Algebra Meets Biology

Stefan Schuster

Matthias Schleiden Institute, Department of Bioinformatics,
Friedrich Schiller University, Ernst-Abbe-Platz 2, 07743 Jena, Germany
stefan.schu@uni-jena.de

Abstract. Fibonacci numbers and the Golden Section occur in many instances in Biology. We have recently found that the potential number of fatty acids increases with their chain length according to the famous Fibonacci series, when *cis/trans* isomerism is neglected. Since the ratio of two consecutive Fibonacci numbers tends to the Golden section, 1.618..., organisms can increase fatty acid variability approximately by that factor per carbon atom invested. Moreover, we show that, under consideration of *cis/trans* isomerism, modification by hydroxy and/or oxo groups, triple bonds or adjacent double bonds, diversity can be described by generalized Fibonacci numbers (e.g. Pell numbers). Similar calculations can be applied to aliphatic amino acids. Our results should be of interest for mass spectrometry, combinatorial chemistry, synthetic biology, patent applications, use of fatty acids as biomarkers and the theory of evolution.

A second example of the role of algebra in biology discussed in this talk concerns intracellular calcium oscillations. Such oscillations are transformed (in a sense, decoded) in the cell by phosphorylation of proteins. In this way, an approximate temporal integral of the signal is computed. This implies that the number of spikes in the oscillation can be counted. In plant cells, an effect is often triggered only if a certain number of spikes (e.g., five) occurred. Some techniques for the mathematical modelling of such phenomena are reviewed here.

References

1. Schuster, S., Fichtner, M., Sasso, S.: Use of Fibonacci numbers in lipidomics – Enumerating various classes of fatty acids. Sci. Rep. **7**, 39821 (2017)
2. Fichtner, M., Voigt, K., Schuster, S.: The tip and hidden part of the iceberg: proteinogenic and non-proteinogenic aliphatic amino acids. Biochim. Biophys. Acta - Gen. Subj. **1861**, 3258–3269 (2017)
3. Bodenstein, C., Knoke, B., Marhl, M., Perc, M., Schuster, S.: Using Jensen's inequality to explain the role of regular calcium oscillations in protein activation. Phys. Biol. **7**, 036009 (2010)

Contents

Tissue P Systems with Point Mutation Rules

Artiom Alhazov[1], Rudolf Freund[2(✉)], Sergiu Ivanov[3], and Sergey Verlan[4]

[1] Institute of Mathematics and Computer Science,
Academiei 5, 2028 Chişinău, Moldova
artiom@math.md

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

[3] IBISC, Université Évry, Université Paris-Saclay,
23 Boulevard de France, 91025 Évry, France
sergiu.ivanov@univ-evry.fr

[4] Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est Créteil,
61 Avenue du Général de Gaulle, 94010 Créteil, France
verlan@u-pec.fr

Abstract. We consider tissue P systems working in the sequential mode on vesicles of multisets with the very simple operations of insertion, deletion, and substitution of single objects. In a computation step, one rule is to be applied if possible, and then, in any case, the whole multiset being enclosed in a vesicle moves to one of the cells as indicated by the underlying graph structure of the system. The target cell is chosen in a non-deterministic way and does not depend on the possibly applied rule. With defining halting as reaching the final cell with a vesicle only containing terminal symbols, computational completeness can be obtained. Imposing the restriction that in each derivation step one rule has to be applied, we only reach the computational power of matrix grammars for multisets. Moreover, we also discuss variants for computations on strings. Finally, we outline a way how to "go beyond Turing" like with red-green register machines.

1 Introduction

Membrane systems were introduced at the end of the last century by Păun, e.g., see [11] and [32], 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*, 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 [28].

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 variants of (tissue) P systems and their expressive power is given in the handbook which appeared in 2010, see [33]. For a short view on the state of the art of the domain, we refer the reader to the P systems website [37] as well as to the Bulletin series of the International Membrane Computing Society [36].

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 [20], and P systems using these operations at the left or right end of string objects were introduced in [25], 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 can 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 [23].

In this paper we follow a different approach which has been introduced in [3]: in order 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 if a rule has been applied or not. Running the tissue P system in the sequential mode we obtain computational completeness when allowing even no rule to be applied in a derivation step, i.e., if none of the rules assigned to the cell where the vesicle currently is to be found can be applied to the multiset contained in the vesicle. Requiring that one rule has to be applied in every derivation step, 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 [21]. Yet in that case, always two vesicles (one of them possibly being an axiom available in an unbounded number) have to interact. In this paper, the rules are always applied

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 [9] 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 a string. 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 [4].

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 [7] and [6], and networks of evolutionary processors only using the elementary polarizations $-1, 0, 1$ were investigated in [31]. The number of processors (cells) needed to obtain computational completeness has been improved in a considerable way in [24] making these results already comparable with those obtained in [4] 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. The tissue P systems considered in this paper now take features of several of the devices mentioned above: we use a set of cells, each of them having assigned a set of point mutation rules as in (H)NEPs as well as two substeps in each derivation step – an *evolution step* in which one rule out of the set of rules assigned to the cell is applied (i.e., if possible) to the multiset in the vesicle currently present in the cell, and a *communication step* in which the vesicle is sent from the current cell to another cell.

In contrast to (H)NEPs, which usually deal with strings, we here deal with multisets enclosed in a vesicle. Yet the most important difference is that the communication is not guided by output and input filters or by polarizations,

but simply by using a directed graph as the underlying communication structure. On the other hand, in contrast to the well-known control mechanism of graph control with appearance checking (e.g., see [10] and [19]), the target cell does not depend on which rule – if any – has been applied, which resembles the special variant of graph control known as unconditional transfer, for example, see [12].

Another control mechanism closely related to this moving a vesicle through the underling communication structure of a tissue P system is using regular control languages (e.g., see [10]): the directed graph describing the control structure can be interpreted as the graph of a finite automaton for the rule labels; in this model, appearance checking, i.e., allowing some rules to be skipped if they cannot be applied, is modeled by fixing a set of rules which may be skipped when following a control word in the regular control language. An important technical detail to be mentioned here is that in regular control languages the control word usually is composed of single rule labels, whereas in the tissue P systems introduced in this paper the control word would be composed of sets of rule labels.

For comparing the results exhibited in this paper by using the control given by the underlying communication structure of the tissue P system, we recall the general framework for grammars working in the sequential derivation mode, developed in [19]; many relations between various regulating mechanisms can be established in a very general setting without any reference to the underlying objects the rules are working on, as, for example, for graph-controlled, programmed, matrix, random-context, and ordered grammars. Other control mechanisms considered in this paper are grammars with regular control languages and time-varying grammars, which in the context of P systems have been investigated in [2]. In that paper, computational completeness is shown for membrane systems with only one membrane, but using non-cooperative rules and a lot of additional control symbols. Moreover, the number of steps needed without applying a rule in the proof given there was two, whereas our result established in this paper only needs one step without applying a rule.

