Natural Computing (2019) 18:593–608
https://doi.org/10.1007/s11047-019-09747-5

Variants of P systems with activation and blocking of rules

Artiom Alhazov[1] · Rudolf Freund[2] · Sergiu Ivanov[3]

Published online: 1 August 2019

Abstract
We introduce new possibilities to control the application of rules based on the preceding applications, which can be defined in a general way for (hierarchical) P systems and the main known derivation modes. Computational completeness can be obtained even with non-cooperative rules and using both activation and blocking of rules, especially for the set modes of derivation, when allowing derivation steps with no rules being applied. When we allow the application of rules to influence the application of rules in previous derivation steps, applying a non-conservative semantics for what we consider to be a valid infinite derivation, we can even "go beyond Turing".

Keywords Activation of rules · Blocking of rules · Computational completeness · Derivation modes · P systems · Go beyond Turing

1 Introduction

Originally defined by Gheorghe Păun in 1998, see Păun (1998), membrane systems, also known as P systems, are a model of computing based on the abstract notion of a membrane which can be seen as a container delimiting a region containing objects which are acted upon by the rewriting rules associated with the membrane. Quite often, these objects are multisets (for basic results on multiset computing, for example, see Kudlek et al. 2001), yet P systems operating on more complex objects (e.g., strings, arrays) are often considered, too, for instance, see Freund (2005).

A comprehensive overview of different flavors of membrane systems and their expressive power is given in the handbook which appeared in 2010, see Păun et al. (2010). For a state of the art snapshot of the domain, we refer the reader to the P systems website (http://ppage.psystems.eu/) as well as to the Bulletin of the International Membrane Computing Society (http://membranecomputing.net/IMCSBulletin/index.php).

Nearly thirty years ago, the monograph on regulated rewriting by Dassow and Păun (1989) already gave a first comprehensive overview on many concepts of regulated rewriting, especially for the string case. Yet as it turned out later, many of the mechanisms considered there for guiding the application of productions/rules can also be applied to other objects than strings, e.g., to n-dimensional arrays (Freund 1994). As exhibited in Freund et al. (2011), for comparing the generating power of grammars working in the sequential derivation mode, 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, using a general model for graph-controlled, programmed, random-context, and ordered grammars of arbitrary type based on the

This is a revised and extended version of the paper presented at UCNC 2018 in Fontainebleau, France, see Alhazov et al. (2018d).

The work is supported by National Natural Science Foundation of China (61320106005, 61033003, and 61772214) and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province (154200510012)

✉ Rudolf Freund
rudi@emcc.at

Artiom Alhazov
artiom@math.md

Sergiu Ivanov
sergiu.ivanov@univ-evry.fr

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science, Academiei 5, 2028 Chişinău, Moldova

[2] Faculty of Informatics, TU Wien, Favoritenstraße 9–11, 1040 Vienna, Austria

[3] IBISC, Université Évry, Université Paris-Saclay, 23 Boulevard de France, 91025 Évry, France

applicability of rules. Also in the field of P systems (Păun et al. 2010; http://ppage.psystems.eu/), where mainly multisets have been considered, such regulating mechanisms were used, e.g., see Cavaliere et al. (2007).

Dynamic evolution of the set of available rules has been considered from the very beginning of membrane computing. Already in 1999, generalized P systems were introduced in Freund (1999); in these systems the membranes, alongside the objects, contain *operators* which act on these objects, while the P system itself acts on the operators, thereby modifying the transformations which will be carried out on the objects in the subsequent steps. Among further ideas on dynamic rules, one may list rule creation (Arroyo et al. 2002), activators (Alhazov 2004), inhibiting/deinhibiting rules (Cavaliere et al. 2004), and symport/antiport *of rules* (Cavaliere and Genova 2004). One of the more recent developments in this direction are *polymorphic P systems* (Alhazov et al. 2015, 2011; Ivanov 2014), in which rules are defined by pairs of membranes, whose contents may be modified by moving objects in or out, as well as P systems with randomized right-hand sides of rules (Alhazov et al. 2017, 2018a), where the right-hand sides are chosen randomly and in different ways from the given set of rules.

We here follow an approach started to be elaborated in Alhazov et al. (2018b) and then continued in Alhazov et al. (2018e), where in the general framework of sequential systems the applicability of rules is controlled by the application of rules in the preceding derivation step(s). The application of a rule in one derivation step may either activate some rules to be applied in the next derivation step(s) or may block their application. We immediately observe that the application of a rule requires its activation in a preceding step. A computation may also take derivation steps without applying a rule as long as there are some rules activated for future derivation steps. In contrast to the general framework for control mechanisms as described in Freund et al. (2011), we here are not dealing with the applicability of rules itself but with the possible activation or blocking of rules by the effective application of rules in preceding steps.

The idea of using activation and blockings of rules in the area of membrane systems first was considered in Alhazov et al. (2018c) and then in Alhazov et al. (2018d). In this paper we extend the results for P systems with activation and blockings of rules already obtained in these papers for several variants of P systems. For example, we will establish computational completeness results for various kinds of one-membrane P systems (resembling multiset grammars) and several derivation modes, using activation and blocking of rules to be applied in succeeding derivation steps. Depending on the derivation mode and the halting condition, the complexity of the systems may vary either with respect to the number of steps rules are activated ahead or whether the use of blocking rules is needed or not. Allowing a derivation step without applying a rule enables the P system to check for the appearance of a symbol in the current multiset.

We may even allow the application of rules to influence previous derivation steps, but using a conservative semantics that considers derivations to be consistent when such backwards activations or blockings of rules are not changing the correctness of the derivation, we cannot "go beyond Turing", which on the other hand can be achieved by allowing such backwards information to change past configurations by triggering the applications of newly activated rules and by using a less conservative semantics looking at infinite computations on finite multisets as in "red-green P automata" (for instance, see Freund 2016).

Various possibilities of how one may "go beyond Turing" are discussed in van Leeuwen and Wiedermann (2012), for example, the definitions and results for red-green Turing machines can be found there. In Aman et al. (2014) 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 Calude and Păun (2004) and Sosík and Valík (2006). Computations with infinite words by P automata were investigated in Freund et al. (2004).

In Freund et al. (2015, 2016), 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 with ω-regular languages over $\{0,1\}$ of a specific form ending with 1^ω as observer languages. The special observer language $\{0,1\}^*\{1\}^\omega$ corresponds with the acceptance condition for P automata called "partial adult halting". This special acceptance variant for P

automata with infinite runs on finite multisets is motivated by an observation made for infinite sequences of infinite runs of a given P automaton: at some moment, a specific initial part in the sequence of configurations in the infinite sequence of runs does not change any more, for example, the initial configuration.

We now may also consider variants of P systems with activation and blocking of rules as well as infinite computations on a given finite multiset. A sequence of such infinite computations is called *valid* if each prefix of these computations becomes stable, i.e., neither the configuration itself nor the set of applicable rules changes any more. This less conservative semantics for activating and/or blocking the rules in preceding computations allows us to take the infinite sequence of stable configurations obtained in this way as the final computation on the given input. Provided such a computation—obtained as the limit of a valid sequence of computations—exists, we may just consider the result of the first computation step and thus the second configuration to see whether the input has been accepted. Again this can be seen as looking at a specific initial part of the computations and requiring that it does not change any more, but also requesting that the whole computation converges.

In the following section, we recall some notions from formal language theory; in the succeeding section the main definitions of the general framework for P systems working under different derivation modes (see Freund and Verlan 2007) are given. Then we define the new concept of activation and blocking of rules based on the applicability of rules within this general framework of static P systems, i.e., of P systems where the membrane structure does not change during any computation. In Sect. 5 we prove first results only using activation of rules for the next step. We also establish relations to the family of sets of multisets generated by tabled Lindenmayer systems and by P systems with promoters and inhibitors.

Computational completeness results using both activation and blocking of rules for at most the next two steps are established in Sect. 6, for different types of rules and various derivation modes, where we use the feature to allow at least one derivation step in between where no rule is applied. Then we extend our systems by allowing activation and blocking of rules in previous derivation steps in Sect. 7, and finally discuss how to "go beyond Turing" in Sect. 8. A summary of the results obtained in this paper and some future research topics extending the notions and results considered in this paper are given in Sect. 9.

2 Definitions

After some preliminaries from formal language theory, we define our model for hierarchical P systems in the general setting of this paper as well as the main derivation modes considered in the area of membrane systems, see Freund and Verlan (2007).

2.1 Preliminaries

The set of integers is denoted by $\mathbb{Z}$, the set of non-negative integers (natural numbers) by $\mathbb{N}_0$, and the set of positive integers by $\mathbb{N}$. An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*; the elements of V^* are called strings, and the *empty string* is denoted by λ; $V^*\backslash\{\lambda\}$ is denoted by V^+. Let $\{a_1,\ldots,a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to $a_1,\ldots a_n$ is $\left(|x|_{a_1},\ldots,|x|_{a_n}\right)$. The *Parikh image* of a language L over $\{a_1,\ldots,a_n\}$ is the set of all Parikh vectors of strings in L, and we denote it by $Ps(L)$. For a family of languages FL, the family of Parikh images of languages in FL is denoted by $PsFL$. The families of regular and recursively enumerable string languages are denoted by REG and RE, respectively.

A (finite) multiset u over the (finite) alphabet V, $V=\{a_1,\ldots,a_n\}$, is a mapping $f: V \longrightarrow \mathbb{N}_0$ and can be represented by any string x the Parikh vector of which with respect to $a_1,\ldots,a_n$ is $(f(a_1),\ldots,f(a_n))$. The *weight* of u is defined as $f(a_1)+\ldots+f(a_n)$. The set of all finite multisets over an alphabet V is denoted by V°.

