(Tissue) P Systems with Vesicles of Multisets

Artiom Alhazov
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
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
Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris Est – Créteil Val de Marne
61, av. Général de Gaulle, 94010, Créteil, France
sergiu.ivanov@u-pec.fr

TIMC-IMAG/DyCTiM, Faculty of Medicine of Grenoble,
5 avenue du Grand Sablon, 38700, La Tronche, France
sergiu.ivanov@univ-grenoble-alpes.fr

Sergey Verlan
Laboratoire d'Algorithmique, Complexité et Logique,
Université Paris Est Créteil,
61 av. du général de Gaulle, 94010 Créteil, 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 – -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 between 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*, 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].

E. Csuhaj-Varjú, P. Dömösi, Gy. Vaszil (Eds.): 15th International Conference on Automata and Formal Languages (AFL 2017)
EPTCS ??, 2017, pp. 11–25, doi:10.4204/EPTCS.??.6

For a state of the art snapshot 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 maximal derivation mode this means that the multiset of rules to be applied must be non-extendable, but all rules must indicate the same target cell. As we will show, with the set maximal 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 [3] and [4]), 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 the succeeding 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 λ; 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 Σ such that $x \in \Sigma^*$.

A *multiset* M with underlying set A is a pair (A,f) where $f : A \to \mathbb{N}$ 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 $supp(M) = \{a_1,\ldots,a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1,\ldots,a_k\}$ (the corresponding vector $f(a_1),\ldots a_, f(a_k)$ of natural numbers is called Parikh vector of the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$), and, moreover, all permutations of this string 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°.

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*. 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 $a \to b$ 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 λ; 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

Register machines are well-known universal devices for computing (generating or accepting) sets of vectors of natural numbers.

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 bijectively labeling the instructions in the set P,*
- *$l_0 \in B$ 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 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 $N(R)$. 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. 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 $p : (SUB(r), q, abort)$. Moreover, acceptance or generation now by definition also requires all registers, except the first k output registers, to be empty (which means all registers $k+1, ..., m$ 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 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 Π;*
- *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 $p \in Ins_V \cup Del_V \cup Sub_V$, 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.*

As in the case of NEPs and HNEPs, we call Π a *hybrid* tPV system if every cell is "specialized" in one type of evolution rules from (at most) one of the sets Ins_V, Del_V, and 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 non-extendable multiset 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.

In any case, the computation of Π 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 Π working in the derivation mode $\alpha \in \{sequ, smax\}$ 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 α and using the output strategy β 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. Moreover, if G can be interpreted as a tree, then we call the tPV system 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 the derivation mode *smax* and using all three types of point mutation rules computational completeness can even be obtained with PV systems:

Theorem 1 *$PsRE \subseteq Ps(PV, smax, \beta)$ for any $\beta \in \{(halt, term), halt, term\}$.*

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 Π generating K, i.e., $Ps(\Pi, smax, \beta) = K$:

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

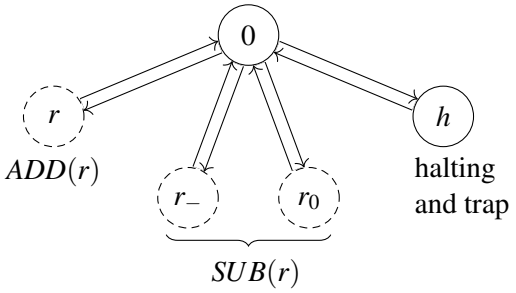

Figure 1: Communication structure of the two-level hierarchical PV system. Each node with a dashed contour is replicated for every register r.

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 $\{(0, p \to q, r), (0, p \to s, r), (r, \lambda \to a_r, 0)\}$; in cell r the number of symbols a_r representing

the contents of register r is incremented by the insertion rule $(r, \lambda \to a_r, 0)$, which also sends back the vesicle to the root cell.