Finally, we should like to mention that the control given by the underlying communication structure of the tissue P system could also be interpreted as having a P system with only one membrane but using states instead; for a discussion on how to use and interpret features of (tissue) P systems as states we refer to [2], where also an example only using the point mutation rules insertion and deletion is given.

Although dealing with multisets, we can also simulate computations on strings: adding the rules for taking an object from the environment into the vesicle and/or sending an object out of the vesicle allows us to define the input and output strings as the sequence of symbols taken in and sent out, respectively. As an input sequence can be encoded as a number and then processed by only two registers (the folklore result for two-counter automata) and the final result can be encoded as a number and then decoded into a sequence of symbols sent out (e.g., see [13]), the basic result showing how to simulate register machines also allows for simulating computations on strings.

Various possibilities of how one may "go beyond Turing" are discussed in [27]; for example, the definitions and results for red-green Turing machines can be found there. In [5] the notion of red-green automata for register machines with input strings given on an input tape (often also called *counter automata*) is introduced and the concept of *red-green P automata* for several specific models of membrane systems is explained. Via red-green counter automata, the results for acceptance and recognizability of finite strings by red-green Turing machines are carried over to red-green P automata. The basic idea of red-green automata is to distinguish between two different sets of states (red and green states) and to consider infinite runs of the automaton on finite input objects (strings, multisets); allowed to change between red and green states more than once, red-green automata can recognize more than the recursively enumerable sets (of strings, multisets), i.e., in that way one can "go beyond Turing". In the area of P systems, first attempts to do that can be found in [8] and [35]. Computations with infinite words by P automata were investigated in [22].

In [17,18], infinite runs of P automata are considered, taking into account the existence/non-existence of a recursive feature of the current sequence of configurations. In that way, infinite sequences over $\{0,1\}$, called "observer languages", are obtained, where 1 indicates that the specific feature is fulfilled by the current configuration and 0 indicates that this specific feature is not fulfilled. The recognizing runs of red-green automata then correspond to having ω-regular languages over $\{0,1\}$ of a specific form ending with 1^ω as observer languages.

The main problem with finding a red-green variant of the special variant of tissue P systems introduced in this paper is its inherent non-determinism even when simulating deterministic register machines and their red-green variant (again see [5]) – the zero-test case and the decrement case of a SUB-instruction are chosen in a non-deterministic way. Yet there is a possibility to overcome this problem by using ideas as, for example, discussed in [30] and in [1]. In [30], k-determinism of variants of P systems like (purely) catalytic P systems is discussed, i.e., with a look-ahead of (at most) k derivation steps always a deterministic continuation of a derivation can be found, although at some moment several (multisets of) rules could be applied, because all other derivation paths would lead to the introduction of the trap symbol; this technique is used very often in proofs in the area of membrane systems to cause non-halting computations. In [1], the concept of toxic objects is introduced which "kill" any derivation branch producing a toxic object as, for example, the trap symbol. As we will show later, our proof for simulating the SUB-instructions of a register machine only needs a look-ahead of 1, i.e., as the deterministic continuation of a derivation in the tissue P system we can just take the one which does not introduce the trap symbol. In that way, also a red-green variant of the model introduced in this paper can be defined, thus even allowing us to "go beyond Turing".

The rest of the paper is structured as follows: In Sect. 2 we recall some well-known definitions from formal language theory, and in the Sect. 3 we describe the general model for sequential grammars as established in [19] and the control mechanisms we are referring to in this paper within this general framework.

In Sect. 4 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. In Sect. 5 we show our main results for tissue P systems with vesicles of multisets using the three operations – insertion, deletion, and substitution – in the sequential derivation mode and moving the vesicle according to the underlying communication structure of the system. Computational completeness can be achieved when allowing also no rule to be applied in a derivation step, whereas otherwise 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 Sect. 6 we briefly discuss how computations on strings can be simulated with this model of tissue P systems. Moreover, we briefly outline how we even can "go beyond Turing" with such systems by simulating red-green register machines in Sect. 7. 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 $(f(a_1), \ldots, f(a_k))$ 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) and of number sets is denoted by $PsRE$ and NRE, respectively. For more details of formal language theory the reader is referred to the monographs and handbooks in this area, such as [34].

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 and we also write $I(b)$. 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 rule 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. 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

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 [29]) 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 are needed to generate any set from the family $PsRE$).

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; hence, 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 $k + 1, \ldots, m$ 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 [16]) 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 A General Model for Sequential Grammars

We first recall the main definitions of the general model for sequential grammars as established in [19].

A *(sequential) grammar* G is a construct $(O, O_T, w, P, \Longrightarrow_G)$ where

- O is a set of *objects*;
- $O_T \subseteq O$ is a set of *terminal objects*;
- $w \in O$ is the *axiom (start object)*;
- P is a finite set of *rules*;
- $\Longrightarrow_G \subseteq O \times O$ is the *derivation relation* of G.

We assume that each of the rules $p \in P$ induces a relation $\Longrightarrow_p \subseteq O \times O$, in sum yielding $\Longrightarrow_G = \bigcup_{p \in P} \Longrightarrow_p$. The reflexive and transitive closure of $\Longrightarrow_G$ is denoted by $\overset{*}{\Longrightarrow}_G$. A rule $p \in P$ is called *applicable* to an object $x \in O$ if and only if there exists at least one object $y \in O$ such that $(x, y) \in \Longrightarrow_p$; we also write $x \Longrightarrow_p y$.

In the following we shall consider different types of grammars depending on the components of G (where the set of objects O is infinite, e.g., V^*, the set of strings over the alphabet V), especially with respect to different types of rules (e.g., context-free string rules). Some specific conditions on the elements of G, especially on the rules in P, may define a special type X of grammars which then will be called *grammars of type* X.

The *language generated by* G is the set of all terminal objects (we also assume $v \in O_T$ to be decidable for every $v \in O$) derivable from the axiom, i.e.,

$$L(G) = \left\{ v \in O_T \mid w \overset{*}{\Longrightarrow}_G v \right\}.$$

The family of languages generated by grammars of type X is denoted by $\mathcal{L}(X)$.

3.1 Specific Types of Objects

As special types of objects, in this paper we consider strings and multisets.

String Grammars. In the general notion as defined above, a *string grammar* G_S is represented as

$$\left((N \cup T)^*, T^*, w, P, \Longrightarrow_{G_S}\right)$$

where N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, $w \in (N \cup T)^+$, P is a finite set of *rules* of the form $u \to v$ with $u \in V^*$ (for generating grammars, $u \in V^+$) and $v \in V^*$ (for accepting grammars, $v \in V^+$), with $V := N \cup T$; the derivation relation for $u \to v \in P$ is defined by $xuy \Longrightarrow_{u \to v} xvy$ for all $x, y \in V^*$, thus yielding the well-known derivation relation $\Longrightarrow_{G_S}$ for the string grammar G_S. In the following, we shall also use the common notation $G_S = (N, T, w, P)$ instead, too. We remark that, usually, the axiom w is supposed to be a non-terminal symbol, i.e., $w \in V \setminus T$, and is called the *start symbol*.

