P Systems with Activation and Blocking of Rules

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

[1] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 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

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 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 derivation step, we can even "go beyond Turing".

1 Introduction

Originally founded by Gheorghe Păun in 1998, see [30], membrane systems, now 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 membranes. Quite often, the objects are plain symbols coming from a finite alphabet, i.e., multisets (for basic results on multiset computing, for example, see [27]), but P systems operating on more complex objects (e.g., strings, arrays) are often considered, too, for instance, see [18].

A comprehensive overview of different flavors of membrane systems and their expressive power is given in the handbook which appeared in 2010, see [31]. For a state of the art snapshot of the domain, we refer the reader to the P systems website [34] as well as to the Bulletin of the International Membrane Computing Society [33].

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).

Nearly thirty years ago, the monograph on regulated rewriting by Jürgen Dassow and Gheorghe Păun [15] 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 [16]. As exhibited in [22], 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 applicability of rules. Also in the field of P systems [31,34] where mainly multisets have been considered, such regulating mechanisms were used, e.g., see [12].

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 [17]; 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 [9], activators [1], inhibiting/deinhibiting rules [14], and symport/antiport *of rules* [13]. One of the more recent developments in this direction are *polymorphic P systems* [5,7,26], 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 [2,3], 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 [4], 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 [22], 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.

In the following 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. 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 [19]).

Various possibilities of how one may "go beyond Turing" are discussed in [28], for example, the definitions and results for red-green Turing machines can be found there. In [8] 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 [11,32]. Computations with infinite words by P automata were investigated in [24].

In [20,21], 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 the evolution of time lines described by P systems – at some moment, a specific part (a succession of configurations) of the evolving time lines, for example, the part describing time 0, shall not change any more.

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. Such an infinite computation is called *valid* if each prefix of the computation 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 derivation steps allows us to take the infinite sequence of stable configurations obtained in this way as the final computation on the given input and – provided it 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 part of the evolving time lines, now the part describing time 1, requiring that it should not change any more, but now also requesting that the whole computation should converge.

In the following section, we recall some notions from formal language theory as well as the main definitions of the general framework for P systems working under different derivation modes, see [25]. 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. In Sect. 4 we prove first results only using activation of rules. Computational completeness results using both activation and blocking of rules are established in Sect. 5. Then we extend our systems by allowing activation and blocking of rules in previous derivation steps in Sect. 6, and finally even discuss how to "go beyond Turing" in Sect. 7. 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. 8.

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 [25]. Then we define the new variant of controlling rule applications in P systems by activation and blocking of rules induced by the application of rules in a derivation step.

2.1 Preliminaries

The set of integers is denoted by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{N}_0$, and the set of positive integers (natural numbers) 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; $V^* \setminus \{\lambda\}$ 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 $(|x|_{a_1}, \ldots, |x|_{a_n})$. 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 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 set of all finite multisets over an alphabet V is denoted by V^o.

For more details of formal language theory the reader is referred to the monographs and handbooks in this area [15,31].

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, 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:

- $p : (\texttt{ADD}(r), q, s)$ 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,
- $p : (\texttt{SUB}(r), q, s)$ 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;
- $h :$ HALT halts the machine.

M is called deterministic if in all ADD-instructions $p : (\texttt{ADD}(r), q, s)$, it holds that $q = s$; in this case we write $p : (\texttt{ADD}(r), q)$. Deterministic register machines can accept all recursively enumerable sets of vectors of natural numbers with k components using precisely $k + 2$ registers, see [29].

2.3 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 [25].

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

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

- 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.

The symbol X in "rules of type X" may stand for "evolution", "communication", "membrane evolution", etc.

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.

The *language generated by* Π is the set of all terminal multisets which can be obtained in the output membrane f starting from the initial configuration $C_1 = (w_1, \ldots, w_m)$ using the derivation mode δ in a halting computation, i.e.,

$$L_{gen,\delta}(\Pi) = \left\{ C(f) \in T^\circ \mid C_1 \overset{*}{\Longrightarrow}_{\Pi,\delta} C \wedge \neg\exists C' : \; C \Longrightarrow_{\Pi,\delta} C' \right\},$$

where $C(f)$ stands for the multiset contained in the output membrane f of the configuration C. The configuration C is halting, i.e., no further configuration C' can be derived from it.