For more details of formal language theory the reader is referred to the monographs and handbooks in this area (Dassow and Păun 1989; Rozenberg and Salomaa 1997).

2.2 Register machines

As a computationally complete model able to generate (accept) all sets in $PsRE$ we will use register machines:

A *register machine* is a construct $M=(n,H,R_M,p_0,h)$ where n, $n\geq 1$, is the number of registers (each of them contains a non-negative integer), H is the set of instruction labels, p_0 is the start label, h is the halting label (only used for the HALT instruction), and R_M is a set of (labeled) instructions being of one of the following forms:

- $p:(\mathtt{ADD}(r),q,s)$ increments the value in register r and in a non-deterministic way chooses to continue either with the instruction labeled by q or with the instruction labeled by s;

- $p : (\mathtt{SUB}(r), q, s)$ decrements the value in register r and continues the computation with the instruction labeled by q if the register was non-empty, otherwise it continues with the instruction labeled by s;
- $h : \mathtt{HALT}$ halts the machine.

M is called deterministic if in all ADD-instructions $p : (\mathtt{ADD}(r), q, s)$, it holds that $q = s$; in this case we write $p : (\mathtt{ADD}(r), q)$. Deterministic register machines can accept all recursively enumerable sets of vectors of natural numbers with k components using precisely $k + 2$ registers, see Minsky (1967).

2.3 *ET0L*-systems

Lindenmayer systems (with tables of rules) are a well-known parallel mechanism for generating strings. An *ET0L*-system is a construct

$G = (V, T, R_1, \ldots, R_m, w)$ where

- V is the alphabet of *objects*;
- $T \subseteq V$ is the alphabet of *terminal objects*;
- $w \in V^*$ is the *initial string*;
- R_i, $1 \leq i \leq m$, is a finite set of *non-cooperative rules*, i.e., rules of the form $X \to u$ with $X \in V$ and $u \in V^*$. We denote $R = \bigcup_{1 \leq i \leq n} R_i$.

A computation in G starts with the initial string w, and in each derivation step the rules from one of the rule sets R_i, also called *tables*, are applied in the (maximally) parallel way; a successful computation ends with a terminal string over the terminal alphabet T.

An *ET0L*-system can also be seen as a multiset generating mechanism, i.e., we start with the initial object w being a multiset and then apply the rules in the tables as multiset rules. The terminal results then are multisets over the terminal alphabet T. To emphasize the multiset character of the system, we also call it an *mET0L*-system. The family of sets of multisets generated by *mET0L*-systems with at most n tables is denoted by $PsET0L_n$. We omit the suffix n if the number of tables is not bounded.

We also consider *ET0L*-systems and *mET0L*-systems with promoters and inhibitors, i.e., the rules in G are of the form (p, P, Q) where $p \in R$ and P, Q are finite sets of finite multisets. The multisets in P are called *promoters*, those in Q *inhibitors*. A rule (p, P, Q) is applicable to a multiset w if and only if every multiset in P and none of the multisets in Q is contained in w.

An *ET0L*-system/*mET0L*-systems is called of type $(pro_{i,j}, inh_{k,m})$ if the number of multisets in the sets of promoters and inhibitors is limited by i and k, respectively, and the weight of the multisets in the sets of promoters and inhibitors does not exceed j and m, respectively. The family of sets of multisets generated by *mET0L*-systems with n tables of rules of type $(pro_{i,j}, inh_{k,m})$ is denoted by $PsET0L_n(pro_{i,j}, inh_{k,m})$.

3 A general model for hierarchical P systems

We first recall the main definitions of the general model for hierarchical P systems and the basic derivation modes as defined, for example, in Freund and Verlan (2007).

A *(hierarchical) P system of type X* working in the derivation mode δ is a construct

$\Pi = \left(V, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f, \Longrightarrow_{\Pi,\delta}\right)$ where

- V is the alphabet of *objects*;
- $T \subseteq V$ is the alphabet of *terminal objects*;
- μ is the hierarchical membrane structure (a rooted tree of membranes) with the membranes uniquely labeled by the numbers from 1 to m;
- $w_i \in V^*$, $1 \leq i \leq m$, is the *initial multiset* in membrane i;
- R_i, $1 \leq i \leq m$, is a finite set of *rules of type X* assigned to membrane i;
- f is the label of the membrane from which the result of a computation has to be taken from (in the generative case) or into which the initial multiset has to be given in addition to w_f (in the accepting case),
- $\Longrightarrow_{\Pi,\delta}$ is the derivation relation under the derivation mode δ.

3.1 Types of rules

The notion "rules of type X", for example, may stand for "evolution rules" or "communication rules". We now give examples for types of rules to be used further in this paper.

3.1.1 Evolution rules

In general, an *evolution rule* is of the form $u \to v$ with $u, v \in V^*$. An evolution rule $u \to v$ is called "non-cooperative" (abbreviated "ncoo") if $u \in V$, otherwise it is called "cooperative" (abbreviated "coo"). An evolution rule is applied within the region of the membrane it is assigned to. The evolution rule $u \to v$ can be applied in a membrane region if and only if the multiset in this region contains u as a submultiset. The application of $u \to v$ eliminates u and adds v. To the symbols in v also targets—out, in_j—may be assigned: target out means that the corresponding symbol is sent out to the surrounding membrane whereas a target in_j means that the corresponding symbol is sent into the inner membrane labeled by j.

3.1.2 Communication rules

A *communication rule* is of the form $u|v$ with $u, v \in V^*$ and not both being empty. The communication rule $u|v$ has the effect that u inside the membrane the rule is assigned to is exchanged with v in the outer region of this membrane (if u and v are available in the respective membrane regions). If both u and v are not empty, then $u|v$ is called an *antiport rule*, otherwise a *symport rule*. The type of rules $anti_{i,j}$ indicates that for every antiport rule $u|v$ in the P system the conditions $1 \leq |u| \leq i$ and $1 \leq |v| \leq j$ hold. The type of rules sym_i indicates that for every symport rule $u|\lambda$ or $\lambda|v$ in the P system the condition $1 \leq |u| \leq i$ or $1 \leq |v| \leq i$, respectively, holds.

3.1.3 Insertion, deletion, and substitution rules

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 $S(a, b)$; such a rule is called a *deletion rule* if $a \neq \lambda$ and $b = \lambda$, and it is also written as $D(a)$; $a \to b$ is called an *insertion rule* if $a = \lambda$ and $b \neq \lambda$, and we also write $I(b)$. The sets of all insertion rules, deletion rules, and substitution rules over an alphabet V are 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. 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.

These types of rules and the corresponding notations can be extended by allowing more than one symbol on the left-hand and/or the right-hand side, i.e., $a, b \in V^*$, and $ab \neq \lambda$. The corresponding sets of all extended insertion rules, deletion rules, and substitution rules over an alphabet V are denoted by Ins^*_V, Del^*_V, and Sub^*_V, respectively.

3.2 Derivation modes

The set of all multisets of rules applicable in each membrane to a given configuration can be restricted by imposing specific conditions, thus yielding the following basic derivation modes (for example, see Freund and Verlan (2007) for formal definitions):

- asynchronous mode (abbreviated *asyn*): at least one rule is applied;
- sequential mode (*sequ*): exactly one rule is applied;
- maximally parallel mode (*max*): a non-extendable multiset of rules is applied;
- maximally parallel mode with maximal number of rules (max_{rules}): a non-extendable multiset of rules of maximally possible cardinality is applied;
- maximally parallel mode with maximal number of objects ($max_{objects}$): a non-extendable multiset of rules affecting as many objects of the current configuration as possible is applied.

In Alhazov et al. (2016), these derivation modes are restricted in such a way that each rule can be applied at most once, thus yielding the set modes *sasyn*, *smax*, $smax_{rules}$, and $smax_{objects}$ (the sequential mode is already a set mode by definition).

A configuration is a list of the contents of each cell; a sequence of configurations $C_1, \ldots, C_k$ is called a *computation* in the derivation mode δ if $C_i \Longrightarrow_{\Pi,\delta} C_{i+1}$ for $1 \leq i < k$. The derivation relation $\Longrightarrow_{\Pi,\delta}$ is defined by the set of rules in Π and the given derivation mode which determines the multiset of rules to be applied to the multisets contained in each membrane.

3.3 Computations

As we are dealing with membrane systems, we consider the classic halting condition, i.e., the computation ends when no rule can be applied any more.

The set of all terminal multisets obtained as results of halting computations in Π by using the derivation mode δ is denoted by $Ps(\Pi, gen, \delta)$, with *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 $N(\Pi, gen, \delta)$.

The families of sets of (k-dimensional) vectors of natural numbers and sets of natural numbers generated by P systems with at most n cells using rules of type X in the derivation mode δ are denoted by $Ps(OP_n(X), gen, \delta)$ and $N(OP_n(X), gen, \delta)$, respectively. If n is not bounded, we simply omit the subscript in these notations.

We also consider P systems as accepting mechanisms: in membrane f, we add the input multiset w_0 to w_f in the initial configuration $C_1 = (w_1, \ldots, w_m)$ thus obtaining $C_1[w_0] = (w_1, \ldots, w_f w_0, \ldots, w_m)$; the input multiset w_0 is accepted if there exists a halting computation in the derivation mode δ starting from $C_1[w_0]$.