As special types of string grammars we consider string grammars with *arbitrary* rules as well as with *context-free* of the form $A \to v$, with $A \in N$ and $v \in V^*$. The corresponding types of grammars are denoted by ARB and CF, thus yielding the families of languages $\mathcal{L}(ARB)$, i.e., the family of recursively enumerable languages (also denoted by RE), as well as $\mathcal{L}(CF)$, i.e., the family of context-free languages, respectively.

We refer to [19] where some examples for string grammars of specific types illustrating the expressive power of this general framework are given.

The subfamily of REG only consisting of 1-star languages of the form W^* for some finite set of strings W is denoted by REG^{1*}; to be more specific, we also consider $REG^{1*}(k,p)$ consisting of all 1-star languages of the form W^* with k being the maximum number of strings in W and p being the maximum lengths of the strings in W. If $W = \{w\}$ for a singleton w, we call the set $\{w\}^*$ *periodic* and $|w|$ its *period*; thus, $REG^{1*}(1,p)$ denotes the family of all periodic sets with period at most p. If any of the numbers k or p may be arbitrarily large, we replace it by $*$.

Multiset Grammars. $G_m = \left((N \cup T)^\circ, T^\circ, w, P, \Longrightarrow_{G_m}\right)$ is called a *multiset grammar*; N is the alphabet of *non-terminal symbols*, T is the alphabet of *terminal symbols*, $N \cap T = \emptyset$, w is a non-empty multiset over V, $V := N \cup T$, and P is a (finite) set of multiset rules yielding a derivation relation $\Longrightarrow_{G_m}$ on the multisets over V; the application of the rule $u \to v$ to a multiset x has the effect of replacing the multiset u contained in x by the multiset v. For the multiset grammar G_m we also write $(N, T, w, P, \Longrightarrow_{G_m})$.

As special types of multiset grammars we consider multiset grammars with *arbitrary* rules as well as *context-free* (*non-cooperative*) rules of the form $A \to v$, with $A \in N$ and $v \in V^\circ$; the corresponding types X of multiset grammars are denoted by $mARB$ and mCF, thus yielding the families of multiset languages $\mathcal{L}(X)$.

As is well known, for example, see [26], even with arbitrary multiset rules, it is not possible to get $Ps(\mathcal{L}(ARB))$:

$$\mathcal{L}(mCF) = Ps(\mathcal{L}(CF)) \subsetneqq \mathcal{L}(mARB) \subsetneqq Ps(\mathcal{L}(ARB)).$$

3.2 Graph-Controlled and Programmed Grammars

A *graph-controlled grammar* (with applicability checking) of type X is a construct

$$G_{GC} = (G, g, H_i, H_f, \Longrightarrow_{GC})$$

where $G = (O, O_T, w, P, \Longrightarrow_G)$ is a grammar of type X; $g = (H, E, K)$ is a labeled graph where H is the set of node labels identifying the nodes of the graph in a one-to-one manner, $E \subseteq H \times \{Y, N\} \times H$ is the set of edges labeled by Y or N, $K : H \to 2^P$ is a function assigning a subset of P to each node of g; $H_i \subseteq H$ is the set of initial labels, and $H_f \subseteq H$ is the set of final labels. The derivation relation $\Longrightarrow_{GC}$ is defined based on $\Longrightarrow_G$ and the control graph g as follows: For any $i, j \in H$ and any $u, v \in O$, $(u,i) \Longrightarrow_{GC} (v,j)$ if and only if

- $u \Longrightarrow_p v$ by some rule $p \in K(i)$ and $(i, Y, j) \in E$ *(success case)*, **or**
- $u = v$, no $p \in K(i)$ is applicable to u, and $(i, N, j) \in E$ *(failure case)*.

The language generated by G_{GC} is defined by

$$L(G_{GC}) = \left\{v \in O_T \mid (w,i) \Longrightarrow^*_{G_{GC}} (v,j),\ i \in H_i, j \in H_f\right\}.$$

If $H_i = H_f = H$, then G_{GC} is called a *programmed grammar*. The families of languages generated by graph-controlled and programmed grammars of type X are denoted by $\mathcal{L}(X\text{-}GC_{ac})$ and $\mathcal{L}(X\text{-}P_{ac})$, respectively. If the set E contains no edges of the form (i, N, j), then the graph-controlled or programmed grammar is said to be *without applicability checking*; the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}GC)$ and $\mathcal{L}(X\text{-}P)$, respectively. If for all pairs $(i,j) \in H \times H$ we have $(i, Y, j) \in E$ if and only if $(i, N, j) \in E$, then this regulated grammar is said to be with *unconditional transfer*, as the transitions in the control graph do not depend on the application of a rule; the corresponding families of languages are denoted by $\mathcal{L}(X\text{-}GC_{ut})$ and $\mathcal{L}(X\text{-}P_{ut})$, respectively.

The notions and concepts *with/without applicability checking* were introduced as *with/without appearance checking* in the original definition for string grammars because the appearance of the non-terminal symbol on the left-hand side of a context-free rule was checked, which coincides with checking for the applicability of this rule in our general model; in both cases – applicability checking and appearance checking – we can use the abbreviation *ac*.

The concept of unconditional transfer has only been investigated in a few papers; the most interesting results are to be found in [12], where the following was shown for strings:

Theorem 1. $\mathcal{L}(CF\text{-}GC_{ut}) = \mathcal{L}(CF\text{-}GC_{ac}) = RE$.

3.3 Matrix Grammars

A *matrix grammar* (with applicability checking) of type X is a construct

$$G_M = (G, M, F, \Longrightarrow_{G_M})$$

where $G = (O, O_T, w, P, \Longrightarrow_G)$ is a grammar of type X, M is a finite set of sequences of the form $(p_1, \ldots, p_n)$, $n \geq 1$, of rules in P, and $F \subseteq P$. For $w, z \in O$ we write $w \Longrightarrow_{G_M} z$ if there are a matrix $(p_1, \ldots, p_n)$ in M and objects $w_i \in O$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either

- $w_i \Longrightarrow_G w_{i+1}$ or
- $w_i = w_{i+1}$, p_i is not applicable to w_i, and $p_i \in F$.

$L(G_M) = \left\{v \in O_T \mid w \Longrightarrow^*_{G_M} v\right\}$ is the language generated by G_M. The family of languages generated by matrix grammars of type X is denoted by $\mathcal{L}(X\text{-}MAT_{ac})$. If the set F is empty, then the grammar is said to be *without applicability checking*; the corresponding family of languages is denoted by $\mathcal{L}(X\text{-}MAT)$.

We mention that in this paper we choose the definition where the sequential application of the rules of the final matrix may stop at any moment.