The family of languages of multisets generated by P systems of type X with at most n membranes in the derivation mode δ is denoted by $Ps_{gen,\delta}OP_n(X)$.

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 $C_1[w_0]$, i.e.,

$$L_{acc,\delta}(\Pi) = \left\{ w_0 \in T^\circ \mid \exists C : \; \left(C_1[w_0] \overset{*}{\Longrightarrow}_{\Pi,\delta} C \wedge \neg\exists C' : \; C \Longrightarrow_{\Pi,\delta} C' \right) \right\}.$$

The family of languages of multisets accepted by P systems of type X with at most n membranes in the derivation mode δ is denoted by $Ps_{acc,\delta}OP_n(X)$.

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 [25] for formal definitions):

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

In [6], 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).

As many variants of P systems can be *flattened* to only one membrane, see [23], throughout the paper we 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

$$\Pi = (V, T, w_1, R, \Longrightarrow_{\Pi,\delta}).$$

3 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 (generating) P systems.

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

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

where $\Pi = (V, T, w, R, \Longrightarrow_{\Pi,\delta})$ is a P system of type X, L is a finite set of labels with each label having assigned one rule from R by the function f_L, A, B are finite subsets of $L \times L \times 2^{\mathbb{N}}$, 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 being a finite subset of $\mathbb{N}$; the elements of T indicate how many 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, $P \subseteq R$, 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 $w_i \in O$, $0 \leq i \leq n$, 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 R_k denoting the set of rules applied to w_k in the k-th derivation step, for every i, $0 \leq i < n$, 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 $i - j \in T$, which means that the application of a rule r_j in the j-th derivation step has activated rule r probably to be applied in the i-th derivation step, and there is no rule relation $(r_j, r, T) \in B$ such that $i - j \in T$, which means that the application of the rule r_j in the j-th derivation step would block rule r to be applied in the i-th derivation step, **or**
- P_i is empty, i.e., no rule r is activated to be applied i-th 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 language generated/accepted by the AB-P system Π_{AB}, now using the derivation relation $\Longrightarrow_{\Pi_{AB},\delta}$ instead of $\Longrightarrow_{\Pi,\delta}$.

The families of languages of multisets generated/accepted by AB-P systems of type X in the derivation mode δ (in only one membrane) is denoted by $Ps_{\gamma,\delta}OP(X, AB)$, $\gamma \in \{gen, acc\}$.

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. In this case we will usually not allow the second case in a derivation of the A-P system that in a derivation step no rule is activated to be applied. Moreover, an A-P system is called an *A1-P system* if for all $(p, q, T) \in A$ we have $T = \{1\}$, which means that the rules applied in one derivation step

activate only the rules which can be applied in the next step; in this case we only write (p, q) instead of (p, q, T).

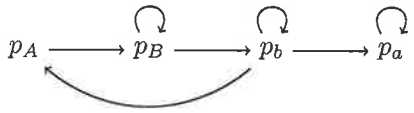
4 Results Below *PsRE*

It is folklore that sequential P systems with non-cooperative rules (i.e., rules with exactly one symbol in their left-hand side) can only generate semilinear sets, i.e., *PsREG*. Our first example shows that using sequential A1-P systems with non-cooperative rules we can generate non-semilinear sets.

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

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

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 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 $\Pi_{AB} = (V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB}, \delta})$ with R already containing the labeled rules.

Corollary 1. $PsREG \subsetneq Ps_{gen,sequ}OP\,(ncoo, A1)$

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

Theorem 1. $PsET0L \subseteq Ps_{gen,max}OP\,(ncoo, A1)$

Proof (Sketch). Like in P systems with states, see [5], we can use new symbols t_k representing the n tables T_k of the extended tabled Lindenmayer system to be simulated. Using a rule $t_{i,j} : t_i \to t_j$ then indicates that after the application of table T_i the table T_j is to be used; hence, all rules in T_j as well as all rules $t_{j,k} : t_j \to t_k$ for all k and $t_{j,e} : t_j \to \lambda$ are activated by corresponding rule relations in A. The 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 ends.

In order to start correctly, we use an initial symbol t_0 and define $L_1 = \{t_{0,k} : t_0 \to t_k \mid 1 \leq k \leq n\}$ which allows us to activate the rules for simulating any table T_k. □

5 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.