The families of sets of (k-dimensional) vectors of natural numbers and sets of natural numbers accepted by P systems with at most n cells using rules of type X in the derivation mode δ are denoted by $Ps(OP_n(X), acc, \delta)$ and $N(OP_n(X), acc, \delta)$, respectively. If n is not bounded, we simply omit the subscript in these notations.

3.4 Flattening

Many variants of P systems can be *flattened* to only one membrane: any object a in membrane i can be represented by an object $[a, i]$ in the *skin membrane*, the outermost membrane of the P system; the rules then can be adapted accordingly. For further details of the flattening procedure we refer to Freund et al. (2014).

For example, an evolution rule $a \to (b, out)$ assigned to membrane 2 inside the skin membrane 1 then is replaced by the rule $[a, 2] \to [b, 1]$ in the single membrane 1 of the corresponding flattened P system.

For communication rules, a second region is necessary, but in this case we can restrict ourselves to the regions inside and outside the skin membrane. If we assume that in the region outside the skin membrane, the *environment*, all objects are available in an unbounded number, then the communication rule $u|v$ assigned to the skin membrane has the same effect as the evolution rule $u \to v$. On the other hand, symport rules of the form $a|\lambda$ and $\lambda|a$ correspond to the deletion rule $D(a)$ and the insertion rule $I(a)$, respectively.

Throughout the paper we therefore will assume the simplest membrane structure of only one membrane which in effect reduces the P system to a multiset processing mechanism, and, observing that $f = 1$, in what follows we will use the reduced notation

$$\Pi = \left(V, T, w, R, \Longrightarrow_{\Pi,\delta}\right).$$

3.5 P systems with promoters and inhibitors

As for *mET0L*-systems we can also consider the rules in a P system

$$\Pi = \left(V, T, w, R, \Longrightarrow_{\Pi,\delta}\right).$$

to be rules with promoters and inhibitors, i.e., the rules in Π are of the form (p, P, Q) where $p \in R$ and P, Q are finite sets of finite multisets; the multisets in P are the *promoters*, those in Q are the *inhibitors*. A rule (p, P, Q) is applicable to a multiset w if and only if every multiset in P and none of the multisets in Q is contained in w.

A P system using rules of type X together with promoters and inhibitors is called of type $(X, pro_{i,j}, inh_{k,m})$ if the number of multisets in the sets of promoters and inhibitors is limited by i and k, respectively, and the number of objects in the multisets of the sets of promoters and inhibitors does not exceed j and m, respectively. The family of sets of multisets generated by P systems of type $(X, pro_{i,j}, inh_{k,m})$ with the derivation mode δ is denoted by $Ps(OP(X, pro_{i,j}, inh_{k,m}), gen, \delta)$. We omit $inh_{k,m}$ or $pro_{i,j}$ if no inhibitors or no promoters are used.

4 P systems with activation and blocking of rules

We now define our new concept of regulating the application of rules at a specific moment by activation and blocking relations for P systems (with only one membrane).

A *P system with activation and blocking of rules* of type X (an *AB-P system of type X* for short) working in the derivation mode δ is a construct

$$\Pi_{AB} = \left(\Pi, L, f_L, A, B, L_1, \Longrightarrow_{\Pi_{AB},\delta}\right)$$

where $\Pi = \left(V, T, w, R, \Longrightarrow_{\Pi,\delta}\right)$ is a P system of type X, L is a finite set of labels, f_L assigns one or more labels to each rule from R, A, B are finite subsets of $L \times L \times \mathbb{N}$, and $L_1 \subseteq L$ describes the set of rules which may be used in the first derivation step. The elements of A and B are of the form (p, q, t) with $p, q \in L$ and $t \in \mathbb{N}$; (p, q, t) indicates that t steps in the future the application of p activates (for A) or blocks (for B) the application of the rule q.

Now let $\Longrightarrow_{\Pi/P,\delta}$, for any set of rules $P \subseteq R$, denote the derivation relation obtained from $\Longrightarrow_{\Pi,\delta}$ by reducing the set of available rules from R to P. Then a sequence of multisets $w_i \in V^\circ$, $0 \le i \le n$, with $w_0 = w$ is called a *valid* derivation of $z = w_n$—we also write $w_0 \Longrightarrow_{\Pi_{AB},\delta} w_1 \Longrightarrow_{\Pi_{AB},\delta} \ldots w_n$—if and only if, with P_k denoting the set of rules applied to w_k in the kth derivation step, for every i, $0 \le i < n$, the following conditions hold true:

- either $w_i \Longrightarrow_{\Pi/P_i,\delta} w_{i+1}$, where P_i is the set of all rules r (identified by their labels) such that there is a relation $(r_j, r, t) \in A$ with $i - j = t$, which means that the application of a rule r_j in the jth derivation step has activated rule r probably to be applied in the ith derivation step, and there is no rule relation $(r_j, r, s) \in B$ such that $i - j = s$, which means that the application of the rule r_j in the jth derivation step would block rule r to be applied in the ith derivation step, **or**
- P_i is empty, i.e., no rule r is activated to be applied ith derivation step or every activated rule is blocked, too; in this case we take $w_i = w_{i-1}$ provided there is still some rule activated to be applied later.

With this interpretation we see that A can be called the set of *activating rule relations* and B the set of *blocking rule relations*. The role of L_1 is to get a derivation started by defining the rules to be applied in the first derivation step.

In the same way as for the original model of P systems we can define the derivations in the AB-P system Π_{AB}, now using the derivation relation $\Longrightarrow_{\Pi_{AB},\delta}$ instead of $\Longrightarrow_{\Pi,\delta}$. As we are dealing with membrane systems, the classic output condition—in the generating case—is to only consider halting computations; yet we also want to allow that for a

bounded number of steps no rule can be applied as long as still some rules are activated for future steps. Hence, for AB-P systems we consider several variants of halting conditions and output strategies.

4.1 Halting conditions

In Alhazov et al. (2013), 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 allow the system to go ahead without applying a rule.

Another natural condition is to take as results only those multisets which only consist of terminal symbols.

Hence, for the 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 rule is activated for future steps, which means that no rule will be applicable any more; the result is the multiset obtained at the end of such a halting computation (which in fact means that specifying the terminal alphabet is obsolete);
- (*halt*, *d*): the result is the multiset obtained at the end of a halting computation, 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 multiset obtained during a computation consists of terminal symbols only (yet the system need not have reached a halting configuration);
- (*term*, *d*): the multiset obtained during a computation 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 multiset obtained as a result at the end of a halting computation consists of terminal symbols only;
- (*halt*, *term*, *d*): all three conditions must be fulfilled, i.e., the multiset obtained as a result at the end of a halting computation consists of terminal symbols only, and in any successful derivation at most d steps without applying a rule may occur.

We may also write $(halt, *)$, $(term, *)$, and $(halt, term, *)$ instead of *halt*, *term*, and (*halt*, *term*), respectively.

4.2 Result of computations

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

$$\mathbb{D} = \{(halt, \alpha), (term, \alpha), (halt, term, \alpha) \mid \alpha \in \mathbb{N}_0 \cup \{*\}\}$$

is denoted by $Ps(\Pi_{AB}, gen, \delta, \beta)$, with *gen* indicating that Π_{AB} 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 $N(\Pi_{AB}, gen, \delta, \beta)$.

The families of sets of (k-dimensional) vectors of natural numbers and sets of natural numbers generated by AB-P systems using rules of type X in the derivation mode δ and using the output strategy β are denoted by $Ps(OP(X), gen, \delta, \beta)$ and $N(OP(X), gen, \delta, \beta)$, respectively.

Also P systems with activation and blocking of rules can be considered as accepting mechanisms: we add the input multiset w_0 to w_1 in the initial configuration $C_1 = (w_1)$ thus obtaining $C_1[w_0] = (w_1 w_0)$; the input multiset w_0 is accepted if there exists a halting computation in the derivation mode δ starting from $C_1[w_0]$. We point out that in the accepting case we only consider halting computations, but may also restrict the number of derivation steps where no rule can be applied, i.e., in total we consider the following halting strategies:

$$\mathbb{D}' = \{(halt, \alpha) \mid \alpha \in \mathbb{N}_0 \cup \{*\}\}$$

The families of sets of (k-dimensional) vectors of natural numbers and sets of natural numbers accepted by AB-P systems using rules of type X in the derivation mode δ and using the halting strategy β are denoted by $Ps(OP(X, AB), acc, \delta, \beta)$ and $N(OP(X, AB), acc, \delta, \beta)$, respectively.

If the set B of blocking rules is empty, then the AB-P system is said to be a *P system with activation of rules* (an *A-P system* for short) of type X; the corresponding sets of multisets generated/ accepted as well as the respective families of languages of multisets are denoted in the same way as for AB-P system by just omitting the B.

Moreover, an AB-P system is called an *AkBm-P system* if for all $(p, q, t) \in A$ we have $t \in \{1, \ldots, k\}$, which means that the rules applied in one derivation step activate only the rules which are to be applied in the next k steps, and for all $(p, q, t) \in B$ we have $t \in \{1, \ldots, m\}$, which means that the rules applied in one derivation step can block only the rules which are to be applied in the next m steps. In the case of $k = 1$ or $m = 1$ we only write (p, q) instead of $(p, q, 1)$.

