consider the variants of the set maximally parallel derivation mode where with respect to this condition the maximal number of rules has to be applied (this mode is abbreviated by *smaxrules*) or where the maximal number of objects has to be affected by the applied rules (this mode is abbreviated by *smaxobjects*). In all cases, we first non-deterministically choose a non-empty set of rules where all the rules indicate the same target and *then* check the maximality condition.

In any case, the computation of  $\Pi$  starts with a vesicle containing the multiset  $w_0$  in cell  $i_0$ , and the computation proceeds in the underlying derivation mode until an output condition is fulfilled, which in all possible cases means that the vesicle has arrived in the output cell  $h$ . As we are dealing with membrane systems, the classic additional condition may be that the computation halts, i.e., in cell  $h$  no rule can be applied any more to the multiset in the vesicle which has arrived there. As we have also specified a terminal alphabet, another condition – for its own or in combination with halting – is that the multiset in the vesicle which has arrived in cell  $h$  only contains terminal symbols. Hence, we may specify one of the following output strategies:

- *halt*: the only condition is that the system halts, the result is the multiset contained in the vesicle to be found in cell  $h$  (which in fact means that



- specifying the terminal alphabet is obsolete);
- *term*: the resulting multiset contained in the vesicle to be found in cell  $h$  consists of terminal symbols only (yet the system need not have reached a halting configuration).
  - *(halt, term)*: both conditions must be fulfilled, i.e., the system halts and the resulting multiset contained in the vesicle to be found in cell  $h$  consists of terminal symbols only.

The set of all multisets obtained as results of computations in  $\Pi$  working in the derivation mode  $\alpha \in \{sequ, smax, smax_{rules}, smax_{objects}\}$  with the output being obtained by taking the output condition  $\beta \in \{halt, term, (halt, term)\}$  is denoted by  $Ps(\Pi, \alpha, \beta)$ ; if we are only interested in the number of symbols in the resulting multiset, the corresponding set of natural numbers is denoted by  $N(\Pi, \alpha, \beta)$ . The families of sets of ( $k$ -dimensional) vectors of natural numbers and sets of natural numbers generated by tPV systems with at most  $n$  cells working in the derivation mode  $\alpha$  and using the output strategy  $\beta$  are denoted by  $Ps(tPV_n, \alpha, \beta)$  ( $Ps^k(tPV_n, \alpha, \beta)$ ) and  $N(tPV_n, \alpha, \beta)$ , respectively. If  $n$  is not bounded, we simply omit the subscript in these notations.

We should like to mention that the communication structure between the cells in a tPV system is implicitly given by the rules in  $R$ , i.e., the underlying (directed!) graph  $G = (N, E)$  with  $N$  being the set of nodes and  $E$  being the set of (directed) edges is given by  $N = L$  and  $E = \{(i, j) \mid (i, p, j) \in R\}$ .

In general, we do not forbid  $G$  to have loops. Now consider the corresponding *undirected* graph  $G_u = (N, E_u)$  with  $E_u = \{\{i, j\} \mid (i, j) \in E \text{ or } (j, i) \in E\}$ . Then we may have the special situation that  $E_u$  is a tree; in this case, we call the tPV system  $\Pi$  a *hierarchical P system working on vesicles of multisets* (abbreviated *PV system*); in all definitions given above for the families of sets of (vectors of) natural numbers we then write *PV* instead of *tPV*.

#### 4. Results for Tissue P Systems with Vesicles of Multisets

Our first result shows that with one of the set maximally parallel derivation modes  $\alpha \in \{smax, smax_{rules}, smax_{objects}\}$  and using all three types of point mutation rules computational completeness can even be obtained with PV systems:

**Theorem 3.**  $PsRE \subseteq Ps(PV, \alpha, \beta)$ ,  
for any set maximally parallel derivation mode  $\alpha \in \{smax, smax_{rules}, smax_{objects}\}$   
and any halting strategy  $\beta \in \{(halt, term), halt, term\}$ .

**Proof.** Let  $K$  be an arbitrary recursively enumerable set of  $k$ -dimensional vectors of natural numbers. Then  $K$  can be generated by a register machine  $M$  with two working registers also using decrement instructions and  $k$  output registers. In order to have a general construction, we do not restrict the number of working registers in the following. Let  $M = (m, B, l_0, l_h, P)$  be a register machine generating  $K$ .

We now define a PV system  $\Pi$  generating  $K$ , i.e.,  $Ps(\Pi, smax, \beta) = K$ :

$$\begin{aligned} \Pi &= (L, V, T, R, (i_0, w_0) = (0, l_0), h), \\ L &= \{r \mid 1 \leq r \leq k\} \cup \{r, r_-, r_0 \mid k+1 \leq r \leq m\} \cup \{0, h\}, \\ V &= L \cup \{a_r \mid 1 \leq r \leq m\} \cup B \cup \{\#\}, \\ T &= \{a_r \mid 1 \leq r \leq k\}, \\ R &= \{(0, p \rightarrow q, r), (0, p \rightarrow s, r), (r, \lambda \rightarrow a_r, 0) \mid p : (ADD(r), q, s) \in P\}, \\ &\quad \cup \{(0, p \rightarrow q, r_-), (0, p \rightarrow s, r_0) \mid p : (SUB(r), q, s) \in P\} \\ &\quad \cup \{(r_-, a_r \rightarrow \lambda, 0), (r_0, s \rightarrow s, 0), (r_0, a_r \rightarrow \#, 0) \mid p : (SUB(r), q, s) \in P\}, \\ &\quad \cup \{(0, l_h \rightarrow \lambda, h), (h, \# \rightarrow \#, 0), (0, \# \rightarrow \#, h)\}. \end{aligned}$$

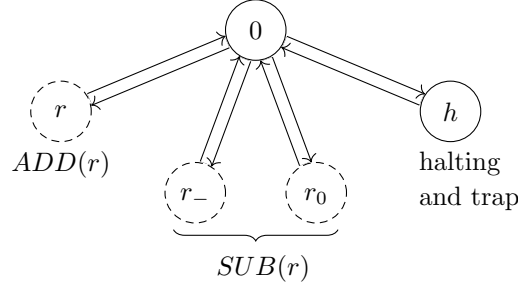


Fig. 1. Communication structure of the two-level hierarchical PV system. Each node with a dashed contour is replicated for every register  $r$ .