3.4 Grammars with Regular Control and Time-Varying Grammars

Another possibility to capture the idea of controlling the derivation in a grammar as with a control graph is to consider the sequence of rules applied during a

A *grammar with regular control and appearance checking* is a construct

$$G_C = (G, H_C, L, F)$$

where $G = (O, O_T, w, P, \Longrightarrow_G)$ is a grammar of type X and L is a regular language over H_C, where H_C is the set of labels identifying the non-empty subsets of productions from P in a one-to-one manner, and $F \subseteq H_C$. We will use the notation $H_C(P')$ to refer to the label of the subset $P' \subseteq P$. The language generated by G_C consists of all terminal objects z such that there exist a string $H_C(P_1) \dots H_C(P_n) \in L$ as well as objects $w_i \in O$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and, for all $1 \leq i \leq n$, either

- $w_i \Longrightarrow_G w_{i+1}$ by some production from P_i or
- $w_i = w_{i+1}$, no production from P_i is applicable to w_i, and $H_C(P_i) \in F$.

It is rather easy to see that the model of grammars with regular control is closely related with the model of graph-controlled grammars in the sense that the control graph corresponds to the deterministic finite automaton accepting L. Hence, we may also speak of a *grammar with regular control and without appearance checking* if $F = \emptyset$, and if $F = H_C$ then G_C is said to be a *grammar with regular control and unconditional transfer*. The corresponding families of languages are denoted by $\mathcal{L}(X\text{-}C(REG)_{ac})$, $\mathcal{L}(X\text{-}C(REG))$, and $\mathcal{L}(X\text{-}C(REG)_{ut})$.

Obviously, the control languages can also be taken from another family of languages Y, e.g., $\mathcal{L}(CF)$, thus yielding the families $\mathcal{L}(X\text{-}C(Y)_{ac})$, etc., but in this paper we shall restrict ourselves to the cases $Y = REG$ and $Y = REG^{1*}(k,p)$. For $Y = REG^{1*}(1,p)$, these grammars are also known as *(periodically) time-varying grammars*, as the control language $\{H_C(P_1) \dots H_C(P_p)\}^*$ means that the set of productions available at a time t in a derivation is P_i if $t = kp+i$, $k \geq 0$; p is called the *period* of the time-varying system. The corresponding families of languages generated by time-varying grammars with appearance checking, without appearance checking, with unconditional transfer, and with period p, are denoted by $\mathcal{L}(X\text{-}TV_{ac}(p))$, $\mathcal{L}(X\text{-}TV(p))$, and $\mathcal{L}(X\text{-}TV_{ut}(p))$, respectively; if p may be arbitrarily large, p is replaced by $*$ or omitted in these notations.

In many cases it is not necessary to insist that the control string $H_C(P_1) \dots H_C(P_n)$ of a derivation is in L, it usually also is sufficient that $H_C(P_1) \dots H_C(P_n)$ is a prefix of some string in L. We call this control *weak* and replace C by wC and TV by wTV in the notions of the families of languages. We should like to mention that in the case of wTV the control words are just prefixes of the ω-word $(H_C(P_1) \dots H_C(P_p))^\omega$.

In the case of string grammars, from the results stated in [12], we obtain the following, for $\alpha \in \{\lambda, w\}$:

$$\begin{aligned} RE &= \mathcal{L}(CF\text{-}GC_{ac}) = \mathcal{L}(CF\text{-}GC_{ut}) \\ &= \mathcal{L}(CF\text{-}P_{ac}) = \mathcal{L}(CF\text{-}MAT_{ac}) \\ &= \mathcal{L}(CF\text{-}\alpha C(REG)_{ac}) = \mathcal{L}(CF\text{-}\alpha C(REG)_{ut}) \\ &= \mathcal{L}(CF\text{-}\alpha TV_{ac}) = \mathcal{L}(CF\text{-}\alpha TV_{ut}) \\ &\supseteq \mathcal{L}(CF\text{-}GC) = \mathcal{L}(CF\text{-}P) = \mathcal{L}(CF\text{-}MAT). \end{aligned}$$

Remark 1. We would like to mention that in the standard definition only the rules themselves are labeled for the control language and not all the subsets of rules, which corresponds with having only one rule assigned to each node in a graph-controlled grammar and not allowing a set of rules to be assigned. Yet in our more general definition, time-varying grammars are just an easy special case of grammars with regular control.

In [2], time-varying (tissue) P systems with only one cell are considered. As a sequential (tissue) P system with only one cell and no interaction with the environment corresponds with a multiset grammar, the main results obtained in this paper can be expressed as follows:

Theorem 2. *For all $\alpha \in \{\lambda, w\}$, $\beta \in \{\lambda, ac, ut\}$, and $k, p \geq 1$,*

$$\begin{aligned} \mathcal{L}(mARB\text{-}MAT) &= \mathcal{L}(mCF\text{-}MAT) \\ &= Ps\mathcal{L}(CF\text{-}MAT) \\ &= \mathcal{L}(mARB) \\ &= \mathcal{L}(mARB\text{-}TV(p)) \\ &= \mathcal{L}\left(mARB\text{-}\alpha C(REG)_\beta\right) \\ &= \mathcal{L}\left(mCF\text{-}\alpha C\left(REG^{1*}(*, p+1)\right)_\beta\right) \\ &= \mathcal{L}\left(mCF\text{-}\alpha C(REG)_\beta\right). \end{aligned}$$

Theorem 3. *For all $\alpha \in \{\lambda, w\}$, $\beta \in \{ac, ut\}$, $p \geq 12$, and $d \geq 2$,*

$$\mathcal{L}(mCF\text{-}\alpha TV_\beta(p,d)) = PsRE.$$

The additional parameter d – for *delay* – indicates that for at most d steps no rule is applied during any derivation before it stops.

4 Tissue P Systems Working on Vesicles of Multisets With Point Mutation Rules

We now define the model of tissue P systems working on vesicles of multisets in the sequential mode using point mutation rules:

Definition 2. *A* tissue P systems working on vesicles of multisets in the sequential derivation mode with point mutation rules *(an* stPV *system for short) is a tuple*

$$\Pi = (L, V, T, R, g, (i_0, w_0), h)$$

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,*

- *R is a* set of rules *of the form (i,p) where $i \in L$ 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 $R_i = \{(i,p) \mid (i,p) \in R\}$, so that $R = \bigcup_{i \in L} R_i$; moreover, for the sake of conciseness, we may simply write $R_i = \{p \mid (i,p) \in R\}$, too;*
- *g is a directed graph describing the underlying communication structure of Π, $g = (N, E)$ with $N = L$ being the set of nodes of the graph g and the set of edges $E \subseteq N \times N$;*
- *(i_0, w_0) describes the* initial vesicle *containing the multiset w_0 in cell i_0;*
- *h is the* output cell *for extracting the terminal results.*

The stPV system now works as follows: The computation of Π starts with a vesicle containing the multiset w_0 in cell i_0, and the computation proceeds with derivation steps until a specific output condition is fulfilled, which in all possible cases means that the vesicle has arrived in the output cell h.

In each derivation step, with the vesicle enclosing the multiset w being in cell i, tentatively (i.e., if possible) one rule p from R_i is applied to w and the resulting multiset in its vesicle is moved to a cell j such that $(i,j) \in E$.