5.1 Sequential P Systems with Non-cooperative Rules

Theorem 2. $PsRE = Ps_{\gamma,sequ}OP\,(ncoo, AB)$ *for* $\gamma \in \{gen, acc\}$.

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

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

in the following way: For every label $p \in H \setminus \{h\}$ we use labels $\left\{l_p, \hat{l}_p, \tilde{l}_p\right\}$ for an ADD-instruction and labels $\left\{l_p, l'_p, l''_p, \hat{l}_p, \tilde{l}_p, \bar{l}_p, \right\}$ for a SUB-instruction; for the final instruction h : 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 $L_1 = \{l_{p_0}\}$.

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

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

If the rule $\hat{l}_p : a_r \to a_{r,p}$ can be applied in the second step, two steps afterwards 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. □

5.2 P Systems Working in the *smax*-Mode

Theorem 3. *$PsRE = Ps_{\gamma,\delta}OP(ncoo, AB)$ for $\gamma \in \{gen, acc\}$ and any set derivation mode δ from $\{smax, smax_{rules}, smax_{objects}\}$.*

Proof. Again we show how to simulate a register machine $M = (n, H, R_M, p_0, h)$. The equivalent AB-P system $\Pi_{AB} = (V, T, w, R, A, B, L_1, \Longrightarrow_{\Pi_{AB},\delta})$ contains similar ingredients as the one constructed in the proof of Theorem 2; yet the simulation of SUB-instructions now allows us to only use activation and blocking of rules for the next step using the possibility of having several rules applied in parallel:

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

If register r is empty, in the second step only $\tilde{l}_p$ is activated to be applied in the third step; otherwise, the application of $\hat{l}_p$ activates $\bar{l}_p$ and at the same time blocks $\tilde{l}_p$. □

5.3 (Purely) Catalytic P Systems Working in the *max*-Mode

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:

Corollary 2. *For $\gamma \in \{gen, acc\}$ and $\delta \in \{max, max_{rules}, max_{objects}\}$,*
$PsRE = Ps_{\gamma,\delta}OP(pcat_2, AB)$ and
$PsRE = Ps_{\gamma,\delta}OP(cat_1, AB)$.

Proof. Looking carefully into the proof of Theorem 3, we see that the only rules where the set mode is needed are those of the form $\hat{l}_p : a_r \to a_{r,p}$. Using one catalyst c_1, we can use the rules $\hat{l}_p : c_1 a_r \to c_1 a_{r,p}$ instead. The remaining details of the proof of Theorem 3 can remain as they are for the *catalytic* case.

For the *purely catalytic* case, we need a second catalyst c_2 for all the other rules, e.g., we take $l_p : c_2 p \to c_2 p'$ instead of $l_p : p \to p'$. These observations complete the proof. □

6 P Systems Using Backwards Activation and Blocking of Rules

The definition of AB-P systems given in Sect. 3 can be extended by allowing the relations in A and B to be of the form (r_j, r, T) with the finite set T also containing negative integers. 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.

7 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 [28], and of red-green register machines, see [8] – and call such an infinite computation valid if each prefix of the computation becomes stable, i.e., neither the configuration itself nor the set of applicable rules changes any more. We consider the infinite sequence of stable configurations obtained in this way as the final computation on the given input; then – provided it exists – we just consider the stable first configuration 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.

There are several ways to look at these infinite computations and the development of the configurations, yet we have in mind the following, based on the ideas elaborated in [20]: we consider the time line of evolutions of the configurations where in each step every configuration evolves again according to the actual activations and blockings of rules including the backwards signals.

One interesting construction principle which may be applied for simulating red-green P systems/automata (starting in red) in all these variants can be 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 rule p changing the color from red to green activates the rule $p_Y : NO \to YES$ by the backwards activation $(p, p_Y, -1)$ (no such rule is allowed to be activated in L_1);

- each rule p changing the color from green to red activates the rule $p_N : YES \to NO$ 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 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 a convention, backwards activation rules have no effect any more if they activate a rule before time 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.

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 [10]), i.e., with such systems we at least get Σ_2; compare this with the results obtained in [20,21].

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

8 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.

Especially for the set modes of derivation, the resulting computational power already reaches computational completeness even with non-cooperative rules and using both activation and blocking of rules. 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. An interesting topic for future research is to investigate how powerful such AB-P systems are in generating ω-strings.