The root of the communication tree is cell 0. From there, all simulations of register machine instructions are initiated:

$(ADD(r), q, s)$  is simulated by moving the vesicle from the root cell to cell  $r$  by applying one of the rules from  $\{(0, p \rightarrow q, r), (0, p \rightarrow s, r)\}$ ; in cell  $r$  the number of symbols  $a_r$  representing the contents of register  $r$  is incremented by the insertion rule  $(r, \lambda \rightarrow a_r, 0)$ , which also sends back the vesicle to the root cell.

$(SUB(r), q, s)$  is simulated by first choosing one of the rules from  $\{(0, p \rightarrow s, r_0), (0, p \rightarrow q, r_-)\}$  in a non-deterministic way, guessing whether the number of symbols  $a_r$  representing the contents of register  $r$  is zero or not. If the number is not zero, then in cell  $r_-$  the deletion operation in the rule  $(r_-, a_r \rightarrow \lambda, 0)$  can be carried out and the vesicle is sent back to cell 0, whereas otherwise the vesicle gets stuck in cell  $r_-$  and therefore no result can be obtained in the output cell  $h$ . If the number of symbols  $a_r$  has been assumed to be zero and the vesicle is in cell  $r_0$ , then there the rule  $(r_0, s \rightarrow s, 0)$  can be applied in any case, and the vesicle is sent back to cell 0. Yet if the assumption has been wrong, then in parallel the rule

10 *Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan*

$(r_0, a_r \rightarrow \#, 0)$  must be applied, thus introducing the trap symbol  $\#$ . This is the only case in the whole construction where the possibility of applying (at least) two rules in parallel is used for appearance checking. We point out that both rules have the same target 0.

Any halting computation in  $M$  finally reaches the halting instruction labeled by  $l_h$ , and thus in  $\Pi$ , by applying the rule  $(0, l_h \rightarrow \lambda, h)$ , the vesicle obtained so far is moved to the final cell  $h$ . Provided no trap symbol  $\#$  has been generated during the simulation of the computation in  $M$  by the tPV system  $\Pi$ , the multiset in this vesicle only contains terminal symbols and the computation in  $\Pi$  halts as well.

In sum, we conclude that  $Ps(\Pi, smax, \beta) = K$  for any halting strategy  $\beta \in \{(halt, term), halt, term\}$ . Yet our construction (in fact, we only have to check the situation in cell  $r_0$ ) also yields  $K = Ps(\Pi, smax_{rules}, \beta) = Ps(\Pi, smax_{objects}, \beta)$ , which observation completes the proof  $\square$

The following corollary is immediate consequence of the preceding theorem:

**Corollary 4.**  $PsRE = Ps(PV, \alpha, \beta) = Ps(tPV, \alpha, \beta)$ ,  
for any set maximally parallel derivation mode  $\alpha \in \{smax, smax_{rules}, smax_{objects}\}$   
and any halting strategy  $\beta \in \{(halt, term), halt, term\}$ .

**Proof.** By definition, any PV system is a tPV system, too. Hence, it only remains to show that  $Ps(tPV, smax, \beta) \subseteq PsRE$ , yet we omit a direct construction as the result can be inferred from the Turing-Church thesis.  $\square$

The construction given in the proof of Theorem 3 offers some additional nice features:

- The PV system  $\Pi$  is a hybrid one, as in each cell only one kind of rules is employed: substitution in cells 0 and  $h$  and in cells  $r_0$ , insertion in cells  $r$ , deletion in cells  $r_-$ .
- The trap rules  $(h, \# \rightarrow \#, 0)$ ,  $(0, \# \rightarrow \#, h)$ , guaranteeing a non-halting computation as soon as the introduction of the trap symbol  $\#$  has been enforced by a wrong guess, are only needed in the case of the output strategy *halt*.
- The vesicle always leaves the current cell whenever a rule can be applied.
- The number of cells in the PV system  $\Pi$  only depends on the number of registers in the register machine  $M$ . Suppose  $M$  has  $k$  output registers and 2 working registers. Since the output registers are never decremented, we only need one cell  $r$  for each such register. We need 3 cells ( $r$ ,  $r_-$ , and  $r_0$ ) for each of the two working (decrementable) registers. Finally, we need the cells 0 and  $h$ , which amounts in a total of  $k + 2 \cdot 3 + 2 = k + 8$  cells to simulate  $M$ . This also means that only 9 cells are needed for generating number sets.

If the underlying register machine is partially blind, we only have to consider the decrement case, which then still works correctly, whereas we can omit the zero test case, and thus can omit the parallelism. Hence, we immediately infer the following result:

**Theorem 5.** For any  $\beta \in \{(halt, term), halt, term\}$ ,  
 $PsPBRM \subseteq Ps(PV, sequ, \beta)$ .

**Proof.** Let  $K \in PsPBRM$ , i.e., the vector set  $K$  can be generated by a partially blind register machine  $M = (m, B, l_0, l_h, P)$ . As in the preceding proof, we now define a PV system  $\Pi$  generating  $K$  in the sequential derivation mode, i.e.,  $Ps(\Pi, sequ, \beta) = K$ :

$$\begin{aligned} \Pi &= (L, V, T, R, (i_0, w_0) = (0, l_0), h), \\ L &= \{r \mid 1 \leq r \leq k\} \cup B \cup \{r, r_- \mid k+1 \leq r \leq m\} \cup \{h\}, \\ V &= L \cup \{a_r \mid 1 \leq r \leq m\} \cup \{\#\}, \\ T &= \{a_r \mid 1 \leq r \leq k\}, \\ R &= \{(0, p \rightarrow q, r), (0, p \rightarrow s, r), (r, \lambda \rightarrow a_r, 0) \mid p : (ADD(r), q, s) \in P\}, \\ &\cup \{(0, p \rightarrow q, r_-), (r_-, a_r \rightarrow \lambda, 0) \mid p : (SUB(r), q) \in P\}, \\ &\cup \{(0, l_h \rightarrow \lambda, h), (h, \# \rightarrow \#, 0), (0, \# \rightarrow \#, h)\} \\ &\cup \{(h, a_r \rightarrow \#, 0) \mid k+1 \leq r \leq m\}. \end{aligned}$$

The simulation of the computations in  $M$  by  $\Pi$  works in a similar way as in the preceding proof, with the main reduction that no zero test case has to be simulated, hence, everything can be carried out in a sequential way.