As we are dealing with membrane systems, the classic output condition – in the generating case – would be to only consider halting computations; yet as we allow that we also move the vesicle to the next cell even if no rule can be applied to the multiset in the vesicle, the most natural condition is to take as results only those multisets which have arrived in the output cell h enclosed in the vesicle and only consist of terminal symbols.

On the other hand, *halting* still can be defined as follows for vesicles enclosing terminal multisets which have arrived in the output cell h: consider any loopless path from the output cell h through the communication graph g; then halting can be defined as having no such path along which we arrive in a cell where still a rule can be applied.

In [2], the notion *halting with delay d* is used to describe the situation that we allow the system to stay inactive (i.e., without applying a rule) for d steps before a computation is said to halt. We will also use this refinement of halting in this paper to specify how many steps we have to go ahead in order to see that the system halts; according to the definition for halting given above, in the worst case d is the length of the longest loopless path through the communication graph g.

Hence, for the tissue P systems considered in this paper we may specify the following derivation and output strategies:

- *halt*: the only condition is that the system halts in the sense explained above, i.e., no matter which path we followed in the communication graph g, no rule will be applicable any more; 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);

- $(halt, d)$: the result is the multiset contained in the vesicle to be found in cell h, but the additional condition is that in no successful derivation (i.e., a derivation yielding a result) more than d steps without applying a rule may occur; the special case $d = 0$ means that we do not allow a derivation step where no rule is applied (again 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);
- $(term, d)$: 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), but, in addition, we require that in any successful derivation at most d steps without applying a rule may occur;
- $(halt, term)$: both conditions must be fulfilled, i.e., the resulting multiset contained in the vesicle to be found in cell h consists of terminal symbols only as well as the system halts in the way defined above;
- $(halt, term, d)$: all three conditions must be fulfilled, i.e., the resulting multiset contained in the vesicle to be found in cell h consists of terminal symbols only, the system halts in the way defined above, and in any successful derivation at most d steps without applying a rule may occur.

Instead of *halt*, *term*, and $(halt, term)$ we may also write $(halt, *)$, $(term, *)$, and $(halt, term, *)$, respectively.

The set of all multisets obtained as results of computations in Π with the output being obtained by using the derivation and output strategy $\beta \in \mathbb{D}$,

$$\mathbb{D} = \{(halt, \alpha), (term, \alpha), (halt, term, \alpha) \mid \alpha \in \mathbb{N} \cup \{*\}\}$$

is denoted by $Ps(\Pi, sequ, gen, \beta)$, with *sequ* specifying that the tissue P system works in the sequential derivation mode and *gen* indicating that Π is considered as a generating device; if we are only interested in the number of symbols in the resulting multiset, the corresponding set of natural numbers is denoted by $N(\Pi, sequ, gen, \beta)$. The families of sets of (k-dimensional) vectors of natural numbers and sets of natural numbers generated by stPV systems with at most n cells using the derivation and output strategy β are denoted by $Ps(tPV_n, sequ, gen, \beta)$ and $N(tPV_n, sequ, gen, \beta)$, respectively. If n is not bounded, we simply omit the subscript in these notations.

The tissue P systems defined above can also be used as accepting and as computing devices: the input multiset w then is added to the initial multiset w_0, i.e., we start with the multiset ww_0 enclosed in the vesicle to be found in cell i_0 at the beginning of the computation. In the computing case, the results are obtained as described above according to the two different derivation and output strategies β as explained above. In the accepting case, the final multiset can either be required to be empty or else even to be any multiset over V, in which case only getting the vesicle into the output cell is the main acceptance criterion. We here prefer to require the strong condition that the final multiset in the vesicle having arrived in cell h must be the empty multiset. In all the notions defined in the preceding paragraph, *gen* is replaced by *acc* or *comp* in

5 Simulation Results

Our first results show that without allowing derivation steps where no rule is applied, the computational power of stPV systems is very much reduced, i.e., such systems only have the power of partially blind register machines.

Lemma 1. *For any* $\beta \in \{(term, 0), (halt, term, 0)\}$,

$$PsPBRM \subseteq Ps(stPV, sequ, gen, \beta).$$

Proof. Let $K \in PsPBRM$, i.e., the vector set K can be generated by a partially blind register machine $M = (m, B, l_0, l_h, P)$. We now define an equivalent stPV system Π generating K, i.e., $Ps(\Pi, sequ, gen, (term, 0)) = K$. The number of symbols a_r represents the contents of register r.

$$\begin{aligned}
\Pi &= (L, V, T, R, g = (L, E), (i_0, w_0), h),\\
L &= B,\\
V &= \{a_r \mid 1 \leq r \leq m\},\\
T &= \{a_r \mid 1 \leq r \leq k\},\\
R &= \{(p, I(a_r)) \mid p : (ADD(r), q, s) \in P\}\\
&\quad \cup \{(p, D(a_r)) \mid p : (SUB(r), q) \in P\},\\
E &= \{(p, q), (p, s) \mid p : (ADD(r), q, s) \in P\}\\
&\quad \cup \{(p, q) \mid p : (SUB(r), q) \in P\},\\
(i_0, w_0) &= (l_0, \lambda),\\
h &= l_h.
\end{aligned}$$

The simulation of the computations in M by Π works in real time, i.e., one step of the register machine is simulated in one step of Π: incrementing register r by an ADD-instruction is simulated by inserting a symbol a_r, decrementing register r by a SUB-instruction is simulated by deleting a symbol a_r; in case we try to decrement an empty register, the partially blind register machine aborts the computation, which is mimicked in the stPV system with Π getting stuck in cell p, as without applying the rule $D(a_r)$ the vesicle cannot move further and thus will never reach the output cell l_h.

Any halting computation in M finally reaches the halting instruction labeled by l_h, and thus in Π the vesicle obtained so far has also reached the output cell h, and as no rule is assigned to cell h, the computation in Π also halts when a vesicle reaches the output cell h. Provided no non-terminal symbol a_r with $k+1 \leq r \leq m$ is still present, the computation in Π yields the same result as the corresponding computation in M, i.e., we conclude that $Ps(\Pi, sequ, gen, (term, 0)) = K$ and $Ps(\Pi, sequ, gen, (halt, term, 0)) = K$, too.

The construction given in the preceding proof does not need additional symbols, the computation is guided by the communication graph g. Moreover, only

If we only require halting, we have to guarantee that a vesicle arriving in the output cell can only stay there if its contents entirely consists of terminal symbols; in this case, we not only need an additional symbol (the trap symbol #), but also substitution rules:

Lemma 2. $PsPBRM \subseteq Ps(stPV, sequ, gen, (halt, 0))$.

Proof. Let $K \in PsPBRM$, i.e., the vector set K can be generated by a partially blind register machine $M = (m, B, l_0, l_h, P)$. We now define an equivalent stPV system Π generating K such that $Ps(\Pi, sequ, gen, (halt, 0)) = K$. The number of symbols a_r represents the contents of register r.

The simulation of the computations in M by Π works in real time as described in the preceding proof: incrementing register r by an ADD-instruction is simulated by inserting a symbol a_r, decrementing register r by a SUB-instruction is simulated by deleting a symbol a_r.