$(SUB(r), q, s)$ is simulated by first choosing one of the rules from $\{(0, p \to s, r_0), (0, p \to q, r_-)\}$ 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 $(r_-, a_r \to \lambda, 0)$ 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 $(r_0, s \to s, 0)$ 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 $(r_0, a_r \to \#, 0)$ 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 Π, by applying the rule $(0, l_h \to \lambda, h)$, 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 Π, the multiset in this vesicle only contains terminal symbols and the computation in Π halts as well.

In sum, we conclude that $Ps(\Pi, smax, \beta) = K$ for any $\beta \in \{(halt, term), halt, term\}$. ■

The construction given in the preceding proof offers some additional nice features:

- The PV system Π 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 $(h, \# \to \#, 0), (0, \# \to \#, h)$, 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 must always leave the current cell whenever a rule can be applied.
- The number of cells in the PV system Π 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 2 *$PsPBRM \subseteq Ps(PV, sequ, \beta)$ for any $\beta \in \{(halt, term), halt, term\}$.*

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 Π generating K in the

sequential derivation mode, i.e., $Ps(\Pi, sequ, \beta) = K$:

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

The simulation of the computations in M by Π 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 Π, by applying the rule $(0, l_h \to \lambda, h)$, 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 Π will halt, but otherwise the trap symbol # will be introduced by (one of) the rules from $\{(h, a_r \to \#, 0) \mid k+1 \leq r \leq m\}$, thus causing an infinite loop.

In sum, we conclude that $Ps(\Pi, sequ, \beta) = K$ for any $\beta \in \{(halt, term), halt, term\}$. ■

The following corollary is immediate consequence of Theorem 1 proved above:

Corollary 3 *$PsRE = Ps(PV, smax, \beta) = Ps(tPV, smax, \beta)$ for any $\beta \in \{(halt, term), halt, term\}$.*

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. ■

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

Proof. (*Sketch*) 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 $L = \{i \mid 1 \leq i \leq l\}$.

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 generation of w_0 in l_0 by a sequence of *ADD*-instructions finally ending up with label l_0 and the correct values in registers r for the numbers of symbols a_r in cell l_0.

We now sketch how the rules of Π can be simulated by register machine instructions in M:

$(i, \lambda \to b, j)$ is simulated by $i : (ADD(b), j)$.

$(i, a \to \lambda, j)$ is simulated by $i : (SUB(a), j)$.

$(i, a \to b, j)$ is simulated by the sequence of two instructions $i : (SUB(a), i')$ and $i' : (ADD(b), j)$ using an intermediate label i'.

Hence, for these simulations we may need $2l$ labels in the sense explained above. 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

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

We observe that $\tilde{h}, \hat{h}$ are labels different from h'. Since $\hat{h}$ is now 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 Π producing a multiset consisting exclusively of terminal symbols.

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

As a consequence of Theorems 2 and 4 we obtain:

Corollary 5 $PsPBRM = Ps(PV, sequ, term)$.

5 Polarized tissue P systems with vesicles of multisets

In a polarized tissue P system Π 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 3 *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 Π;*

- *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 $p \in Ins_V \cup Del_V \cup Sub_V$, 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;*
- *π_L is the function assigning an integer polarization to each cell (as already mentioned above, we here restrict ourselves to the elementary polarizations from $\{-1,0,1\}$);*
- *π_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 $\{-1,0,1\}$);*
- *φ is the evaluation function yielding an integer value for each multiset.*

As in the case of NEPs and HNEPs, we call Π a *hybrid* ptPV system if a cell is "specialized" in one type of evolution rules from (at most) one of the sets Ins_V, Del_V, and 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 mode *smax*. 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 (caution: we allow the set of applied rules to be empty) 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 evaluation function φ which computes the value of a multiset as the sum of the values of the symbols contained in it; we write φ_s for this function.