Any halting computation in  $M$  finally reaches the halting instruction labeled by  $l_h$ , and thus in  $\Pi$ , by applying the rule  $(0, l_h \rightarrow \lambda, h)$ , the vesicle obtained so far is moved to the final cell  $h$ . Provided no non-terminal symbol  $a_r$  with  $k+1 \leq r \leq m$  is still present, the computation in  $\Pi$  will halt, but otherwise the trap symbol  $\#$  will be introduced by (one of) the rules from  $\{(h, a_r \rightarrow \#, 0) \mid k+1 \leq r \leq m\}$ , thus causing an infinite loop. In sum, we conclude that  $Ps(\Pi, sequ, \beta) = K$  for any  $\beta \in \{(halt, term), halt, term\}$ .  $\square$

We now also show that the computations of a sequential tPV system using the output strategy *term* can be simulated by a partially blind register machine.

**Theorem 6.**  $Ps(tPV, sequ, term) \subseteq PsPBRM$ .

**Proof.** Let  $\Pi = (L, V, T, R, (i_0, w_0), h)$  be an arbitrary tPV system working in the sequential derivation mode yielding an output in the output cell provided the multiset in the vesicle having arrived there contains only terminal symbols; without loss of generality we assume  $L = \{i \mid 1 \leq i \leq l\}$ .

12 *Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan*

We now construct a register machine  $M = (m, B, l_0, l_h, P)$  generating  $Ps(\Pi, sequ, term)$ , yet using a more relaxed definition for the labeling of instructions in  $M$ , i.e., one label may be used for different instructions, which does not affect the computational power of the register machine as shown in [7]. For example, instead of a nondeterministic ADD-instruction  $p : (ADD(r), q, s)$  we use the two ADD-instructions  $p : (ADD(r), q)$  and  $p : (ADD(r), s)$ . Moreover, we omit the sequence of instructions for generating the representation of  $w_0$  in the initial vesicle  $i_0$  by a sequence of ADD-instructions starting with the instruction labeled by  $l_0$  and finally ending up with the label  $i_0$  and the correct values in registers  $r$  representing the numbers of symbols  $a_r$  in the initial vesicle in cell  $i_0$ . We denote the set of labels used for this initialization procedure by  $L_0$  and the corresponding set of labeled instructions by  $P_0$ .

For all the rules in cell  $i$  of the tPV system  $\Pi$  we use the same label  $i$  in the partially blind register machine  $M$ ; the rules of  $\Pi$  then can be simulated by register machine instructions in  $M$  as follows:

$(i, \lambda \rightarrow b, j)$  is simulated by  $i : (ADD(b), j)$ ;  
 $(i, a \rightarrow \lambda, j)$  is simulated by  $i : (SUB(a), j)$ ;  
 $(i, a \rightarrow b, j)$  is simulated by the sequence of two instructions  $i : (SUB(a), l_{b,j})$  and  $l_{b,j} : (ADD(b), j)$  using an intermediate label  $l_{b,j}$ ; hence, for these simulations we may need  $card(V) \times card(L)$  additional labels.

If a vesicle reaches the final cell  $h$  with the multiset inside only consisting of terminal symbols, we also have to allow  $M$  to have this multiset as a result: this goal can be accomplished by using the final sequence

$$h : (ADD(1), \tilde{h}), \quad \tilde{h} : (SUB(1), \hat{h}), \quad \hat{h} : HALT.$$

We observe that  $\tilde{h}, \hat{h}$  are labels different from  $h$ . Since  $l_h = \hat{h}$  now is the only halting instruction of  $M$ , it must reset to zero all its working registers before reaching  $\hat{h}$  to satisfy the final zero check, which corresponds to  $\Pi$  producing a multiset consisting exclusively of terminal symbols, i.e., the output strategy  $term$ .

According to the construction described above, for the register machine  $M = (m = card(V), B, l_0, l_h = \hat{h}, P)$  we have obtained the following constituents:

$$\begin{aligned} B &= L_0 \cup L \cup \{l_{b,j} \mid b \in V, j \in L\}, \\ P &= P_0 \cup \{i : (ADD(b), j) \mid (i, \lambda \rightarrow b, j) \in R\} \\ &\cup \{i : (SUB(a), j) \mid (i, a \rightarrow \lambda, j) \in R\} \\ &\cup \{i : (SUB(a), l_{b,j}) \mid (i, a \rightarrow b, j) \in R\} \\ &\cup \{l_{b,j} : (ADD(b), j) \mid b \in V, j \in L\} \\ &\cup \{h : (ADD(1), \tilde{h}), \tilde{h} : (SUB(1), \hat{h}), \hat{h} : HALT\}. \end{aligned}$$

In sum, we conclude that  $Ps(M) = Ps(\Pi, sequ, term)$ . □

As a consequence of Theorems 5 and 6 we obtain:

**Corollary 7.**  $PsPBRM = Ps(PV, sequ, term) = Ps(tPV, sequ, term)$ .

**Proof.** Summarizing the results obtained above, we shown:

- $PsPBRM \subseteq Ps(PV, sequ, term)$  according to Theorem 5;
- $Ps(PV, sequ, term) \subseteq Ps(tPV, sequ, term)$  by definition; and
- $Ps(tPV, sequ, term) \subseteq PsPBRM$  according to Theorem 6;

hence, in sum, we conclude

$$PsPBRM = Ps(PV, sequ, term) = Ps(tPV, sequ, term). \quad \square$$

## 5. Polarized Tissue P Systems with Vesicles of Multisets

In a polarized tissue P system  $\Pi$  working on vesicles of multisets, each cell gets assigned an elementary polarization from  $\{-1, 0, 1\}$ ; each symbol from the alphabet  $V$  also has an integer polarization but every terminal symbol from the terminal alphabet has polarization 0. As we shall see later, we can even restrict ourselves to elementary polarizations from  $\{-1, 0, 1\}$  for each symbol, too.