Remark 1 As we need to activate rules for further derivation steps to be able to have unbounded derivations,

only using blocking of rules makes no sense. Yet with implicitly activating all rules in every derivation step, we can only specify those rules to be blocked and in that way get a kind of B-P systems. Yet these are just a special variant of AB-P systems, as instead of only specifying the rules to be blocked in succeeding steps, by the application of any rule we can also activate *all* rules for the next $k+1$ steps where k is the maximum of values t in blocking relations $(p,q,t) \in B$. Hence, in this paper we will not consider this variant further. □

5 Results below *PsRE*

In this section we first give an illustrative example showing that with sequential $A1$-P systems we can get more than semilinear sets. Then we exhibit some relations between $A1$-P systems working in maximal derivation modes, *ET0L*-systems, and P systems with promoters working in maximal derivation modes.

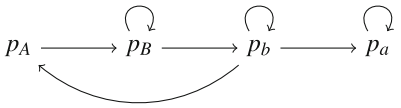
5.1 More than semilinear

It is folklore that sequential P systems with non-cooperative rules (i.e., rules with exactly one symbol on their left-hand side) can only generate semilinear sets, i.e., *PsREG*. Our first example shows that using sequential A1-P systems (i.e., P systems in which the rules in a derivation step activate only the rules that may be applied in the next step) with non-cooperative rules we can already generate non-semilinear sets.

Example 1 The non-semilinear set $L_1 = \{a^n b^m \mid 2 \le n, 2 \le m \le 2^{n-1}\}$ can be generated by a sequential A1-P systems with non-cooperative rules (we remind that this type of rules is abbreviated *ncoo*):

$$\begin{aligned}
\Pi =& (V = \{a,b,A,B\}, T = \{a,b\},\\
& w = Ab, R, \Longrightarrow_{\Pi,sequ}),\\
R =& \{A \to a, b \to BB, A \to AA, B \to b\},\\
\Pi_{AB} =& (\Pi, L, f_L, A, B = \emptyset, L_1, \Longrightarrow_{\Pi_{AB},sequ}),\\
L =& \{p_a, p_b, p_A, p_B\},\\
L_1 =& \{p_A\},\\
f_L =& \{(p_a, A \to a), (p_b, B \to b), (p_A, A \to AA),\\
& (p_B, b \to BB)\},\\
A =& \{(p_a,p_a), (p_b,p_a), (p_b,p_b), (p_b,p_A), (p_A,p_B),\\
& (p_B,p_b), (p_B,p_B)\}.
\end{aligned}$$

The set A of activating rule relations is graphically illustrated in the following figure which shows that this construction is rather similar to using graph control:

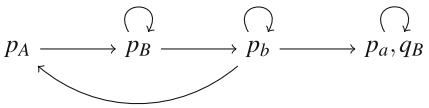


With every adding of one symbol A we may at most double the current number of symbols b using the rules labeled p_B and p_b. At some moment instead of activating p_A by p_b we may switch to p_a whereafter only p_a can be applied any more, yielding a terminal multiset provided all symbols B have been derived to the terminal symbol b before switching from p_b to p_a. In sum, we conclude

$$L_1 \in Ps(OP(ncoo, A1), gen, sequ, \beta)$$

for $\beta \in \{(term,0), (halt,term,0)\}$.

For a similar result with only halting computations we have to guarantee that the multiset obtained at the end of a halting computation is terminal, too. This can be achieved in different ways: in any case, we add an additional label q_B and and add the activations (p_a, q_B) and (q_B, q_B) to A, which results in the following control diagram:



For the rule assigned to the new label q_B we may either take $B \to b$ guaranteeing that all nonterminal symbols not already derived to the terminal symbol b can do this now, *or* $B \to B$ trapping the derivation in an infinite loop if and only if not all nonterminal symbols have already been derived to the terminal symbol b by p_b. In both cases we obtain

$$L_1 \in Ps(OP(ncoo, A1), gen, sequ, \beta)$$

for $\beta \in \{(halt,0), (halt,term,0)\}$, too. □

In the following proofs we will simplify the notation for AB-P systems by writing labeled rules as $p : r$ instead of first listing all rules r in the underlying P system Π and then in Π_{AB} listing the labels p as well as finally defining the function f_L by listing all pairs (p,r). In a shorter way, the whole AB-P system then can be written as

$$\Pi_{AB} = \left(V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB},\delta}\right)$$

with R already containing the labeled rules.

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

$$PsREG \subsetneq Ps(OP(ncoo, A1), gen, sequ, \beta).$$

Proof Let $G = (V, T, P, S)$ be a regular grammar where V is the total alphabet, T the terminal alphabet, $N := V \backslash T$ the set of nonterminal symbols, P a finite set of regular rules over N and T, and $S \in N$ the start symbol. We assume the

rules in P to be of the forms $A \to bC$ or $A \to \lambda$with $A, C \in N$ and $b \in T$. For all $X \in N$, let P_X denote the set of all rules from P with X on the left-hand side. For each rule $r = A \to bC$, let $rhs(r)$ denote the nonterminal symbol C on the right-hand side of the rule.

The set of terminal multisets generated by G can be generated by a sequential A1-P system with non-cooperative rules:

$$\begin{aligned}\Pi_A &= \big(V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_A, sequ}\big)\\ R &= \{p_r : r \mid r \in P\} \cup \{p_X : X \to X \mid X \in N\},\\ L_1 &= \{p_r \mid r \in P_S\},\\ A &= \big\{(p_r, p_q) \mid q \in P_{rhs(r)}\big\}\\ &\quad \cup \{(p_r, p_X) \mid r \in P,\ X \in N\} \cup \{(p_X, p_X) \mid X \in N\}.\end{aligned}$$

With the application of a labeled rule $p_r : r$ for the next step of the derivation all rules are activated which contain the nonterminal symbol on the right-hand side of r on their left-hand side with the activation relations $(p_r, p_q),\ q \in P_{rhs(r)}$. If r is of the form $A \to \lambda$, then $P_{rhs(r)}$ is empty, the result is terminal, and the derivation must halt, as also the rules $p_X : X \to X,\ X \in N$ activated by (p_r, p_X) cannot be applied. Hence, we infer

$$Ps(L(G)) = Ps(\Pi_A, gen, sequ, \beta)$$

for $\beta \in \{(halt, 0), (term, 0), (halt, term, 0)\}$.

On the other hand, the rules $p_X : X \to X,\ X \in N$ and the related activation relations are only needed for the output strategy $(halt, 0)$, but they guarantee that a derivation cannot halt as long as the actual multiset is not terminal. Moreover, these rules and the related activation relations can always be omitted if—without loss of generality—we assume that for every nonterminal symbol A there is at least one rule $A \to bC$ or $A \to \lambda$ in P.

The strictness of the inclusion follows from Example 1, which observation completes the proof. □

Using the maximally parallel derivation mode, we can at least simulate $ET0L$-systems:

Theorem 2 *For any* $\beta \in \{(halt, 0), (term, 0), (halt, term, 0)\}$,

$$PsET0L \subseteq Ps(OP(ncoo, A1), gen, max, \beta).$$

Proof The main idea of constructing an equivalent $A1$-P system Π_A for a given $ET0L$-system $G = (V, T, R_1, \ldots, R_m, w)$ follows the idea of the proof for P systems with states, see Alhazov et al. (2015): we use new symbols t_k representing the m tables R_k, $1 \le k \le m$, of the $ET0L$-system to be simulated. Using a rule $t_{i,j} : t_i \to t_j$ then indicates that after the application of table R_i the table R_j is to be used; hence, all rules in R_j as well as all rules $t_{j,k} : t_j \to t_k$ for all k and the final rule $t_{j,e} : t_j \to \lambda$ are activated by corresponding rule relations in A. The final rules $t_{j,e} : t_j \to \lambda$ do not activate any rule, which means that after having applied this rule the computation in the A1-P system Π_A ends.

In order to start correctly, we use an initial symbol t_0 and define

$$L_1 = \{t_{0,k} : t_0 \to t_k \mid 1 \le k \le n\}$$

which allows us to activate the rules for simulating any table T_k.

As in the proof of Theorem 1, the construction described so far only works for the output strategies $\beta \in \{(term, 0), (halt, term, 0)\}$; for $\beta = (halt, 0)$ we again need the additional labeled rules $p_X : X \to X,\ X \in N$, which are to be activated by the final rules $t_{j,e} : t_j \to \lambda$ in order to trap the system in an infinite loop if a final rule is applied when the current multiset is not yet terminal. In any case, the derivations in G using the tables R_i are simulated correctly by the $A1$-P system Π_A using the adequate rule activations, and the terminal results obtained by the $ET0L$-system G interpreted as multisets are the same as the multisets obtained by the $A1$-P system Π_A. Hence, we conclude

$$Ps(L(G)) = Ps(\Pi_A, gen, max, \beta)$$

for $\beta \in \{(halt, 0), (term, 0), (halt, term, 0)\}$. □

Obviously, the same constructions as described in the proof above work if we use the maximal number of rules or the maximal number of objects, hence, we immediately obtain the following results:

Corollary 1 *For any* $\delta \in \{max, max_{rules}, max_{objects}\}$,

$$PsET0L \subseteq Ps(OP(ncoo, A1), gen, \delta, \beta)$$

for any $\beta \in \{(halt, 0), (term, 0), (halt, term, 0)\}$.

