P Systems Working in Maximal Variants of the Set Derivation Mode

Artiom Alhazov[1], Rudolf Freund[2(✉)], and Sergey Verlan[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 Wien, Austria
rudi@emcc.at

[3] LACL, Université Paris Est – Créteil Val de Marne, 61, av. Général de Gaulle, 94010 Créteil, France
verlan@u-pec.fr

Abstract. In P systems working in the set derivation mode, even in the maximally parallel derivation mode, rules are only applied in at most one copy in each derivation step. We also consider the set mode in the cases of taking those sets of rules with the maximal number of applicable rules or with affecting the maximal number of objects. For many variants of P systems, the computational completeness proofs even literally still hold true for these new set derivation modes. On the other hand, we obtain new results for P systems using target selection for the rules to be chosen together with these set derivation modes.

1 Introduction

Membrane systems with symbol objects are a theoretical framework of parallel distributed multiset processing. Usually, multisets of rules are applied in parallel to the objects in the underlying configuration; for example, in the maximally parallel derivation mode (abbreviated *max*), a non-extendable multiset of rules is applied to the current configuration. In this paper we now consider variants of these derivation modes, where each rule is only used in at most one copy, i.e., we consider sets of rules to be applied in parallel, for example, in the *set-maximally parallel derivation mode* (abbreviated *smax*) we apply non-extendable *sets* of rules, and in two other set derivation modes we apply sets of rules which contain a maximal number of applicable rules (abbreviated $smax_{rules}$) and sets of rules which affect a maximal number of objects (abbreviated $smax_{objects}$).

Taking sets of rules instead of multisets is a quite natural restriction which has already appeared implicitly in [9] as the variant of the min_1-derivation mode where each rule forms its own partition. Other motivations arise when we consider firing a maximal set of transitions in Petri Nets [5,10] or optimizing an implementation of FPGA simulators [19]. A natural question arises concerning the power of set-based modes in contrast to multiset-based ones. In an explicit way, the set derivation mode first was investigated in [13] where the derivation

A. Leporati et al. (Eds.): CMC 2016, LNCS 10105, pp. 83–102, 2017.
DOI: 10.1007/978-3-319-54072-6_6

mode *smax* was called *flat maximally parallel derivation mode*. Yet we here keep the notation of the *set-maximally parallel derivation mode* as we have already used it at the Conference on Membrane Computing 2015. In [13] it was already shown that in some cases the computational completeness results established for the *max*-mode also hold for the flat maximally parallel derivation mode, i.e., for the *smax*-mode.

In this paper we continue this line of research and we show that for several well-known variants of P systems the proofs for computational completeness for *max* can be taken over even literally for *smax* as well as for the derivation modes max_{rules}, $max_{objects}$ and $smax_{rules}$, $smax_{objects}$, where multisets or sets of rules with the maximal number of rules and multisets or sets of rules affecting the maximal number of objects, respectively, are taken into account. For P systems using target selection for the rules to be chosen these set derivation modes yield even stronger new results. Full proofs of some of the results mentioned in this paper and a series of additional results can be found in [4].

2 Definitions

We assume the reader to be familiar with the underlying notions and concepts from formal language theory, e.g., see [7,15].

2.1 Prerequisites

The set of non-negative integers is denoted by $\mathbb{N}$. Given an alphabet V, a finite non-empty set of abstract symbols, the free monoid generated by V under the operation of concatenation is denoted by V^*. The elements of V^* are called strings, the empty string is denoted by λ, and $V^* \setminus \{\lambda\}$ is denoted by V^+. For an arbitrary alphabet $V = \{a_1, \ldots, a_n\}$, the number of occurrences of a symbol a_i in a string x is denoted by $|x|_{a_i}$, while the length of a string x is denoted by $|x| = \sum_{a_i \in V} |x|_{a_i}$. With respect to a specific order on the elements $a_1, \ldots, a_n$ of the alphabet V, the n-tuple $(|x|_{a_1}, \ldots, |x|_{a_n}\}$ is called the Parikh vector of $|x|$.

A finite multiset over an alphabet $V = \{a_1, \ldots, a_n\}$ is a mapping $f : V \to \mathbb{N}$ and can be represented by $\left\langle a_1^{f(a_1)}, \ldots, a_n^{f(a_n)} \right\rangle$ or by any string x for which $(|x|_{a_1}, \ldots, |x|_{a_n}) = (f(a_1), \ldots, f(a_n))$. The set of all multisets over V is denoted by V° or by $Ps\,(V^*)$.

The families of recursively enumerable string languages is denoted by RE. For any family of languages X, $Ps\,(X)$ denotes the set of Parikh sets of the languages in X; if we do not distinguish between different symbols and only consider sets of numbers, we write $N\,(X)$.

2.2 Register Machines

A *register machine* is a tuple $M = (d, B, l_0, l_h, R)$, where d is the number of registers, B is a set of labels, $l_0 \in B$ is the initial label, $l_h \in B$ is the final label, and R is the set of instructions bijectively labeled by elements of B. The instructions of M can be of the following forms:

- $p : (ADD(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq d$.
  Increases the value of register r by one, followed by a non-deterministic jump to instruction p or s. This instruction is usually called *increment*.
- $p : (SUB(r), q, s)$, with $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq d$.
  If the value of register r is zero, then the register machine jumps to instruction s; otherwise, the value of register r is decreased by one, followed by a jump to instruction q. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stops the computation of the register machine.