Given a multiset, we need an evaluation function computing the polarization of the whole multiset from the polarizations of the symbols it contains. Given the result  $m$  of this evaluation of the multiset in the vesicle, we apply the sign function  $sign(m)$ , which returns one of the values  $+1/0/-1$ , provided that  $m$  is a positive integer / is 0 / is a negative integer, respectively.

The main difference between polarized tPV systems and normal tPV systems, besides the polarizations assigned to symbols and multisets as well as to the cells, is the way the resulting vesicles are moved from one cell to another one: although in the rules themselves still a target is specified, the vesicle can only move to a cell having the same polarization as the multiset contained in it. As a special additional feature we require that the vesicle must not stay in the current cell even if its polarization would fit (if there is no other cell with a fitting polarization, the vesicle is eliminated from the system). As by the convention mentioned above we assume every terminal symbol from the terminal alphabet to have polarization 0, it is necessary that the output cell itself also has to have polarization 0.

**Definition 8.** A polarized tissue P systems working on vesicles of multisets (*a ptPV system for short*) is a tuple

$$\Pi = (L, V, T, R, (i_0, w_0), h, \pi_L, \pi_V, \varphi)$$

where

- $L$  is a set of labels identifying in a one-to-one manner the  $|L|$  cells of the tissue P system  $\Pi$ ;
- $V$  is the polarized alphabet of the system,
- $T$  is the terminal alphabet of the system (the terminal symbols have no polarization, i.e., polarization 0),

- $R$  is a set of rules of the form  $(i, p, j)$  where  $i, j \in L$  and  $p \in \text{Ins}_V \cup \text{Del}_V \cup \text{Sub}_V$ , i.e.,  $p$  is an insertion, deletion or substitution rule over the alphabet  $V$ ; collecting all rules from cell  $i$  in one set, we write  $R_i = \{(i, p, j) \mid (i, p, j) \in R\}$ , i.e.,  $R = \bigcup_{i \in L} R_i$ ; moreover, for the sake of conciseness, we also write  $R_i = \{(p, j) \mid (i, p, j) \in R\}$ , too;
- $(i_0, w_0)$  describes the initial vesicle containing the multiset  $w_0$  in cell  $i_0$ ;
- $\pi_L$  is the function assigning an integer polarization to each cell (as already mentioned above, we here restrict ourselves to the elementary polarizations from  $\{-1, 0, 1\}$ );
- $\pi_V$  is the function assigning an integer polarization to each symbol in  $V$  (as already mentioned above, we here restrict ourselves to the elementary polarizations from  $\{-1, 0, 1\}$ );
- $\varphi$  is the evaluation function yielding an integer value for each multiset.

As in the case of NEPs and HNEPs, we call  $\Pi$  a *hybrid* ptPV system if a cell is “specialized” in one type of evolution rules from (at most) one of the sets  $\text{Ins}_V$ ,  $\text{Del}_V$ , and  $\text{Sub}_V$ , respectively.

The ptPV system again can work with different derivation modes for applying the rules in  $R$ , e.g., the sequential mode *sequ* or the set maximally parallel derivation modes *smax*, *smax<sub>rules</sub>*, *smax<sub>objects</sub>*. Yet a derivation step now consists of two substeps – the *evolutionary step* with applying the rule(s) from  $R$  in the way required by the derivation mode and the *communication step* with sending the vesicle to a cell with the same polarization as the multiset in it.

In the following, we will only use the specific evaluation function  $\varphi$  which computes the value of a multiset as the sum of the values of the symbols contained in it; we write  $\varphi_s$  for this function.

In any case, the computation of  $\Pi$  starts with a vesicle containing the multiset  $w_0$  in cell  $i_0$  (obviously, the initial multiset  $w_0$  must have the same polarization as the initial cell  $i_0$ ), and the computation proceeds using the underlying derivation mode for the evolutionary steps until an output condition is fulfilled, which in all possible cases means that the vesicle has arrived in the output cell  $h$ . Again we use one of the output strategies *halt*, *term* and *(halt, term)*.

The set of all multisets obtained as results of computations in  $\Pi$  working in the derivation mode  $\alpha \in \{\text{sequ}, \text{smax}, \text{smax}_{\text{rules}}, \text{smax}_{\text{objects}}\}$ , using the evaluation function  $\varphi_s$  and the output condition  $\beta \in \{\text{halt}, \text{term}, (\text{halt}, \text{term})\}$ , is denoted by  $Ps(\Pi, \alpha, \beta)$ ; if we are only interested in the number of symbols in the resulting multiset, the corresponding set of natural numbers is denoted by  $N(\Pi, \alpha, \beta)$ . The families of sets of ( $k$ -dimensional) vectors of natural numbers and sets of natural numbers generated by ptPV systems with at most  $n$  cells working in the derivation mode  $\alpha$  and using the output strategy  $\beta$  are denoted by  $Ps(\text{ptPV}_n, \alpha, \beta)$  ( $Ps^k(\text{ptPV}_n, \alpha, \beta)$ ) and  $N(\text{ptPV}_n, \alpha, \beta)$ , respectively. If  $n$  is not bounded, we simply omit the subscript in these notations.

Again we mention that the possible communication structure between the cells in a ptPV system is implicitly given by the rules in  $R$ , i.e., the underlying (directed) graph  $G = (N, E)$  with  $N$  being the set of nodes and  $E$  being the set of (directed) edges is given by  $N = L$  and  $E = \{(i, j) \mid (i, p, j) \in R\}$ .

Moreover, in general, again we do not forbid  $G$  to have loops. Now consider the corresponding *undirected* graph  $G_u = (N, E_u)$  with  $E_u = \{\{i, j\} \mid (i, j) \in E \text{ or } (j, i) \in E\}$ . Then we may have the special situation that  $E_u$  is a tree; in this case, we call the ptPV system  $\Pi$  a **hierarchical polarized P system working on vesicles of multisets** (abbreviated *pPV system*); in all definitions given above for the families of sets of (vectors of) natural numbers we then write *pPV* instead of *ptPV*.