The additional control symbols t_i used in the proof of Theorem 2 can also be interpreted as promoters, which idea is essential for proving the following result:

Theorem 3 *For any* $\delta \in \{max, max_{rules}, max_{objects}\}$,

$$\begin{aligned}&Ps(OP(ncoo, A1), gen, \delta, (halt, term, 0))\\ &\quad \subseteq Ps(OP(ncoo, pro_{1,1}), gen, \delta).\end{aligned}$$

Proof Let $\Pi_A = (V, T, w, R, A, L_1, \Longrightarrow_{\Pi_A, \delta})$ be an arbitrary A1-P system. We construct an equivalent P system

$$\Pi_{pro} = \big(V', T, w_1, R', \Longrightarrow_{\Pi_{pro}, \delta}\big)$$

with atomic promoters, i.e., a rule in R' is of the form $(a \to u, \{b\}, \emptyset)$ with $a, b \in V'$ and $u \in V'^*$, which we will write as $a \to u|_b$.

$$V' = V \cup \{p_r \mid r \in R\},$$
$$w_1 = w \prod_{r \in L_1} p_r,$$
$$R = \{a \to u \prod_{(r,q) \in A} p_q|_{p_r} \mid r : a \to u \in R\} \cup \{p_r \to \lambda \mid r \in R\}.$$

The additional work of Π_{pro} consists of keeping track of which rules are allowed to be executed in the next step by generating the corresponding promoters p_q, which are eliminated again in the next derivation step. Both systems stop the derivation when no rule is applicable any more, and only terminal results are extracted. Hence, we conclude

$$Ps(\Pi_A, gen, \delta, (halt, term, 0)) = Ps(\Pi_{pro}, gen, \delta)$$

for any $\delta \in \{max, max_{rules}, max_{objects}\}$.

Putting together the results of Theorem 3 and Corollary 1 we obtain the following result, already shown in Sburlan (2006) for $\delta = max$.

Corollary 2 *For any* $\delta \in \{max, max_{rules}, max_{objects}\}$,

$$PsET0L \subseteq Ps(OP(ncoo, pro_{1,1}), gen, \delta).$$

The construction given in the proof of Theorem 3 can be extended from only activation of rules simulated by atomic promoters to activation and blocking of rules simulated by atomic promoters and inhibitors, provided we still require the system to not allow delays or look-aheads more than one:

Theorem 4 *For any* $\delta \in \{max, max_{rules}, max_{objects}\}$,

$$Ps(OP(ncoo, A1B1), gen, \delta, (halt, term, 0)) \subseteq Ps(OP(ncoo, pro_{1,1}, inh_{1,1}), gen, \delta).$$

Proof Let $\Pi_{AB} = (V, T, w, R, A, L_1, \Longrightarrow_{\Pi_{AB},\delta})$ be an arbitrary AB-P system without delays (which is reflected by the output strategy (*halt*, *term*, 0)) and no look-aheads more than one, i.e., activations and blockings of rules are just for the next step. We construct an equivalent P system

$$\Pi = \left(V', T, w_1, R', \Longrightarrow_{\Pi,\delta}\right)$$

with atomic promoters and inhibitors, i.e., a rule in R' is of the form $(a \to u, \{b\}, \{c\})$ with $a, b, c \in V'$ and $u \in V'^*$, which we will write as $a \to u|_{b,\neg c}$.

For every labeled rule $p_r : r$ we use two control symbols—the promoter p_r and the inhibitor $\bar{p_r}$, respectively, which are eliminated again in the next step. As a technicality we observe that the presence of both promoter p_r and inhibitor $\bar{p_r}$ prevents the rule r from being applied.

$$V' = V \cup \{p_r, \bar{p_r} \mid r \in R\},$$
$$w_1 = w \prod_{r \in L_1} p_r,$$
$$R' = \left\{ a \to u \prod_{(r,q) \in A} p_q \prod_{(r,q) \in B} \bar{p_q} \mid_{p_r, \neg \bar{p_r}} \mid r : a \to u \in R \right\} \cup \{p_r \to \lambda, \bar{p_r} \to \lambda \mid r \in R\}.$$

Π keeps track of which rules are allowed or prohibited to be executed in the next step by generating the corresponding promoters p_q and inhibitors $\bar{p_q}$, which are eliminated again in the next derivation step. Both systems stop the derivation when no rule is applicable any more, and only terminal results are extracted. Hence, we conclude

$$Ps(\Pi_{AB}, gen, \delta, (halt, term, 0)) = Ps(\Pi, gen, \delta)$$

for any $\delta \in \{max, max_{rules}, max_{objects}\}$. □

If we allow arbitrary look-aheads, the construction of the simulating P system with promoters and inhibitors becomes more involved than that given in the proof of Theorem 4, as we have to check that the *AB*-system does not have delays; therefore, in the following proof we will need two promoters of weight one or, equivalently, one promoter of weight two:

Theorem 5 *For any* $\delta \in \{max, max_{rules}, max_{objects}\}$,

$$Ps(OP(ncoo, AB), gen, \delta, (halt, term, 0)) \subseteq Ps(OP(ncoo, pro_{2,1}, inh_{1,1}), gen, \delta)$$

and

$$Ps(OP(ncoo, AB), gen, \delta, (halt, term, 0)) \subseteq Ps(OP(ncoo, pro_{1,2}, inh_{1,1}), gen, \delta).$$

Proof Let $\Pi_{AB} = (V, T, w, R, A, L_1, \Longrightarrow_{\Pi_{AB},\delta})$ be an arbitrary AB-P system without delays (which is reflected by the output strategy (*halt*, *term*, 0)). We construct an equivalent P system

$$\Pi = \left(V', T, w_1, R', \Longrightarrow_{\Pi,\delta}\right)$$

with promoters and inhibitors of type $(ncoo, pro_{2,1}, inh_{1,1})$, i.e., a rule in R' is of the form $(a \to u, \{d, b\}, \{c\})$ with $a, b, c \in V'$, $d \in V' \cup \{\lambda\}$, and $u \in V'^*$, which we will write as $a \to u|_{d,b,\neg c}$.

Let $k = \max\{t \mid (p, q, t) \in A \cup B\}$ be the maximal look-ahead. Then for every labeled rule $p_r : r$ we use control symbols $[p_r, t]$, $[\bar{p_r}, t]$, $0 \le t \le k-1$, which count down t to zero to let them become active as promoter ($[p_r, 0]$) or inhibitor ($[\bar{p_r}, 0]$), respectively, before these promoters and inhibitors are eliminated again. As a technicality we

observe that the presence of both promoter $[p_r, 0]$ and inhibitor $[\bar{p_r}, 0]$ prevents the rule r from being applied. The additional symbol d is only generated if a rule from R is simulated; the necessity of its presence in order to apply a rule in Π guarantees that only derivations in Π_{AB} without delay are simulated in Π.

$$
\begin{aligned}
V' =& V \cup \{d\} \cup \{[p_r, t], [\bar{p_r}, t] \mid r \in R, 0 \le t \le k-1\},\\
w_1 =& wd \prod_{r \in L_1} p_r,\\
R' =& \Bigg\{ a \to du \prod_{(r,q,t) \in A} [p_q, t-1] \\
& \prod_{(r,q,t) \in B} [\bar{p_q}, t-1] \mid_{d,[p_r,0],\neg[\bar{p_r},0]} \mid r : a \to u \in R \Bigg\}\\
& \cup \{[p_r, t] \to [p_r, t-1] \mid_d, \ [\bar{p_r}, t]\\
& \to [\bar{p_r}, t-1] \mid_d \mid r \in R, \ k-1 > t > 0\}\\
& \cup \{[p_r, 0] \to \lambda \mid_d, [\bar{p_r}, 0]\\
& \to \lambda \mid_d \mid r \in R\} \cup \{d \to \lambda\}.
\end{aligned}
$$

Each couple of two atomic promoters d, p_r can be replaced by the corresponding singleton promoter dp_r of weight two, which observation completes the proof. □

6 Computational completeness results

In this section we show that several simple variants of P systems become computationally complete when using the control of activation and blocking of rules if we allow at most delays of one, i.e., in any case, after one derivation step without applying a rule, in the next step a rule has to be applied to continue the derivation.

6.1 Sequential P systems with non-cooperative rules

Theorem 6 *For any* $\beta \in \{(halt, 1), (term, 1), (halt, term, 1)\}$,

$$PsRE = Ps(OP(ncoo, A2B1), gen, sequ, \beta).$$

Moreover,

$$PsRE = Ps(OP(ncoo, A2B1), acc, sequ, (halt, 1)).$$

Proof The proof idea is to show how to simulate register machines. For a given register machine $M = (n, H, R_M, p_0, h)$ we construct an equivalent AB-P system

$$\Pi_{AB} = \left(V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB}, sequ}\right)$$

in the following way: For every label $p \in H \backslash \{h\}$ we use labels $\{l_p, \bar{l}_p, \tilde{l}_p\}$ for an ADD-instruction and labels $\left\{l_p, l'_p, \hat{l}_p, \tilde{l}_p, \bar{l}_p, \right\}$ for a SUB-instruction; for the final instruction $h : \mathtt{HALT}$ we only use the rule $l_h : h \to \lambda$. For any p, we also use the symbols p, p', and for each register r its contents is described by the number of symbols a_r in (the configurations of) Π_{AB}. The starting rule is given by $L_1 = \{l_{p_0}\}$, and we start with $w = p_0$.

An ADD-instruction $p : (\mathtt{ADD}(r), q, s)$ is simulated by the following labeled rules in R and rule relations in A:

1. first step:
   $l_p : p \to p' a_r$ and $(l_p, \bar{l}_p), (l_p, \tilde{l}_p) \in A$;
2. second step:
   $\bar{l}_p : p' \to q, \tilde{l}_p : p' \to s$ and $(\bar{l}_p, l_q), (\tilde{l}_p, l_s) \in A$.