In any case, the computation of Π starts with a vesicle containing the multiset w_0 in cell i_0 (obviously, the initial multiset w_0 has to 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 Π working in the derivation mode $\alpha \in \{sequ, smax\}$, using the evaluation function φ_s and 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 ptPV systems with at most n cells working in the derivation mode α and using the output strategy β are denoted by $Ps(ptPV_n, \alpha, \beta)$ ($Ps^k(ptPV_n, \alpha, \beta)$) and $N(ptPV_n, \alpha, \beta)$, respectively. If n is not bounded, we simply omit the subscript in these notations.

We should like to mention that again the 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
- $E = \{(i,j) \mid (i,p,j) \in R\}$.

In general, we do not forbid G to have loops. Moreover, if G can be interpreted as a tree, then we call the ptPV system Π 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*.

Moreover, there is another variant of interpreting the functioning of the ptPV Π if 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. Yet as G is an *undirected* graph this makes a big difference as communication now by default is bidirectional, i.e., we cannot enforce the direction of the movement of the vesicle any more. According to these explanations it becomes obvious that this variant is a special case of ptPV systems. In fact, in this variant, if (i, p, j) is a rule in R, then also (j, p, i) must be a rule in R. As a special variant of ptPV systems, we then call it a uptPV system (with u specifying that the communication structure is an undirected graph).

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

Theorem 6 $PsRE \subseteq Ps(uptPV_n, sequ, term)$.

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 Π generating the same set of multisets as M, i.e., $Ps(\Pi, sequ, term) = Ps(M)$.

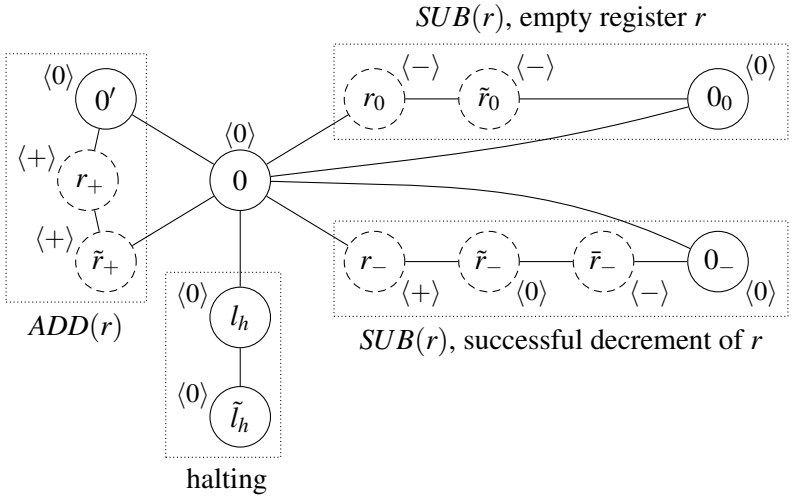

Figure 2: The communication graph G of the computationally complete uptPV system. We also represent the polarizations of the nodes in angular brackets. Each node with a dashed contour is replicated for every register r.

$$\Pi = \left(L, V, T, R, (0, l_0), \tilde{l}_h, \pi_L, \pi_V, \varphi_s, G\right),$$
$$L = \{0, 0', 0_0, 0_-, l_h, \tilde{l}_h\} \cup \{r_+, r_0, r_-, \tilde{r}_+, \tilde{r}_0, \tilde{r}_-, \hat{r}_- \mid 1 \leq r \leq m\},$$
$$V = \{a_r, a_r{}^-, a_r{}^+ \mid 1 \leq r \leq m\} \cup \{p, p^+, p^- \mid p \in B\},$$
$$T = \{a_r \mid 1 \leq r \leq k\}.$$

The evaluation π_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 π_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:

root cell 0 All simulations start from cell 0 and again end there.

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

increment group Any ADD-instruction $p : (ADD(r), q, s)$ is simulated by passing from cell 0 to $0'$, from where only the correct path through r_+ and then $\tilde{r}_+$ for the suitable r will lead back to cell 0.