**Remark 9.** *In all variants for polarized P systems working on vesicles of multisets defined above, a computation step consists of the following phases:*

- choose a non-empty set of rules  $R'$  applicable to the vesicle with all rules having the same target  $t$ ;
- check that  $R'$  cannot be extended any more with another rule from  $R$  with target  $t$ ;
- check if the polarization of the resulting vesicle after the application of  $R'$  is the same as the polarization of the target cell  $t$ ;
- if the polarization of the resulting vesicle is the same as the polarization of the target cell  $t$ , then move the vesicle to cell  $t$ , otherwise try to apply another set of rules.

Finally, we may even consider the variant where  $G$  is interpreted as an undirected graph  $(L, \{\{i, j\} \mid (i, p, j) \in R\})$ . Then we may adopt the way of communication from polarized HNEPs and instead of specifying the set of rules as given above, change the definition in the following way:

$$\Pi = (L, V, T, R, (i_0, w_0), h, \pi_L, \pi_V, \varphi, G)$$

where  $G$  now is an undirected graph defining the communication structure between the cells, and the rules in  $R$  are specified without targets, i.e., they are written as  $(i, p)$  instead of  $(i, p, j)$  as the targets now are specified by the communication graph  $G$ . We now also write  $R_i = \{p \mid (i, p) \in R\}$ ,  $i \in L$ . As  $G$  is an *undirected* graph this makes a big difference as we cannot enforce the direction of the movement of the vesicle anymore. We call such a system with an undirected communication graph a *uptPV system* (with *u* specifying that the communication structure is an undirected graph).

**Remark 10.** *The notion uptPV system is justified because uptPV systems are a special case of ptPV systems. In fact, if*

$$\Pi = (L, V, T, R, (i_0, w_0), h, \pi_L, \pi_V, \varphi, G),$$



16 *Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan*

is a *uptPV* system and  $p \in R_i$  for  $i \in L$ , i.e.,  $(i, p) \in R$ , then this can be captured in a corresponding *ptPV* system

$$\Pi' = (L, V, T, R', (i_0, w_0), h, \pi_L, \pi_V, \varphi)$$

by defining

$$R'_i = \{(i, p, j) \mid p \in R_i, j \in L\}, i \in L.$$

Even with *uptPV* systems we can obtain computational completeness with the sequential derivation mode:

**Theorem 11.** For any  $\beta \in \{\text{halt, term}, \text{halt, term}\}$ ,  
 $PsRE \subseteq Ps(\text{uptPV}, \text{sequ}, \beta)$ .

**Proof.** Let  $M = (m, B, l_0, l_h, P)$  be an arbitrary register machine generating  $k$ -dimensional vectors. We now construct a *uptPV* system  $\Pi$  generating the same set of multisets as  $M$ , i.e.,  $Ps(\Pi, \text{sequ}, \beta) = Ps(M)$ .

$$\begin{aligned} \Pi &= (L, V, T, R, (0, l_0), \hat{l}_h, \pi_L, \pi_V, \varphi_s, G), \\ L &= \{0, l_h, \tilde{l}_h, \hat{l}_h\} \cup \{r, r_+, \hat{r}_+, r_0, \tilde{r}_0, \hat{r}_0, r_-, \tilde{r}_-, \bar{r}_-, \hat{r}_- \mid 1 \leq r \leq m\}, \\ V &= \{a_r, a_r^-, a_r^+ \mid 1 \leq r \leq m\} \cup \{p, p^+, p^-, \hat{p}^+, \tilde{p}^- \mid p \in B\}, \\ T &= \{a_r \mid 1 \leq r \leq k\}. \end{aligned}$$

The evaluation  $\pi_V$  for the symbols in  $V$  corresponds to the superscript of the symbol, i.e., for  $\alpha^z \in V$  with  $z \in \{+, 0, -\}$  we define  $\pi_V(\alpha^0) = 0$  (we usually omit the superscript 0),  $\pi_V(\alpha^+) = +1$ , and  $\pi_V(\alpha^-) = -1$ .

The connection structure, i.e., the undirected graph  $G$ , as well as the polarizations of the cells given by  $\pi_L$  can directly be derived from the graph depicted in Figure 2. The rules from  $R$  are grouped in five different groups;  $R$  is the union of all the sets  $R_u$ ,  $u \in L$  as defined below.

The set of rules in the central cell 0 is defined as follows:

$$\begin{aligned} R_0 &= \{p \rightarrow p \mid p : (ADD(r), q, s) \in P\} \\ &\cup \{p \rightarrow p^+, p \rightarrow p^- \mid p : (SUB(r), q, s) \in P\} \\ &\cup \{l_h \rightarrow l_h^+\} \end{aligned}$$

For the three cells in the *increment group* used for simulating the ADD-instructions we define the following sets of rules:

$$\begin{aligned} R_r &= \{p \rightarrow p^+ \mid p : (ADD(r), q, s) \in P\} \\ R_{r^+} &= \{\lambda \rightarrow a_r\} \\ R_{\hat{r}^+} &= \{p^+ \rightarrow q, p^+ \rightarrow s \mid p : (ADD(r), q, s) \in P\} \end{aligned}$$

For simulating SUB-instructions we have two paths, one of which has to be chosen in a non-deterministic way, one assuming the underlying register to be empty (*zero check group*), the other one assuming it to be non-empty (*decrement group*):

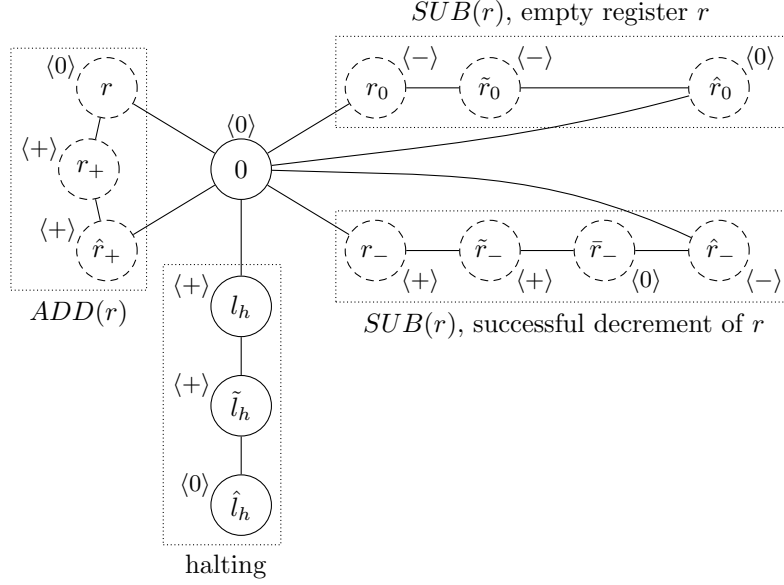


Fig. 2. The communication graph  $G$  of the computationally complete uptPV system II. We also represent the polarizations of the nodes in angular brackets. Each node with a dashed contour is replicated for every register  $r$ .