A SUB-instruction $p : (\mathtt{SUB}(r), q, s)$ is simulated by the following labeled rules in R and rule relations in A and B:

1. first step:
   $l_p : p \to p'$ and $(l_p, \hat{l}_p), (l_p, \tilde{l}_p, 2) \in A$;
2. second step:
   $\hat{l}_p : a_r \to \lambda$ and $(\hat{l}_p, \bar{l}_p) \in A$, $(\hat{l}_p, \tilde{l}_p) \in B$;
3. third step:
   $\bar{l}_p : p' \to q, \tilde{l}_p : p' \to s$ and $(\bar{l}_p, l_q), (\tilde{l}_p, l_s) \in A$.

In sum, we obtain the AB-P system Π_{AB} as follows:

$$
\begin{aligned}
\Pi_{AB} =& \left(V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB}, sequ}\right)\\
V =& \{p, p' \mid p \in H \backslash \{h\}\} \cup \{h\} \cup \{a_r \mid 1 \le r \le n\},\\
w =& p_0,\\
L_1 =& \{l_{p_0}\},\\
R =& \{l_p : p \to p' a_r, \ \bar{l}_p : p' \to q, \ \tilde{l}_p : p' \to s \mid\\
& p : (\mathtt{ADD}(r), q, s) \in R_M\}\\
& \cup \{l_p : p \to p', \ \hat{l}_p : a_r \to \lambda, \ \bar{l}_p : p' \to q,\\
& \tilde{l}_p : p' \to s \mid p : (\mathtt{SUB}(r), q, s) \in R_M\}\\
& \cup \{l_h : h \to \lambda\},\\
A =& \{(l_p, \bar{l}_p), \ (l_p, \tilde{l}_p), \ (\bar{l}_p, l_q), (\tilde{l}_p, l_s) \mid\\
& p : (\mathtt{ADD}(r), q, s) \in R_M\}\\
& \cup \{(l_p, \hat{l}_p), \ (l_p, \tilde{l}_p, 2), \ (\hat{l}_p, \bar{l}_p), \ (\bar{l}_p, l_q), \ (\tilde{l}_p, l_s) \mid\\
& p : (\mathtt{SUB}(r), q, s) \in R_M\},\\
B =& \{(\hat{l}_p, \tilde{l}_p) \mid p : (\mathtt{SUB}(r), q, s) \in R_M\}.
\end{aligned}
$$

If the rule $\hat{l}_p : a_r \to \lambda$ can be applied in the second step, it activates $\bar{l}_p$ and at the same time blocks $\tilde{l}_p$, which has been activated in the first simulation step and thus will be applied if the register is empty, i.e., if $\hat{l}_p$ cannot be applied. Only the second step of the simulation of a A SUB-instruction can be a derivation step where no rule is applied, but afterwards, in any case, a rule will be applied again in Π_{AB}, which guarantees that the delay is at most one.

Hence, in the accepting case, the halting strategy (*halt*, 1) is already sufficient. In the generating case, again delay one is enough, and the special features of the underlying register machine guarantee that also the *AB*-P

system Π_{AB} halts with a terminal multiset whenever it halts, and on the other hand, the multiset can only become terminal when the final rule $l_h : h \to \lambda$ is applied, which terminates the derivation in Π_{AB}. □

6.2 P systems working in set-maximally parallel derivation modes

Corollary 3 *For any* $\delta \in \{smax, smax_{rules}, smax_{objects}\}$, *we have the following results*:

$$PsRE = Ps(OP(ncoo, A2B1), gen, \delta, \beta)$$

for any $\beta \in \{(halt, 1), (term, 1), (halt, term, 1)\}$. *Moreover*,

$$PsRE = Ps(OP(ncoo, A2B1), acc, \delta, (halt, 1)).$$

Proof We can use exactly the same construction as in the proof of Theorem 6. The set mode guarantees that the rule $\hat{l}_p : a_r \to \lambda$ in the second step of the simulation of a SUB-instruction can be applied only once. The other arguments remain exactly the same as in that proof. □

6.3 (Purely) Catalytic P systems working in maximally parallel derivation modes

A typical variant of rules in P systems are so-called *catalytic rules* of the form $ca \to cv$, where c is a catalyst, a symbol which never evolves itself, but helps another symbol a to evolve into a multiset v. The type of P systems using only catalytic rules is called *purely catalytic* (abbreviated *pcat*); if both catalytic rules and non-cooperative rules are allowed, we speak of a catalytic P system (abbreviated *cat*). In the description of the families of sets of multisets generated/accepted by such (purely) catalytic P systems the maximal number of catalysts to be used is indicated as a subscript, i.e., we write $pcat_n$ and cat_n.

The following result then is a consequence of the preceding proofs; we emphasize the important fact that even in the purely catalytic case only *one* catalyst is needed.The bracket notation $[p]cat$ indicates that the type is either *cat* or *pcat*.

Corollary 4 *For any* $\delta \in \{max, max_{rules}, max_{objects}\}$, *we have the following results*:

$$PsRE = Ps(OP([p]cat_1, A2B1), gen, \delta, \beta)$$

for any $\beta \in \{(halt, 1), (term, 1), (halt, term, 1)\}$. *Moreover*,

$$PsRE = Ps(OP([p]cat_1, A2B1), acc, \delta, (halt, 1)).$$

Proof Looking carefully into the proof of Theorem 6, we see that the only rules where the set mode would be needed are those of the form $\hat{l}_p : a_r \to \lambda$. Using just one catalyst c, we can use the rules $\hat{l}_p : ca_r \to c$ instead. The remaining details of the proof of Theorem 6 can remain as they are for the *catalytic* case.

For the *purely catalytic* case, we observe the astonishing fact that, as we are following the construction of a sequential P system, the same catalyst c can also be used for all the other rules; for example, we take $l_p : cp \to cp'$ instead of $l_p : p \to p'$. These observations complete the proof. □

6.4 P systems with insertion and deletion rules

In this section we return to a sequential simulation of register machines using insertion and deletion rules; yet it turns out that the same construction also works for the set-maximally parallel derivation modes, as we have already seen in the case of non-cooperative rules. The type of rules with insertion and deletion of singleton symbols is denoted by *ID*.

Theorem 7 *For any* $\delta \in \{sequ, smax, smax_{rules}, smax_{objects}\}$, *we have the following results*:

$$PsRE = Ps(OP(ID, A2B1), gen, \delta, \beta)$$

for any $\beta \in \{(halt, 1), (term, 1), (halt, term, 1)\}$. *Moreover*,

$$PsRE = Ps(OP(ID, A2B1), acc, \delta, (halt, 1)).$$

Proof The proof idea again is to show how to simulate register machines. For a given register machine $M = (n, H, R_M, p_0, h)$ we construct an equivalent AB-P system

$$\Pi_{AB} = \left(V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB}, \delta}\right)$$

with insertion and deletion rules; as symbols we only need the symbols a_r for representing the contents of registers r, $1 \le r \le n$. We mention that the HALT-instruction is never activated.

$$\begin{aligned}
\Pi_{AB} &= \left(V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB}, \delta}\right)\\
V &= \{a_r \mid 1 \le r \le n\},\\
w &= \lambda,\\
L_1 &= \{l_{p_0}\},\\
R &= \{l_p : I(a_r) \mid p : (\texttt{ADD}(r), q, s) \in R_M\}\\
&\quad \cup \{l_p : I(a_r),\ \hat{l}_p : D(a_r),\ \tilde{l}_p : D(a_r) \mid p : (\texttt{SUB}(r), q, s) \in R_M\},\\
A &= \{(l_p, q) \mid p : (\texttt{ADD}(r), q, s) \in R_M,\ q \ne h\}\\
&\quad \cup \{(l_p, s) \mid p : (\texttt{ADD}(r), q, s) \in R_M,\ s \ne h\}\\
&\quad \cup \{(l_p, \hat{l}_p), (\hat{l}_p, \tilde{l}_p) \mid p : (\texttt{SUB}(r), q, s) \in R_M\}\\
&\quad \cup \{(\hat{l}_p, l_s, 2) \mid p : (\texttt{SUB}(r), q, s) \in R_M,\ s \ne h\}\\
&\quad \cup \{(\tilde{l}_p, l_q) \mid p : (\texttt{SUB}(r), q, s) \in R_M,\ q \ne h\},\\
B &= \{(\tilde{l}_p, l_s) \mid p : (\texttt{SUB}(r), q, s) \in R_M,\ s \ne h\}.
\end{aligned}$$

The simulation of an ADD-instruction only needs one step with inserting one register symbol a_r using the rule l_p :

$I(a_r)$ and then branching to l_q or l_s in the non-deterministic case or just activating l_q in the deterministic case.

The simulation of a SUB-instruction starts with two steps inserting and immediately afterwards again deleting a register symbol a_r with the only purpose to allow the *AB*-P system Π_{AB} to activate both the decrement and the zero-test case with $(\hat{l}_p, \tilde{l}_p) \in A$ and $(\hat{l}_p, l_s, 2) \in A$, respectively. If the rule $\tilde{l}_p : D(a_r)$ can be applied in the third step, it activates l_q and at the same time blocks l_s, which has been activated in the previous simulation step and thus will be applied if the register is empty, i.e., if $\tilde{l}_p$ cannot be applied.

In the *AB*-P system defined above, only the third step of the simulation of a SUB-instruction can be a derivation step where no rule is applied, but afterwards, in any case, a rule will be applied again in Π_{AB}, which guarantees that the delay is at most one.