$R_{0'} = \{p \to p^+ \mid p : (ADD(r), q, s) \in P\}$

$R_{r^+} = \{\lambda \to a_r\}$ In order to guarantee that the rule $\lambda \to a_r$ is applied only once, we need the condition that after the application of a rule the vesicle has to leave the cell, which here means to pass to cell $\tilde{r}^+$ where the polarization is changed so that the vesicle will not be able to immediately return to cell r^+.

$R_{\tilde{r}^+} = \{p^+ \to q, p^+ \to s \mid p : (ADD(r), q, s) \in P\}$

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

zero check $R_{r_0} = \{a_r \to a_r{}^+\}$. Cell 0 sends the vesicle to r_0 by non-deterministically applying the rule $p \to p^-$ and thus setting the polarization of the multiset to -1. If the rule $a_r \to a_r{}^+$ is applicable, then the polarization goes back to 0 and therefore the correct continuation in cell $\tilde{r}_0$ is blocked. On the other hand, when the vesicle returns back to cell 0, no rule can be applied there, and then moving to cell $0'$ or cell l_h also does not yield a successful continuation.

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

Cell 0_0 is needed for blocking the way from cell 0 to cell $\tilde{r}_0$.

The rule set R_{0_0} can be taken to be empty. If for formal reasons one would not like to have such a situation where a vesicle can pass through a cell without undergoing an evolution rule, we could take:

$R_{0_0} = \{s \to s \mid s \in B\}$

decrement Passing the sequence of cells 0–r_-–$\tilde{r}_-$–$\bar{r}_-$–0_- allows for decrementing the number of symbols a_r. Cell 0 sends the vesicle to r_- by non-deterministically applying the rule $p \to p^+$ and by setting the polarization of the multiset to $+1$.

$R_{r_-} = \{a_r \to a_r{}^-\}$ After the application of the rule $a_r \to a_r{}^-$ the polarization is again 0, so the vesicle might also go back to cell 0, but all possible continuations from there finally get blocked with the p^+ in there for a label p of a SUB-instruction when moving into the increment group.

$R_{\tilde{r}_-} = \{p^+ \to q\}$ With the application of the rule $p^+ \to q$ the polarization changes; if the wrong r-branch has been chosen from cell 0, the computation gets stuck here.

$R_{\tilde{r}_-} = \{a_r{}^- \to \lambda\}$

As in the zero-check group, the set R_{0_0} can be chosen to be empty or we take:

$R_{0_0} = \{s \to s \mid s \in B\}$

halting group As soon as M has reached the $HALT$-label l_h, we may pass to cell l_h containing the rule $l_h \to \lambda$; the resulting vesicle then can go to the output cell $\tilde{l}_h$ to yield the result of the computation.

In the way described above Π can simulate the computations of M. If the vesicle reaches the output cell $\tilde{l}_h$, 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. ■

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 the maximally parallel set derivation mode 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 vesicles 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.

Besides the maximally parallel set derivation mode, also the other set derivation modes (see [2]) promise to yield similar results. Another topic 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 cells remain to be investigated 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 descriptional complexity of polarized tissue P systems.

References

[1] Artiom Alhazov, Rudolf Freund, Vladimir Rogozhin & Yurii Rogozhin (2016): *Computational completeness of complete, star-like, and linear hybrid networks of evolutionary processors with a small number of processors*. Natural Computing 15(1), pp. 51–68, doi:10.1007/s11047-015-9534-1.

[2] Artiom Alhazov, Rudolf Freund & Sergey Verlan (2016): *P Systems Working in Maximal Variants of the Set Derivation Mode*. In: Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers, pp. 83–102, doi:10.1007/978-3-319-54072-6_6.