*zero check group*

$$R_{r_0} = \{p^- \rightarrow \tilde{p}^- \mid p : (SUB(r), q, s) \in P\}$$

$$R_{\tilde{r}_0} = \{\tilde{p}^- \rightarrow s \mid p : (SUB(r), q, s) \in P\}$$

$$R_{\hat{r}_0} = \{a_r \rightarrow a_r^+\}$$

*decrement group*

$$R_{r_-} = \{p^+ \rightarrow \tilde{p}^+ \mid p : (SUB(r), q, s) \in P\}$$

$$R_{\tilde{r}_-} = \{a_r \rightarrow a_r^- \mid p : (SUB(r), q, s) \in P\}$$

$$R_{\bar{r}_-} = \{\tilde{p}^+ \rightarrow s \mid p : (SUB(r), q, s) \in P\}$$

$$R_{\hat{r}_-} = \{a_r^- \rightarrow \lambda\}$$

*halting group*

$$R_{l_h} = \{l_h^+ \rightarrow \tilde{l}_h^+\}$$

$$R_{\tilde{l}_h} = \{\tilde{l}_h^+ \rightarrow \lambda\}$$

$$R_{\hat{l}_h} = \emptyset$$

We now explain in more detail how the simulations work and, moreover, also argue that no erroneous computations can end with the vesicle arriving in the final cell  $\hat{l}_h$ .

**root cell 0** All simulations start from cell 0 and again end there. The correct simulation of any instruction from  $P$  starts with the suitably chosen rule from  $R_0$ .

**increment group** Any ADD-instruction  $p : (ADD(r), q, s)$  is simulated by passing from cell 0 to the suitable cell  $r$ , from where only the correct path through  $r_+$  and then  $\hat{r}_+$  for the suitable  $r$  will lead back to cell 0. If at the beginning of this path the wrong  $r$  is chosen, i.e., the vesicle arrives in the wrong cell  $r$ , then the vesicle gets stuck there, as no rule can be applied. If the vesicle goes to a cell  $\hat{r}_0$ , then it can get stuck there if the rule  $a_r \rightarrow a_r^+$  there cannot be applied or otherwise, after the application of this rule the polarization has changed to  $+1$  and cell  $\hat{r}_0$  has no connection to any cell with polarization  $+1$ .

In order to guarantee that in the next step of the simulation the rule  $\lambda \rightarrow a_r$  in cell  $r_+$  is applied only once in cell  $r_+$ , we need the condition that after the application of a rule the vesicle has to leave the cell, which here means that the vesicle has to pass from cell  $r_+$  to cell  $\hat{r}_+$  where the polarization is changed so that the vesicle will not be able to immediately return to cell  $r_+$ . We observe that a rule from  $R_{\hat{r}_+}$  must be applicable, as otherwise the vesicle could not have arrived in cell  $\hat{r}_+$ .

We also observe that no vesicle with a  $p^+$  can go from cell 0 to cell  $\hat{r}_+$  without the vesicle then immediately being caught there in cells  $\hat{r}_+$  and  $r_+$ , as the  $p^+$  from cell 0 is for a SUB-instruction or the HALT-instruction and the rules in  $\hat{r}_+$  are for labels of ADD-instructions.

**zero check group** Cell 0 sends the vesicle to  $r_0$  by non-deterministically applying the rule  $p \rightarrow p^-$  and thus setting the polarization of the multiset to  $-1$  and remains  $-1$  if the vesicle moves to cell  $\tilde{r}_0$ . Only in case the vesicle has moved to the cell  $r_0$  with the correct  $r$ , the corresponding rule  $p^- \rightarrow \tilde{p}^-$  can be applied in cell  $r_0$  and then in  $\tilde{r}_0$  the suitable rule  $\tilde{p}^- \rightarrow s$  can be applied, which is the only way to get back to polarization 0 and thus to come back to cell 0 via cell  $\hat{r}_0$ . If the rule  $a_r \rightarrow a_r^+$  in cell  $\tilde{r}_0$  is applicable, then the polarization goes to  $+1$ , and therefore the correct continuation in cell 0 – without having applied a rule in cell  $\hat{r}_0$  – is blocked.

**decrement group** Cell 0 sends the vesicle to  $r_-$  by non-deterministically applying a rule  $p \rightarrow p^+$  and by setting the polarization of the multiset to  $+1$ . Now, in total, passing the sequence of cells  $0-r_--\tilde{r}_--\bar{r}_--\hat{r}_-$  allows for decrementing the number of symbols  $a_r$ . As already explained in the *increment group*, the computation gets stuck if the vesicle moves to cell  $l_h$  or to a cell  $\hat{r}_0$ , as in these cells no rule can be applied to change the polarization again.

If the rule  $a_r \rightarrow a_r^-$  in cell  $\bar{r}_-$  cannot be applied, the polarization cannot go to 0 in order to allow the vesicle to move to cell  $\bar{r}_-$ , so the continuation is blocked. If we go into cell  $r_-$  with a label  $p^+$  which is for another register  $r' \neq r$  or for the HALT-instruction, then the vesicle might continue its way

to cell  $\tilde{r}_-$  without applying a rule and then applying the rule  $a_r \rightarrow a_r^-$  in cell  $\tilde{r}_-$  thus reaching cell  $\bar{r}_-$ , but there the vesicle is blocked as no rule from  $R_{\bar{r}_-}$  then will be applicable, hence, no change of polarization allowing the vesicle to reach cell  $\hat{r}_-$  and then cell 0 again is possible.