Acknowledgements. The authors are very grateful for the useful comments of the referees.

References

1. Alhazov, A.: 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 February 2004, pp. 16–19 (2004)
2. Alhazov, A., Freund, R., Ivanov, S.: P systems with randomized right-hand sides of rules. In: 15th Brainstorming Week on Membrane Computing, Sevilla, Spain, January 31–February 5 2017 (2017)
3. Alhazov, A., Freund, R., Ivanov, S.: Hierarchical P systems with randomized right-hand sides of rules. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) CMC 2017. LNCS, vol. 10725, pp. 15–39. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73359-3_2
4. Alhazov, A., Freund, R., Ivanov, S.: Systems with activation and blocking of rules (2018)
5. Alhazov, A., Freund, R., Ivanov, S., Oswald, M.: Observations on P systems with states. In: Gheorghe, M., Petre, I., Pérez-Jiménez, M.J., Rozenberg, G., Salomaa, A. (eds.) Multidisciplinary Creativity. Hommage to Gheorghe Păun on His 65th Birthday, Spandugino (2015)
6. 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://doi.org/10.1007/978-3-319-54072-6_6
7. Alhazov, A., Ivanov, S., Rogozhin, Yu.: Polymorphic P systems. In: Gheorghe, M., Hinze, T., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) CMC 2010. LNCS, vol. 6501, pp. 81–94. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-18123-8_9
8. 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
9. Arroyo, F., Baranda, A.V., Castellanos, J., Păun, Gh.: Membrane computing: the power of (rule) creation. J. Univ. Comput. Sci. 8, 369–381 (2002)
10. Budnik, P.: What Is and What Will Be. Mountain Math Software (2006)
11. Calude, C.S., Păun, Gh.: Bio-steps beyond turing. Biosystems **77**(1–3), 175–194 (2004)
12. Cavaliere, M., Freund, R., Oswald, M., Sburlan, D.: Multiset random context grammars, checkers, and transducers. Theor. Comput. Sci. **372**(2–3), 136–151 (2007)
13. Cavaliere, M., Genova, D.: P systems with symport/antiport of rules. In: Păun, Gh., Riscos-Núñez, A., Romero-Jiménez, A., Sancho-Caparrini, F. (eds.) Second Brainstorming Week on Membrane Computing, Sevilla, Spain, 2–7 February 2004, pp. 102–116 (2004)
14. Cavaliere, M., Ionescu, M., Ishdorj, T.O.: Inhibiting/de-inhibiting rules in P systems. In: Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5), Milano, Italy, June 2004, pp. 174–183 (2004)
15. Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
16. Freund, R.: Control mechanisms on #-context-free array grammars. In: Păun, Gh. (ed.) Mathematical Aspects of Natural and Formal Languages, pp. 97–137. World Scientific Publishing, Singapore (1994)

17. Freund, R.: Generalized P-systems. In: Ciobanu, G., Păun, Gh. (eds.) FCT 1999. LNCS, vol. 1684, pp. 281–292. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48321-7_23
18. 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
19. Freund, R.: P automata: new ideas and results. In: Bordihn, H., Freund, R., Nagy, B., Vaszil, Gy. (eds.) Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, 29–30 August 2016. Proceedings, books@cg.at, vol. 321, pp. 13–40. ÖsterreichischeComputer Gesellschaft (2016)
20. 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
21. 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)
22. 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
23. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (Tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) CMC 2013. LNCS, vol. 8340, pp. 173–188. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54239-8_13
24. 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
25. Freund, R., Verlan, S.: A formal framework for static (Tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 271–284. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77312-2_17
26. Ivanov, S.: Polymorphic P systems with non-cooperative rules and no ingredients. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 258–273. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14370-5_16
27. 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
28. 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
29. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall Inc, Upper Saddle River (1967)
30. Păun, Gh.: Computing with membranes. J. Comput. Syst. Sci. **61**, 108–143 (1998)

31. Păun, Gh., Rozenberg, G., Salomaa, A.: The Oxford Handbook of Membrane Computing. Oxford University Press Inc, New York (2010)
32. 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
33. Bulletin of the International Membrane Computing Society (IMCS). http://membranecomputing.net/IMCSBulletin/index.php
34. The P Systems Website. http://ppage.psystems.eu/