Hence, in the accepting case, the halting strategy (*halt*, 1) is already sufficient. In the generating case, again delay one is enough, and the special features of the underlying register machine guarantee that also the *AB*-P system Π_{AB} halts with a terminal multiset whenever it halts, and on the other hand, the multiset can only become terminal when no rule is activated any more with M having reached the final label h, which also terminates the derivation in Π_{AB}. □

Remark 2 We emphasize that the construction in the preceding proof uses the minimal amount of symbols, i.e., one symbol for each register. The whole control just works with the activation and blocking of rules. □

Remark 3 We finally observe that all proof constructions given in this section provide *deterministic* simulations of deterministic register machines, which is an important feature for the discussions in Sect. 8. □

7 P systems using backwards activation and blocking of rules

The definition of AB-P systems given in Sect. 4 can be extended by allowing the relations in A and B to be of the form (r_j, r, t) with t possibly also being a negative integer. In that way rules can be activated or blocked in previous steps.

A conservative semantics for this extension is calling a derivation

$$w_0 \Longrightarrow_{\Pi_{AB},\delta} w_1 \Longrightarrow_{\Pi_{AB},\delta} \ldots w_n$$

to be *consistent* if and only if the available sets of rules for previous steps are not changed by having rules activated or blocked backwards in time.

In that way, at least for computationally complete AB-P systems, no increase in the computational power is obtained, as this condition can be checked by any computationally complete device, for example, a Turing machine.

A very special case is to allow $t = 0$, but not to insist on consistency (consistency for $t = 0$ would mean that the set of rules available for the current step does not change when considering activations and blockings for $t = 0$). As long as only activations of rules are considered, this only opens the field for probably new rules to become applicable at the same step, but at the end the whole set of rules available for being applied will be fixed. The situation changes if we also allow blocking of rules, as in this case with the application of some rules other ones having been applicable could be blocked. In the simplest case, consider $(p, p, 0) \in B$, i.e., a rule when being applied is blocking itself to be applied. On the other hand, starting with a set of activated rules, only a finite number of sets of rules eventually to be applied in the current derivation step in a non-conflicting way has to be checked to find out all possible continuations of the ongoing derivation. Hence, in the following section we shall focus especially on activations of rules backwards in time, i.e., on activations $(p, q, -t)$ for $t > 0$. As a special technical detail we mention that activations going back behind the first step just are ignored.

8 Going beyond Turing

We are now discussing how to "go beyond Turing" by using a less conservative semantics for activating (and/or blocking) the rules in preceding derivation steps.

The main idea is to consider infinite computations on given finite multisets —compare this with the idea of red-green Turing machines, see van Leeuwen and Wiedermann (2012), and of red-green register machines, see Aman et al. (2014).

There are several ways to look at these infinite computations and the development of the configurations, yet we have in mind the one based on the ideas elaborated in Freund et al. (2015): we consider sequences of computations where every computation in this sequence the evolution of configurations starts again from the beginning with the initial activations and blockings, but also takes into the account the activations and blockings of rules including the backwards signals obtained in the previous evolution of configurations. We call such an infinite sequence of computations *valid* if each prefix of length n of the computation becomes stable, i.e., for every k with $1 \leq k \leq n$ neither the kth configuration itself nor the set of rules applicable in the kth computation step change any more. We consider the infinite sequence of

stable configurations obtained in this way as the final computation on the given input.

To make such a process of getting an infinite sequence of computations easier to be described, we consider the following procedure: the first computation makes only one step, the second one makes two derivation steps, ..., the nth computation starting with the initial configuration and using the initial activations and blockings enlarged with the actual activations and blockings of rules including the backwards signals obtained in the previous computation makes n derivation steps, ... We emphasize that all activations and blockings obtained in the nth computation of the given deterministic P system are taken over for the next computation.

Remark 4 That the condition for an infinite sequence of computations as defined above to be valid is a Π_3-condition shortly can be argued as follows: Any valid infinite sequence of computations of a deterministic P system with activations and blockings of rules can be encoded as an ω-word, i.e., as an infinite sequence of encodings of finite configurations and the activated and blocked rules for the next computation. Observe that the number of steps ahead for which these activations and blockings are to be considered is bounded by a constant.

The condition that such an ω-word represents a valid infinite sequence of computations then can be checked by a deterministic Turing machine: the deterministic derivation in a deterministic P system with activations and blockings of rules is simulated by a deterministic Turing machine, which goes back on its tape to the encoding of a configuration and the multiset of rules to be applied to it whenever one of these two still changes. Then the input ω-word represents a valid sequence of computations of the deterministic P system with activations and blockings of rules if and only if the corresponding Turing machine has a complete non-oscillating run on it (a run is called *complete* when every position on the tape is reached, and it is called *non-oscillating* when every position on the tape is reached only finitely often during the infinite computation).

The family of ω-languages accepted by deterministic Turing machines by complete non-oscillating runs equals Π_3, for example, see Freund and Staiger (2001). Therefore, all ω-languages representing exactly the valid sequences of computations of a deterministic P system with activations and blockings of rules are in Π_3. □

Provided it exists, we can just consider the stable first configuration of a valid infinite sequence of computations to see whether the input has been accepted. This idea can be used for all the computationally complete variants of P systems with activation and blocking of rules considered in this paper in Sect. 6, as according to Remark 3 all of them allow for a deterministic simulation of deterministic register machines.

One interesting construction principle which can be applied for simulating red-green P systems/automata (starting in red) in all these variants is the following:

- in order to even capture sequential P systems with activation and blocking of rules, we expand the times in the rule relations by a factor of two, hence, the original computations will happen in each odd derivation step;
- we use two new symbols YES and NO; in the initial configuration we add the new symbol NO;
- each application of a rule p changing the color from red to green activates the rule $p_Y : NO \rightarrow YES$ by the backwards activation $(p, p_Y, -1)$ (no such rule is allowed to be activated in L_1);
- each application of a rule p changing the color from green to red activates the rule $p_N : YES \rightarrow NO$ by the backwards activation $(p, p_N, -1)$ (no such rule is allowed to be activated in L_1);
- the mind change (change of color) is propagated backwards by using the backwards activation relations $(p_N, p_N, -2)$ and $(p_Y, p_Y, -2)$, respectively;
- these rules (labeled by) p_N and p_Y then are used "backwards" in every even derivation step; the backwards propagation stops when one of these rules is applied in the second derivation step (as already mentioned, we assume that backwards activations of rules have no effect any more if they activate a rule before step 1);
- if the computation of a red-green P automaton stabilizes in green, i.e., no mind (color) change from green to red takes place any more, then, of course, no changes in the second configuration occur any more, i.e., it has become stable and therefore available for "reading out" the result of the computation.

It is interesting to mention that only "backwards " rule activations are used in the algorithm described above, but no "backwards" rule blockings.

We conclude that with every kind of P systems with activation and blocking of rules which allows for the *deterministic* simulation of register machines we can simulate the corresponding variant of red-green P automata which characterize the Σ_2-sets in the Arithmetical Hierarchy (see Budnik 2006), i.e., with such systems we get at least Σ_2; compare this with the results obtained in Freund et al. (2015, 2016). Yet Σ_2 is only a lower bound: as already pointed out in Remark 4, the condition for a sequence of computations to be valid is a Π_3-condition, which implies that the upper bound is Π_3. Hence, one of the most challenging open problems is to find out if we can obtain more than Σ_2.

9 Conclusion

We have considered the concept of regulating the applicability of rules based on the application of rules in the preceding step(s) within a very general model for hierarchical P systems and for the main derivation modes. These concepts of activation and blocking of rules can also be extended in a natural way to the many variants of tissue P systems, i.e., networks of cells where a rule to be applied can affect multiple cells at the same time.

For the maximally parallel derivation modes and P systems with activation and blocking of non-cooperative rules not allowing derivation steps without applying any rule we have established several relations to $ET0L$-systems as well as to P systems with promoters and inhibitors.

Even with the sequential derivation mode and for the set modes of derivation, the resulting computational power already reaches computational completeness even with non-cooperative rules and using activation of rules with look-ahead two and blocking of rules for the next step when we allow at most one derivation step without any rule being applied before in the next step again some rule must be applied for continuing the derivation.

Using a special semantics for activating and/or blocking the rules in preceding derivation steps, we could even show how to "go beyond Turing" with activating rules in preceding derivation steps and to get Σ_2 as a lower bound and Π_3 as an upper bound. Another interesting topic for future research is to investigate how powerful such AB-P systems are in generating ω-strings.

Acknowledgements Open access funding provided by TU Wien (TUW). Rudolf Freund is very grateful for interesting discussions with Ludwig Staiger on the topics elaborated in Sects. 7 and 8.