[3] Fernando Arroyo, Sandra Gómez Canaval, Victor Mitrana & Ştefan Popescu (2014): *Networks of polarized evolutionary processors are computationally complete*. In: International Conference on Language and Automata Theory and Applications, Springer, pp. 101–112, doi:10.1007/978-3-319-04921-2_8.

[4] Fernando Arroyo, Sandra Gómez Canaval, Victor Mitrana & Ştefan Popescu (2017): *On the computational power of networks of polarized evolutionary processors*. Information and Computation 253(3), pp. 371–380, doi:10.1016/j.ic.2016.06.004.

[5] Juan Castellanos, Carlos Martín-Vide, Victor Mitrana & José M Sempere (2003): *Networks of evolutionary processors*. Acta informatica 39(6-7), pp. 517–529, doi:10.1007/s00236-004-0158-7.

[6] Jürgen Dassow & Gheorghe Păun (1999): *On the Power of Membrane Computing*. J. UCS 5(2), pp. 33–49, doi:10.3217/jucs-005-02-0033.

[7] Rudolf Freund, Oscar Ibarra, Gheorghe Păun & Hsu-Chun Yen (2005): *Matrix Languages, Register Machines, Vector Addition Systems*. Third Brainstorming Week on Membrane Computing, pp. 155–167.

[8] Rudolf Freund, Marian Kogler, Yurii Rogozhin & Sergey Verlan (2010): *Graph-Controlled Insertion-Deletion Systems*. In: Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS 2010, Saskatoon, Canada, 8-10th August 2010., pp. 88–98, doi:10.4204/EPTCS.31.11.

[9] Rudolf Freund & Marion Oswald (2007): *Tissue P Systems and (Mem)Brane Systems with Mate and Drip Operations Working on Strings*. Electronic Notes in Theoretical Computer Science 171(2), pp. 105–115, doi:10.1016/j.entcs.2007.05.011.

[10] Rudolf Freund & Gheorghe Păun (2013): *How to Obtain Computational Completeness in P Systems with One Catalyst*. In: Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9-11, 2013., pp. 47–61, doi:10.4204/EPTCS.128.13.

[11] Rudolf Freund, Vladimir Rogojin & Sergey Verlan (2017): *Computational Completeness of Networks of Evolutionary Processors with Elementary Polarizations and a Small Number of Processors*. In Giovanni Pighizzini & Cezar Câmpeanu, editors: Descriptional Complexity of Formal Systems: 19th IFIP WG 1.02 International Conference, DCFS 2017, Milano, Italy, July 3-5, 2017, Proceedings, Springer, pp. 140–151, doi:10.1007/978-3-319-60252-3_11.

[12] Rudolf Freund, Yurii Rogozhin & Sergey Verlan (2014): *Generating and accepting P systems with minimal left and right insertion and deletion*. Natural Computing 13(2), pp. 257–268, doi:10.1007/s11047-013-9396-3.

[13] Carlos Martín-Vide, Juan Pazos, Gheorghe Păun & Alfonso Rodríguez-Patón (2002): *A new class of symbolic abstract neural nets: Tissue P systems*. In: Computing and Combinatorics, Springer, pp. 290–299, doi:10.1007/3-540-45655-4_32.

[14] Marvin L. Minsky (1967): *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ.

[15] Ştefan Popescu (2016): *Networks of polarized evolutionary processors with elementary polarization of symbols*. In: NCMA 2016, pp. 275–285.

[16] Gheorghe Păun (2000): *Computing with Membranes*. Journal of Computer and System Sciences 61(1), pp. 108–143, doi:10.1006/jcss.1999.1693.

[17] Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa, editors (2010): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Oxford, England.

[18] Grzegorz Rozenberg & Arto Salomaa, editors (1997): *Handbook of Formal Languages*. 1-3, Springer.

[19] *Bulletin of the International Membrane Computing Society (IMCS)*. `http://membranecomputing.net/IMCSBulletin/index.php`.

[20] *The P Systems Website*. `http://ppage.psystems.eu/`.