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

Any halting computation in M finally reaches the halting instruction labeled by l_h, and thus in Π the vesicle obtained so far has also reached the output cell h. Yet now we have to check by applying trap rules $S(a_r, \#)$ that no non-terminal symbol a_r with $k+1 \leq r \leq m$ is still present to guarantee that the computation in Π yields the same result as the corresponding computation in M: If the multiset enclosed in the vesicle which has reached the output cell h only contains terminal symbols a_r with $1 \leq r \leq k$, then no rule $S(a_r, \#)$ for a non-terminal symbol a_r with $k+1 \leq r \leq m$ is applicable, i.e., the computation halts. Otherwise, the application of one of these trap rules $S(a_r, \#)$ forces the the rule $S(\#, \#)$ to be applicable in cell h again and again, i.e., the computation gets trapped in an infinite loop, so no unwanted result will be obtained in that case.

In sum, we conclude that $Ps(\Pi, sequ, gen, (halt, 0)) = K$. □

We now also show that the computations of an stPV system using the derivation and output strategy $(term, 0)$ can be simulated by a partially blind register machine.

Lemma 3. $Ps(stPV, sequ, gen, (term, 0)) \subseteq PsPBRM$.

Proof (Sketch). Let $(L, V, T, R, g = (L, E), (i_0, w_0), h)$ be an arbitrary stPV system 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 $L = \{i \mid 1 \leq i \leq n\}$.

We now construct a register machine $M = (m, B, l_0, l_h, P)$ generating $Ps(\Pi, sequ, gen, (term, 0))$, 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 [16]. For example, instead of a nondeterministic ADD-instruction $p : (ADD(r), q, s)$ we use the two deterministic 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. In general, the number of symbols a_r in the current multiset enclosed in the vesicle is represented by the corresponding number of symbols in the registers r, $1 \leq r \leq m$.

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

- for any combination of rules $(p, I(a_r)) \in R$ and edges $(p, q) \in E$ we take the instruction $p : (ADD(r), q)$ into P.
- for any combination of rules $(p, D(a_r)) \in R$ and edges $(p, q) \in E$ we take the instruction $p : (SUB(r), q)$ into P.
- for any combination of rules $(p, S(a_r, a_s)) \in R$ and edges $(p, q) \in E$ we take the sequence of two instructions $p : (SUB(a_r), p')$ and $p' : (ADD(a_s), q)$ into P using an intermediate label p'.

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

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

We observe that $\tilde{h}, \hat{h}$ are new labels different from all others. The new label $\hat{h}$ now the only halting instruction of M, i.e., $l_h = \hat{h}$. Hence, M must have reset to zero all its working registers before reaching $\hat{h}$ to satisfy the final zero check, which corresponds to Π having produced a multiset consisting exclusively of terminal symbols.

As a consequence of Lemmas 1 and 3 we obtain:

Theorem 4. $PsPBRM = Ps(stPV, sequ, (term, 0))$.

It remains as a challenging open question if a similar characterization result can also be obtained for the derivation and output strategies $(halt, 0)$ and $(term, halt, 0)$ including the condition that no rule can be applied any more in the output cell. This immediately turns out to be true if we require that no rules should be applied in the output cell, which seems to be a very natural condition. In general, without this condition, this need not be true: just consider a loop in the output cell h with an insertion rule leading back to h itself or via an additional cell h' as used in the proof of Lemma 2.

We now turn our attention to stPV systems allowing to continue the computation even if in some derivation step(s) no rule can be applied:

Theorem 5. $PsRE \subseteq Ps(stPV, sequ, gen, \beta)$ *for any* β *with*

$$\beta \in \mathbb{D} \setminus \{(halt, term, 0), (halt, 0), (term, 0)\}.$$

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 $M = (m, B, l_0, l_h, P)$ with the first k registers being the output registers and the other ones being working registers, which without loss of generality can be assumed to be empty at the end of any successful computation. We now define an stPV system Π generating K, i.e., $Ps(\Pi, sequ, gen, \beta) = K$:

$$\begin{aligned}
\Pi &= (L, V, T, R, g = (L, E), (i_0, w_0), h),\\
L &= B \cup \{p, \bar{p}, \bar{p}', \hat{p}, \hat{p}' \mid p : (SUB(r), q, s) \in P\},\\
V &= \{a_r \mid 1 \leq r \leq m\} \cup \{e, \#\},\\
T &= \{a_r \mid 1 \leq r \leq k\},\\
R &= \{(p, I(a_r)) \mid p : (ADD(r), q, s) \in P\}\\
&\quad \cup \{(p, I(e)), (\hat{p}, S(a_r, \#)), (\hat{p}', D(e)),\\
&\quad\ (\bar{p}, S(e, \#)), (\bar{p}, D(a_r)), (\bar{p}', D(e)) \mid p : (SUB(r), q) \in P\}\\
&\quad \cup \{(h, S(\#, \#)), (h', S(\#, \#))\},\\
E &= \{(p, q), (p, s) \mid p : (ADD(r), q, s) \in P\}\\
&\quad \cup \{(p, \bar{p}), (p, \hat{p}), (\bar{p}, \bar{p}'), (\hat{p}, \hat{p}'), (\bar{p}', q), (\hat{p}', s) \mid p : (SUB(r), q) \in P\}\\
&\quad \cup \{(h, h'), (h', h)\},\\
(i_0, w_0) &= (l_0, \lambda),\\
h &= l_h.
\end{aligned}$$

$(ADD(r), q, s)$ is simulated by applying the rule $(p, I(a_r))$ and then sending the vesicle from cell p either to cell q or cell s.

$(SUB(r), q, s)$ is simulated by first inserting one symbol e, which at the end of a correct simulation path is to be eliminated again in cells $\bar{p}'$ and $\hat{p}'$, respectively.
The *decrement case* follows the path $p - \bar{p} - \bar{p}' - q$; if chosen correctly, in cell $\bar{p}$ the rule $D(a_r)$ can be applied, otherwise the trap rule $S(e, \#))$ *must* be applied, in which case instead of eliminating e in cell $\bar{p}'$ no rule will be applicable there.
The *zero-test case* follows the path $p - \hat{p} - \hat{p}' - s$; if chosen correctly, i.e., if no symbol a_r is present (indicating that register r is empty), the rule $S(a_r, \#))$ will not be applicable in cell $\hat{p}$, i.e., the vesicle will move to cell $\hat{p}'$ without a rule having been applied. In cell $\hat{p}'$, the rule $D(e)$ will be applicable in any case.

Any halting computation in M finally reaches the halting instruction labeled by l_h, and thus in Π the vesicle obtained so far has arrived in the output cell h. Provided no trap symbol # has been generated during the simulation of the computation in M by the stPV system Π, the multiset in this vesicle only contains terminal symbols and the computation in Π halts as well. In case a trap symbol occurs, the computation gets stuck in an infinite loop in cell h.