**halting group** As soon as  $M$  has reached the *HALT*-label  $l_h$ , we may pass to cell  $l_h$  containing the rule  $l_h^+ \rightarrow \tilde{l}_h^+$  sending the rule to cell  $\tilde{l}_h$ ; there the rule  $\tilde{l}_h^+ \rightarrow \lambda$  has to be applied; the resulting vesicle then can go to the output cell  $\hat{l}_h$  to yield the terminal result of the computation. Moreover, the computation also will halt there.

In the way described above  $\Pi$  can simulate the computations of  $M$ . If the vesicle reaches the output cell  $\hat{l}_h$ , only terminal symbols from  $\{a_r \mid 1 \leq r \leq k\}$  are contained in its multiset which represents the  $k$ -dimensional vector computed by  $M$  by the number of symbols  $a_r$  for the number contained in register  $r$ , and, moreover, the computation halts, i.e., for any  $\beta \in \{(halt, term), halt, term\}$ , we obtain  $Ps(\Pi, sequ, \beta) = Ps(M)$ .

The first construction given above has the advantage that it can clearly be argued how the simulations work and why an erroneous computation never can be successful yielding a terminal result in cell  $\hat{l}_h$ . On the other hand, we can reduce the number of cells in a significant way with only needing a few additional arguments. The main idea is to collect cells needed for every  $r$  in just one cell, which yields the structure depicted in Figure 3.

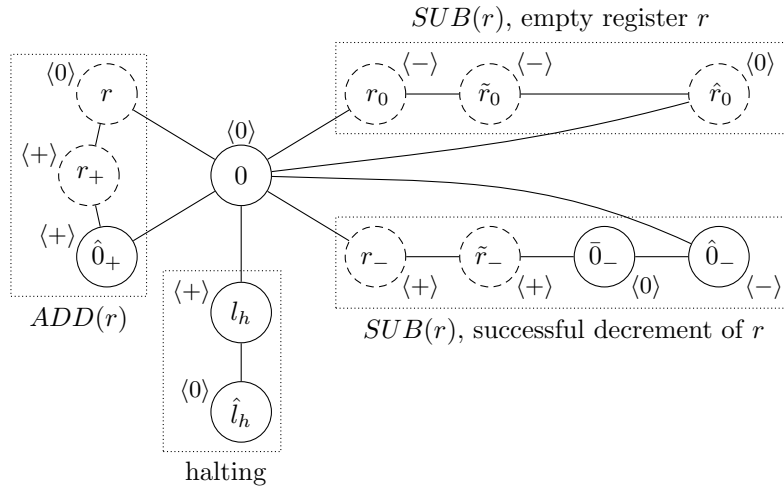


Fig. 3. The communication graph  $G'$  of the computationally complete uptPV system  $\Pi'$ . We also represent the polarizations of the nodes in angular brackets. Each node with a dashed contour is replicated for every register  $r$ .

20 *Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan*

The cells  $\hat{r}_+$  are replaced by the one cell  $\hat{0}_+$ ; all the cells  $\bar{r}_-$  and  $\hat{r}_-$  are replaced by the two cells  $\bar{0}_-$  and  $\hat{0}_-$ , respectively. All the information needed in the preceding cells about the register  $r$  is not needed any more in these new cells. Moreover, we can omit cell  $\tilde{l}_h$ : if the vesicle goes back to cell 0 instead of ending up in cell  $\hat{l}_h$ , the vesicle might move to cells  $r$  or  $\hat{r}_0$ , yet either no polarization changes (cells  $r$ ) are possible there or polarization changes ( $\hat{r}_0$ ) will lead the computation to get stuck. Summing up, the new system  $\Pi'$  is defined as follows:

$$\begin{aligned} \Pi' &= \left( L', V, T, R', (0, l_0), \hat{l}_h, \pi_L, \pi_V, \varphi_s, G' \right), \\ L' &= \{0, \hat{0}_+, \bar{0}_-, \hat{0}_-, l_h, \hat{l}_h\} \cup \{r, r_+, r_0, \bar{r}_0, \hat{r}_0, r_-, \bar{r}_- \mid 1 \leq r \leq m\}, \\ V &= \{a_r, a_r^-, a_r^+ \mid 1 \leq r \leq m\} \cup \{p, p^+, p^-, \bar{p}^+, \bar{p}^- \mid p \in B\}, \\ T &= \{a_r \mid 1 \leq r \leq k\}, \\ R' &= R_0 \cup R'_{l_h} \cup R'_{\hat{0}_+} \cup R'_{\bar{0}_-} \cup R'_{\hat{0}_-} \\ &\quad \cup \bigcup_{1 \leq r \leq m} (R_r \cup R_{r_+} \cup R_{r_-} \cup R_{\bar{r}_-} \cup R_{r_0} \cup R_{\bar{r}_0} \cup R_{\hat{r}_0}), \\ R'_{l_h} &= \{l_h^+ \rightarrow \lambda\}, \\ R'_{\hat{0}_+} &= \bigcup_{1 \leq r \leq m} R_{\hat{r}_+}, \\ R'_{\bar{0}_-} &= \bigcup_{1 \leq r \leq m} R_{\bar{r}_-}, \\ R'_{\hat{0}_-} &= \bigcup_{1 \leq r \leq m} R_{\hat{r}_-}. \end{aligned}$$

As already explained for the uptPV system  $\Pi$ , a computation in the register machine  $M$  yields a multiset represented as the numbers in the output registers as a result if and only if there is a halting computation in the uptPV system  $\Pi'$  yielding a terminal vesicle in cell  $\hat{l}_h$  containing this multiset. Hence, also for the new uptPV system  $\Pi'$  we get  $Ps(\Pi', sequ, \beta) = Ps(M)$  for any  $\beta \in \{(halt, term), halt, term\}$ .