References

Alhazov A (2004) A note on P systems with activators. In: Păun Gh, Riscos-Núñez A, Romero-Jiménez A, Sancho-Caparrini F (eds) Second brainstorming week on membrane computing, Sevilla, Spain, 2–7 Feb 2004, pp 16–19

Alhazov A, Ivanov S, Rogozhin Yu (2011) Polymorphic P systems. In: Gheorghe M, Hinze T, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing. Lecture notes in computer science, vol 6501. Springer, pp 81–94

Alhazov A, Freund R, Heikenwälder H, Oswald M, Rogozhin Yu, Verlan S (2013) Sequential P systems with regular control. In: Csuhaj-Varjú E, Gheorghe M, Rozenberg G, Salomaa A, Vaszil Gy (eds) Membrane computing—13th international conference, CMC 2012, Budapest, Hungary, 28–31 Aug 2012, Revised selected papers. Lecture notes in computer science, vol 7762. Springer, pp 112–127. https://doi.org/10.1007/978-3-642-36751-9_9

Alhazov A, Freund R, Ivanov S, Oswald M (2015) Observations on P systems with states. In: Gheorghe M, Petre I, Pérez-Jiménez MJ, Rozenberg G, Salomaa A (eds) Multidisciplinary creativity. Hommage to Gheorghe Păun on His 65th Birthday, pp 17–28. Spandugino

Alhazov A, Freund R, Verlan S (2016) P systems working in maximal variants of the set derivation mode. In: Leporati A, Rozenberg G, Salomaa A, Zandron C (eds) Membrane Computing—17th International Conference, CMC 2016, Milan, Italy, 25–29 July 2016, Revised selected papers. Lecture Notes in Computer Science, vol 10105. Springer, pp 83–102

Alhazov A, Freund R, Ivanov S (2017) P systems with randomized right-hand sides of rules. In: Graciani C, Păun Gh, Riscos-Núñez A, Valencia-Cabrera L (eds) 15th brainstorming week on membrane computing, Sevilla, Spain, Jan 31–Feb 5 2017, pp 13–42

Alhazov A, Freund R, Ivanov S (2018a) Hierarchical P systems with randomized right-hand sides of rules. In: Gheorghe M, Rozenberg G, Salomaa A, Zandron C (eds) Membrane computing—18th international conference, CMC 2017, Bradford, UK, 25–28 July 2017, Revised selected papers. Lecture notes in computer science, vol 10725. Springer, pp 15–39. https://doi.org/10.1007/978-3-319-73359-3

Alhazov A, Freund R, Ivanov S (2018b) Introducing the concept of activation and blocking of rules in the general framework for regulated rewriting in sequential grammars. In: Orellana-Martín D, Păun Gh, Riscos-Núñez A, Valencia-Cabrera L (eds) Proceedings of the 16th brainstorming week on membrane computing, Sevilla, Spain, Jan 30–Feb 2 2018, pp 1–22

Alhazov A, Freund R, Ivanov S (2018c) One-membrane P systems with activation and blocking of rules. In: Orellana-Martín D, Păun Gh, Riscos-Núñez A, Valencia-Cabrera L (eds) Proceedings of the 16th brainstorming week on membrane computing, Jan 30–Feb 2 2018, Sevilla, Spain, pp 23–38

Alhazov A, Freund R, Ivanov S (2018d) P systems with activation and blocking of rules. In: Stepney S, Verlan S (eds) Unconventional computation and natural computation—17th international conference, UCNC 2018, Fontainebleau, France, 25–29 June 2018, Proceedings. Lecture notes in computer science, vol 10867. Springer, pp 1–15. https://doi.org/10.1007/978-3-319-92435-9

Alhazov A, Freund R, Ivanov S (2018e) Sequential grammars with activation and blocking of rules. In: Durand-Lose J, Verlan S (eds) Machines, Computations, and Universality—8th international conference, MCU 2018, Fontainebleau, France, 28–30 June 2018, Proceedings. Lecture notes in computer science, vol 10881. Springer, pp 51–68. https://doi.org/10.1007/978-3-319-92402-1

Aman B, Csuhaj-Varjú E, Freund R (2014) Red-green P automata. In: Gheorghe M, Rozenberg G, Salomaa A, Sosík P, Zandron C (eds) Membrane computing—15th international conference, CMC 2014, Prague, Czech Republic, 20–22 Aug 2014, Revised selected papers. Lecture notes in computer science, vol 8961. Springer, pp 139–157. https://doi.org/10.1007/978-3-319-14370-5

Arroyo F, Baranda AV, Castellanos J, Păun Gh (2002) Membrane computing: the power of (rule) creation. J Univers Comput Sci 8:369–381

Budnik P (2006) What is and what will be. Mountain Math Software, Los Gatos

Calude CS, Păun Gh (2004) Bio-steps beyond Turing. BioSystems 77(1–3):175–194

Cavaliere M, Genova D (2004) P systems with symport/antiport of rules. In: Păun Gh, Riscos-Núñez A, Romero-Jiménez A, Sancho-Caparrini F (eds) 2nd Brainstorming week on membrane computing, Sevilla, Spain, 2–7 Feb 2004, pp 102–116

Cavaliere M, Ionescu M, Ishdorj TO (2004) Inhibiting/de-inhibiting rules in P systems. In: Pre-proceedings of the 5th workshop on membrane computing (WMC5), Milano, Italy, June 2004, pp 174–183

Cavaliere M, Freund R, Oswald M, Sburlan D (2007) Multiset random context grammars, checkers, and transducers. Theor Comput Sci 372(2–3):136–151

Dassow J, Păun Gh (1989) Regulated rewriting in formal language theory. Springer, Berlin

Freund R (1994) Control mechanisms on #-context-free array grammars. In: Păun Gh (ed) Mathematical aspects of natural and formal languages. World Scientific Publ, Singapore, pp 97–137

Freund R (1999) Generalized P-systems. In: Ciobanu G, Păun Gh (eds) Fundamentals of computation theory, 12th international symposium, FCT '99, Iaşi, Romania, Aug 30–Sept 3 1999, Proceedings. Lecture notes in computer science, vol 1684. Springer, pp 281–292

Freund R (2005) P systems working in the sequential mode on arrays and strings. Int J Found Comput Sci 16(4):663–682. https://doi.org/10.1142/S0129054105003224

Freund R (2016) P automata: new ideas and results. In: Bordihn H, Freund R, Nagy B, Vaszil Gy (eds) 8th workshop on non-classical models of automata and applications, NCMA 2016, Debrecen, Hungary, 29–30 Aug 2016. Proceedings, books@ocg.at, vol 321, pp 13–40. Österreichische Computer Gesellschaft

Freund R, Staiger L (2001) Acceptance of ω-languages by communicating deterministic Turing machines. In: Martín-Vide C, Mitrana V (eds) Where mathematics, computer science, linguistics and biology meet: essays in honour of Gheorghe Păun. Kluwer Academic Publishers, Dordrecht, pp 115–126

Freund R, Verlan S (2007) A formal framework for static (tissue) P systems. In: Eleftherakis G, Kefalas P, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing. Lecture notes in computer science, vol 4860. Springer, pp 271–284. https://doi.org/10.1007/978-3-540-77312-2_17

Freund R, Oswald M, Staiger L (2004) ω-P automata with communication rules. In: Martín-Vide C, Mauri G, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing. Lecture notes in computer science, vol 2933. Springer, pp 203–217. https://doi.org/10.1007/978-3-540-24619-0_15

Freund R, Kogler M, Oswald M (2011) A general framework for regulated rewriting based on the applicability of rules. In: Kelemen J, Kelemenová A (eds) Computation, cooperation, and life—essays dedicated to Gheorghe Păun on the Occasion of His 60th Birthday. Lecture notes in computer science, vol 6610. Springer, pp 35–53

Freund R, Leporati A, Mauri G, Porreca AE, Verlan S, Zandron C (2014) Flattening in (tissue) P systems. In: Alhazov A, Cojocaru S, Gheorghe M, Rogozhin Yu, Rozenberg G, Salomaa A (eds) Membrane computing. Lecture notes in computer science, vol 8340. Springer, pp 173–188

Freund R, Ivanov S, Staiger L (2015) Going beyond Turing with P automata: Partial adult halting and regular observer ω-languages. In: Calude CS, Dinneen MJ (eds.) Unconventional computation and natural computation – 14th international conference, UCNC 2015, Auckland, New Zealand, Aug 30–Sept 3 2015, Proceedings. Lecture notes in computer science, vol 9252. Springer, pp 169–180

Freund R, Ivanov S, Staiger L (2016) Going beyond Turing with P automata: regular observer ω-languages and partial adult halting. IJUC 12(1):51–69

Ivanov S (2014) Polymorphic P systems with non-cooperative rules and no ingredients. In: Gheorghe M, Rozenberg G, Salomaa A, Sosík P, Zandron C (eds) Membrane computing—15th international conference, CMC 2014, Prague, Czech Republic, 20–22 Aug 2014, Revised selected papers. Lecture notes in computer science, vol 8961. Springer, pp 258–273

Kudlek M, Martín-Vide C, Păun Gh (2001) Toward a formal macroset theory. In: Calude CS, Păun Gh, Rozenberg G, Salomaa A (eds) Multiset processing—mathematical, computer science and molecular computing points of view. Lecture notes in computer science, vol 2235. Springer, pp 123–134

Minsky ML (1967) Computation: finite and infinite machines. Prentice-Hall Inc., Upper Saddle River

Păun Gh (1998) Computing with membranes. J Comput Syst Sci 61:108–143

Păun Gh, Rozenberg G, Salomaa A (2010) Oxford handbook of membrane computing. Oxford University Press Inc., New York

Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages. Springer, New York

Sburlan D (2006) Further results on P systems with promoters/inhibitors. Int J Found Comput Sci (Special Volume: Membrane Computing) 17(01):205–221. https://doi.org/10.1142/S0129054106003772

Sosík P, Valík O (2006) On evolutionary lineages of membrane systems. In: Freund R, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing. Lecture notes in computer science, vol 3850. Springer, pp 67–78. https://doi.org/10.1007/11603047_5

van Leeuwen J, Wiedermann J (2012) Computation as an unbounded process. Theor Comput Sci 429:202–212. https://doi.org/10.1016/j.tcs.2011.12.040