We observe that by construction in two succeeding derivation steps it may happen at most once that no rule is applicable; hence, we conclude that $Ps(\Pi, smax, \beta) = K$ for any β with

$$\beta \in \mathbb{D} \setminus \{(halt, term, 0), (halt, 0), (term, 0)\}$$

which observation completes the proof. □

The construction given in the preceding proof could be modified in several ways: for example, we could replace the additional symbol e by # and replace the rules $(\bar{p}, S(e, \#))$ by the corresponding rules $(\bar{p}, I(\#))$.

Another variant would be to avoid the introduction of e anyway and take two simple paths for the decrement and the zero-test case:

The *decrement case* follows the path $p - q$ with $R_p = \{D(a_r), I(\#))\}$; if chosen correctly, the rule $D(a_r)$ can be applied, otherwise the trap rule $I(\#))$ *must* be applied introducing the trap symbol #.

The *zero-test case* follows the path $p' - s$ with $R_{p'} = \{S(a_r, \#)\}$; if chosen correctly, i.e., if no symbol a_r is present (indicating that register r is empty), the rule $S(a_r, \#)$ will not be applied, otherwise the trap symbol is introduced.

In sum, with this variant we only need two cells p and p' for a SUB-instruction p as well as a simpler communication structure, but from every cell having an edge to p we now need both edges to p *and* p'.

The major drawback of this simpler construction is that we cannot guarantee that the delay between two rule applications is not bigger than one: if the register machine M performs k successful zero checks, then Π in a correct simulation path does not apply any rule to the vesicle for k successive steps.

As already discussed in Sect. 1, the control given by the communication graph in stPV systems nicely corresponds to the control mechanisms in graph-controlled multiset grammars with unconditional transfer and in multiset gram-

of an stPV system can be interpreted as the control graph of the corresponding graph-controlled multiset grammar. In a multiset grammar with regular control and unconditional transfer, the communication graph of the stPV system describes a regular control language over L, hence, the proof of Theorem 5 also yields a proof of the following result for graph-controlled multiset grammars with unconditional transfer as well as multiset grammars with regular control and unconditional transfer, both using insertion, deletion, and substitution point mutation rules (we refer to this type of rules by $mIDS$):

Theorem 6. $PsRE = \mathcal{L}(mIDS\text{-}C(REG)_{ut}) = \mathcal{L}(mIDS\text{-}GC_{ut})$.

6 Simulating Computations with Strings

As already discussed in Sect. 1, we can also simulate computations on strings by adding rules for taking an object from the environment into the vesicle and/or sending an object out of the vesicle. We then define the input and output strings as the sequences of symbols taken in and sent out, respectively (for example, see [14,15]). As an input sequence can be encoded as a number and then processed by only two registers (the folklore result for two-counter automata) and the final result can be encoded as a number and then decoded into a sequence of symbols sent out (e.g., see [13]), the basic result how to simulate register machines elaborated in Sect. 5 also allows for simulating computations on strings.

As these additional rules we may use $read(a)$ and $write(a)$ which correspond to taking an object from the environment into the vesicle and sending an object out from the vesicle, respectively.

7 Going Beyond Turing

As already discussed in Sect. 1, a possibility to "go beyond Turing" is to simulate red-green register machines as first introduced in the area of membrane systems in [5], where the concept of *red-green P automata* for several specific models of membrane systems is explained. The basic idea of these red-green automata is to distinguish between two different sets of states (red and green states) and to consider infinite runs of the automaton on finite input objects (strings, multisets); allowed to change between red and green states more than once, red-green automata can recognize more than the recursively enumerable sets (of strings, multisets).

The main problem with finding a red-green variant of the special variant of tissue P systems introduced in this paper is its inherent non-determinism as can be seen from the proof of Theorem 5, even when simulating deterministic register machines and their red-green variant – the zero-test case and the decrement case of a SUB-instruction are chosen in a non-deterministic way. To overcome this problem we now rely on k-determinism (looking ahead k steps in order to know how to proceed in a deterministic way) as, for example, discussed in [30], as well as on the concept of toxic objects, which "kill" any derivation branch that introduces one of the them, as it happens [illegible]

As we can see from the proof of Theorem 5, simulating the SUB-instructions of a register machine only needs a look-ahead of 1, i.e., as the deterministic continuation of a derivation in the tissue P system we can just take the one which does not introduce the trap symbol. In that way, also a red-green variant of the model introduced in this paper can be defined, thus even allowing us to "go beyond Turing":

We use the simulation technique as outlined in the proof of Theorem 5, and then all variants of an instruction label p get the same color as p itself. In that way, mind changes are also simulated in a correct way. The only additional trick is to make the infinite run on a finite input deterministic by choosing the branch which does not introduce the trap symbol #.

8 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 in the sequential derivation mode. Without allowing computation steps to not apply a rule, we obtain a characterization of the sets of (vectors of) natural numbers generated by partially blind register machines, whereas otherwise we can generate every recursively enumerable set of (vectors of) natural numbers. Adding the possibility of taking in symbols from the environment and/or sending out symbols to the environment from the vesicle, we are able to simulate computations on strings. Moreover, we even showed how to "go beyond Turing" based on the model of red-green register machines.

Several challenging topics remain for future research: for example, what happens if we are not allowed to use substitution rules (with all possible derivation and output strategies)?

References

1. Alhazov, A., Freund, R.: P systems with toxic objects. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 99–125. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14370-5_7
2. Alhazov, A., Freund, R., Heikenwälder, H., Oswald, M., Rogozhin, Yu., Verlan, S.: Sequential P systems with regular control. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, Gy. (eds.) CMC 2012. LNCS, vol. 7762, pp. 112–127. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36751-9_9
3. Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: (tissue) P systems with vesicles of multisets. In: Csuhaj-Varjú, E., Dömösi, P., Vaszil, Gy. (eds.) Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, 4–6 September 2017. EPTCS, vol. 252, pp. 11–25 (2017). https://doi.org/10.4204/EPTCS.252.6
4. Alhazov, A., Freund, R., Verlan, S.: P systems working in maximal variants of the set derivation mode. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) CMC 2016. LNCS, vol. 10105, pp. 83–102. Springer, Cham (2017). https://