At the end, we now describe the descriptonal complexity of the uptPV system  $\Pi'$  based on the constituents of the underlying register machine  $M$ : let  $r_{ADD}$  denote the number of registers on which only ADD-instructions are carried out and let  $r_{SUB}$  denote the number of registers for which also SUB-instructions are found in  $R$ ; moreover, let  $n_{ADD}$  denote the number of ADD-instructions and let  $n_{SUB}$  denote the number of SUB-instructions in  $R$ . Only counting the relevant constituents of  $\Pi'$ , we obtain the following for the number of cells and the number of rules in  $R'$ :

$$\begin{aligned} \text{number of cells in } \Pi' &= 2 \cdot r_{ADD} + 7 \cdot r_{SUB} + 6, \\ \text{number of rules in } R' &= 4 \cdot n_{ADD} + 4 \cdot n_{SUB} + 1 \cdot r_{ADD} + 5 \cdot r_{SUB} + 1. \end{aligned}$$

Finally, we also mention that the uptPV system  $\Pi'$  constructed for simulating the computations of the register machine  $M$  still fulfills its task for the more general

task when the register machine computes an output multiset for a given input multiset, which then has to be added in the initial vesicle. A special variant of a register machine computing an output multiset for a given input multiset is an *accepting* register machine where the simulating uptPV system  $\Pi'$  accepts the input multiset given in the initial vesicle by an empty vesicle appearing in the final cell  $\hat{l}_h$  (in fact, any final multiset could be allowed provided the computation halts with the vesicle appearing in the final cell  $\hat{l}_h$ ).  $\square$

## 6. Conclusion and Future Research

In this paper, we have investigated tissue P systems operating on vesicles of multisets with point mutations, i.e., with insertion, deletion, and substitution of single symbols, working either in one of the set maximally parallel derivation modes or in the sequential derivation mode. Without any additional control features, when using the sequential derivation mode, we obtain a characterization of the sets of (vectors of) natural numbers generated by partially blind register machines, whereas when using all three operations insertion, deletion, and substitution on the vesicle of multisets we can generate every recursively enumerable set of (vectors of) natural numbers. If we add the feature of elementary polarizations  $-1, 0, 1$  to the multisets and to the cells of the tissue P systems, even sequential tissue P systems are computationally complete.

An interesting topic for future research is to investigate the influence of the underlying communication structure on the generative power, especially in the case of polarized tissue P systems. Moreover, complexity issues like the number of polarizations and the number of cells remain to be investigated further in the future, for example, also with respect to find small universal devices, e.g., see [2].

We may also consider tissue P systems with more than one vesicle moving around, which, for example, offers the possibility to require the whole system to halt in order to obtain a result. Finally, using different evaluation functions may have an influence on the descriptive complexity of polarized tissue P systems.

## Acknowledgements

The authors gratefully thank the two referees for their useful comments.

## References

- [1] A. Alhazov, R. Freund, V. Rogozhin and Yu. Rogozhin, Computational completeness of complete, star-like, and linear hybrid networks of evolutionary processors with a small number of processors, *Natural Computing* **15**(1) (2016) 51–68.
- [2] A. Alhazov, R. Freund and S. Verlan, P systems working in maximal variants of the set derivation mode, *Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers*, (2016), pp. 83–102.
- [3] F. Arroyo, S. Canaval, V. Mitrana and Ş. Popescu, On the computational power of networks of polarized evolutionary processors, *Information and Computation* **253**(3) (2017) 371–380.

22 Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan

- [4] F. Arroyo, S. G. Canaval, V. Mitrana and Ş. Popescu, Networks of polarized evolutionary processors are computationally complete, *International Conference on Language and Automata Theory and Applications*, Springer (2014), pp. 101–112.
- [5] J. Castellanos, C. Martín-Vide, V. Mitrana and J. M. Sempere, Networks of evolutionary processors, *Acta informatica* **39**(6-7) (2003) 517–529.
- [6] J. Dassow and G. Păun, On the power of membrane computing, *J. UCS* **5**(2) (1999) 33–49.
- [7] R. Freund, O. Ibarra, Gh. Păun and H.-C. Yen, Matrix languages, register machines, vector addition systems, *Third Brainstorming Week on Membrane Computing* (2005) 155–167.
- [8] R. Freund, M. Kogler, Y. Rogozhin and S. Verlan, Graph-controlled insertion-deletion systems, *Proceedings Twelfth Annual Workshop on Descriptive Complexity of Formal Systems, DCFS 2010, Saskatoon, Canada, 8-10th August 2010.*, (2010), pp. 88–98.
- [9] R. Freund and M. Oswald, Tissue P systems and (mem)brane systems with mate and drip operations working on strings, *Electr. Notes Theor. Comput. Sci.* **171**(2) (2007) 105–115.
- [10] R. Freund and G. Păun, How to obtain computational completeness in P systems with one catalyst, *Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9-11, 2013.*, (2013), pp. 47–61.
- [11] R. Freund, V. Rogojin and S. Verlan, Computational completeness of networks of evolutionary processors with elementary polarizations and a small number of processors, *Proceedings DCFS 2017*, (Springer, 2017).
- [12] R. Freund, Yu. Rogozhin and S. Verlan, Generating and accepting P systems with minimal left and right insertion and deletion, *Natural Computing* **13**(2) (2014) 257–268.
- [13] C. Martín-Vide, J. Pazos, Gh. Păun and A. Rodríguez-Patón, A new class of symbolic abstract neural nets: Tissue P systems, *Computing and Combinatorics*, (Springer, 2002), pp. 290–299.
- [14] M. L. Minsky, *Computation. Finite and Infinite Machines* (Prentice Hall, Englewood Cliffs, NJ, 1967).
- [15] S. Popescu, Networks of polarized evolutionary processors with elementary polarization of symbols, *NCMA 2016*, (2016), pp. 275–285.
- [16] G. Păun, Computing with membranes, *J. Comput. Syst. Sci.* **61**(1) (2000) 108–143.
- [17] Gh. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook of Membrane Computing* (Oxford University Press, Oxford, England, 2010).
- [18] G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages* (Springer, 1997).
- [19] Bulletin of the International Membrane Computing Society (IMCS) <http://membranecomputing.net/IMCSBulletin/index.php>.
- [20] The P Systems Website <http://ppage.psystems.eu/>.