5. Aman, B., Csuhaj-Varjú, E., Freund, R.: Red–green P automata. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 139–157. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14370-5_9
6. Arroyo, F., Gómez-Canaval, S., Mitrana, V., Popescu, Ş.: On the computational power of networks of polarized evolutionary processors. Inf. Comput. **253**(3), 371–380 (2017). https://doi.org/10.1016/j.ic.2016.06.004
7. Arroyo, F., Gómez Canaval, S., Mitrana, V., Popescu, Ş.: Networks of polarized evolutionary processors are computationally complete. In: Dediu, A.-H., Martín-Vide, C., Sierra-Rodríguez, J.-L., Truthe, B. (eds.) LATA 2014. LNCS, vol. 8370, pp. 101–112. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04921-2_8
8. Calude, C.S., Păun, Gh.: Bio-steps beyond Turing. BioSystems **77**(1–3), 175–194 (2004)
9. Castellanos, J., Martín-Vide, C., Mitrana, V., Sempere, J.M.: Networks of evolutionary processors. Acta Informatica **39**(6–7), 517–529 (2003). https://doi.org/10.1007/s00236-004-0158-7
10. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
11. Dassow, J., Păun, Gh.: On the power of membrane computing. J. UCS **5**(2), 33–49 (1999). https://doi.org/10.3217/jucs-005-02-0033
12. Fernau, H.: Unconditional transfer in regulated rewriting. Acta Informatica **34**(11), 837–857 (1997). https://doi.org/10.1007/s002360050108
13. Fernau, H., Freund, R., Oswald, M., Reinhardt, K.: Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. J. Autom. Lang. Comb. **12**(1–2), 117–138 (2007)
14. Freund, R.: P systems working in the sequential mode on arrays and strings. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 188–199. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30550-7_16
15. Freund, R.: P systems working in the sequential mode on arrays and strings. Int. J. Found. Comput. Sci. **16**(4), 663–682 (2005). https://doi.org/10.1142/S0129054105003224
16. Freund, R., Ibarra, O., Păun, Gh., Yen, H.C.: Matrix languages, register machines, vector addition systems. In: Third Brainstorming Week on Membrane Computing, pp. 155–167 (2005)
17. Freund, R., Ivanov, S., Staiger, L.: Going beyond turing with P automata: partial adult halting and regular observer ω-languages. In: Calude, C.S., Dinneen, M.J. (eds.) UCNC 2015. LNCS, vol. 9252, pp. 169–180. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21819-9_12
18. Freund, R., Ivanov, S., Staiger, L.: Going beyond Turing with P automata: regular observer ω-languages and partial adult halting. IJUC **12**(1), 51–69 (2016)
19. Freund, R., Kogler, M., Oswald, M.: A general framework for regulated rewriting based on the applicability of rules. In: Kelemen, J., Kelemenová, A. (eds.) Computation, Cooperation, and Life. LNCS, vol. 6610, pp. 35–53. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20000-7_5
20. Freund, R., Kogler, M., Rogozhin, Yu., Verlan, S.: Graph-controlled insertiondeletion systems. In: Proceedings of Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS 2010, Saskatoon, Canada, 8–10th August 2010, pp. 88–98 (2010). https://doi.org/10.4204/EPTCS.31.11

21. Freund, R., Oswald, M.: Tissue P systems and (mem)brane systems with mateand drip operations working on strings. Electron. Notes Theor. Comput. Sci. **171**(2), 105–115 (2007). https://doi.org/10.1016/j.entcs.2007.05.011
22. Freund, R., Oswald, M., Staiger, L.: ω-P automata with communication rules. In: Martín-Vide, C., Mauri, G., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2003. LNCS, vol. 2933, pp. 203–217. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24619-0_15
23. Freund, R., Păun, Gh.: How to obtain computational completeness in P systems with one catalyst. In: 2013 Proceedings of Machines, Computations and Universality, MCU 2013, Zürich, Switzerland, 9–11 September 2013, pp. 47–61 (2013). https://doi.org/10.4204/EPTCS.128.13
24. Freund, R., Rogojin, V., Verlan, S.: Computational completeness of networks of evolutionary processors with elementary polarizations and a small number of processors. In: Pighizzini, G., Câmpeanu, C. (eds.) DCFS 2017. LNCS, vol. 10316, pp. 140–151. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60252-3_11
25. Freund, R., Rogozhin, Yu., Verlan, S.: Generating and accepting P systems with minimal left and right insertion and deletion. Nat. Comput. **13**(2), 257–268 (2014). https://doi.org/10.1007/s11047-013-9396-3
26. Kudlek, M., Martín-Vide, C., Păun, Gh.: Toward a formal macroset theory. In: Calude, C.S., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2000. LNCS, vol. 2235, pp. 123–133. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45523-X_7
27. van Leeuwen, J., Wiedermann, J.: Computation as an unbounded process. Theor. Comput. Sci. **429**, 202–212 (2012). https://doi.org/10.1016/j.tcs.2011.12.040
28. Martín-Vide, C., Pazos, J., Păun, Gh., Rodríguez-Patón, A.: A new class of symbolic abstract neural nets: tissue P systems. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 290–299. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45655-4_32
29. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)
30. Oswald, M.: P automata. Ph.D. thesis, Faculty of Computer Science, Vienna University of Technology (2003)
31. Popescu, Ş.: Networks of polarized evolutionary processors with elementary polarization of symbols. In: NCMA 2016, pp. 275–285 (2016)
32. Păun, Gh.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693
33. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
34. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1–3. Springer, Heidelberg (1997)
35. Sosík, P., Valík, O.: On evolutionary lineages of membrane systems. In: Freund, R., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 67–78. Springer, Heidelberg (2006). https://doi.org/10.1007/11603047_5
36. Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php
37. The P Systems Website. http://ppage.psystems.eu/

Adaptive P Systems

Bogdan Aman[1,2(✉)] and Gabriel Ciobanu[1,2]

[1] Institute of Computer Science, Romanian Academy, Iaşi, Romania
bogdan.aman@iit.academiaromana-is.ro
[2] "A.I.Cuza" University of Iaşi, Iaşi, Romania
gabriel@info.uaic.ro

Abstract. In this paper we introduce a membrane system named *adaptive P system* which is able to adjust dynamically its behaviour depending on resource availability. Such a system is defined as a tree of membranes in which the objects are organized in multisets, and the rules are applied in a maximal parallel manner. We use *guards* on the right side of the rules in order to model the biological sensitivity to context, and in this way we are able to describe an adaptive behaviour. The Turing completeness of the adaptive P systems can be obtained only by using non-cooperative rules (with guards) working in the accepting case. Using the adaptive P systems, we provide a polynomially uniform solution for an NP-complete problem (Subset Sum) by using specific membrane computing techniques. The solution employs a linear number of resources and evolution steps.

1 Introduction

Membrane systems (also called P systems) are introduced by Gheorghe Păun as a model of distributed, parallel and non-deterministic systems inspired by cell biology [27]. A membrane system (P system) is a tree-like structure of hierarchically arranged membranes embedded in the *skin* membrane as the outermost part of the system. The membranes can be arranged in a tree (cell-like [28]) structure, or in a graph form (tissue-like [26] and neural-like [19]). In this paper we use the symbol-object P systems [28] in which cells are divided in various regions containing specific objects and evolution rules, each region with a different task and all of them working simultaneously to accomplish a more general task of the whole system. The objects evolve according to the specific rules associated with each region, and the regions cooperate in order to maintain the proper behaviour of the whole system. Several results and variants of membrane systems (inspired by different aspects of living cells like symport and antiport communication through membranes, catalytic objects, membrane charge, etc.) are presented in [28]. There are defined various semantics which express how membrane systems evolve [9,10]. Over the years several books on membrane computing were published presenting the latest results in theory and applications [11,14,25,35,36]. Links between membrane systems and process calculi are