A *configuration* of a register machine is described by the contents (i.e., by the number stored in the register) of each register and by the current label, which indicates the next instruction to be executed. Computations start by executing the instruction l_0 of R, and terminate with reaching the HALT-instruction l_h.

As is well known (e.g., see [12]), register machines constitute a computationally complete model for generating or accepting sets of (vectors of) non-negative numbers, and thus they are often used as a reference model to be simulated in the area of membrane computing.

For accepting recursively enumerable sets of numbers, deterministic register machines with (at most) three registers are sufficient. In a deterministic register machine, the ADD-instructions are of the form $p : (ADD(r), q)$, with $p \in B \setminus \{l_h\}$, $q \in B$, $1 \leq r \leq d$.

In the succeeding proofs, for a register machine $M = (d, B, l_0, l_h, R)$ we will denote the sets of deterministic ADD-instructions, of non-deterministic ADD-instructions, and of SUB-instructions in R by $ADD^1(R)$, $ADD^2(R)$, and $SUB(R)$, respectively. Moreover, by B_{ADD} we denote the set of labels for ADD-instructions, and by B_{SUB} we denote the set of labels for SUB-instructions of M in R.

Finally, without loss of generality, for a register machine $M = (d, B, l_0, l_h, R)$ with d registers and $m \leq d$ decrementable registers, we will assume that the following conditions hold: the output registers are $m + 1, \cdots, d$, and they are never decremented; moreover, the decrementable registers $1, \cdots, m$ are empty in any reachable halting configuration.

3 Variants of P Systems

In this section we recall the well-known definitions of several variants of P systems as well as some variants of derivation modes and also introduce the variants of set derivation modes considered in the following.

For all the notions and results not referred to otherwise we refer the reader to the Handbook of Membrane Computing [14] as well as to the webpage [18] of P systems.

A (cell-like) P system is a construct

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f_O, f_I) \quad \text{where}$$

- O is the alphabet of objects,
- $C \subset O$ is the set of catalysts,
- μ is the membrane structure (with m membranes, labeled by 1 to m),
- $w_1, \ldots, w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation,
- $R_1, \ldots, R_m$ are finite sets of rules, associated with the regions of μ,
- f_O is the label of the membrane region from which the outputs are taken (in the generative case),
- f_I is the label of the membrane region where the inputs are put at the beginning of a computation (in the accepting case).

$f_O = 0/f_I = 0$ indicates that the output/input is taken from the environment. If f_O and f_I indicate the same label, we only write f for both labels.

If a rule $u \to v$ has at least two objects in u, then it is called *cooperative*, otherwise it is called *non-cooperative*. *Catalytic rules* are of the form $ca \to cv$, where $c \in C$ is a special object called *catalyst* which never evolves and never passes through a membrane, it just assists object a to evolve to the multiset v.

In *catalytic P systems* we use non-cooperative as well as catalytic rules. In a *purely catalytic P system* we only allow catalytic rules.

We call a P system *simple* if it contains only one membrane (the skin membrane), which also serves as input and output membrane. Only specifying the relevant parts, we then may write $\Pi = (O, C, w_1, R_1)$ where

- O is the alphabet of objects,
- $C \subset O$ is the set of catalysts,
- w_1 is the finite multiset of objects over O present in the skin membrane at the beginning of a computation,
- R_1 is a finite set of rules.

We omit the set C if the P system contains no catalysts.

3.1 Derivation Modes

The definitions and the corresponding notions used in this subsection follow the definitions and notions elaborated in [9]. Given a P system Π, the set of multisets of rules applicable to a configuration C is denoted by $Appl(\Pi, C)$; this set also equals the set $Appl(\Pi, C, asyn)$ of multisets of rules applicable in the *asynchronous derivation mode* (abbreviated *asyn*). The set $Appl(\Pi, C, sequ)$ denotes the set of multisets of rules applicable in the *sequential derivation mode* (abbreviated *sequ*), where in each derivation step exactly one rule is applied.

In the *maximally parallel derivation mode* (abbreviated by *max*), in any computation step of Π we choose a multiset of rules from $\mathcal{R}$ (which is defined as the union of the sets $R_1, \ldots, R_m$; in this context, rules in different rule sets R_i and R_j with $i \neq j$ are always considered to be different rules) in such a way that no further rule can be added to it so that the obtained multiset would still be applicable to the existing objects in the regions $1, \ldots, m$:

$$Appl(\Pi, C, max) = \{R \in Appl(\Pi, C) \mid \text{there is no } R' \in Appl(\Pi, C) \text{ such that } R' \supset R\}$$

The basic set derivation mode is defined as the derivation mode where in each derivation step at most one copy of each rule may be applied in parallel with the other rules; this variant of a basic derivation mode corresponds to the asynchronous mode with the restriction that only those multisets of rules are applicable which contain at most one copy of each rule, i.e., we consider *sets* of rules:

$$Appl(\Pi, C, set) = \{R \in Appl(\Pi, C, asyn) \mid |R|_r \leq 1 \text{ for each } r \in \mathcal{R}\}$$

In the *set-maximally parallel derivation mode* (this derivation mode is abbreviated by *smax* for short), in any computation step of Π we choose a non-extendable multiset R of rules from $Appl(\Pi, C, set)$:

$$Appl(\Pi, C, smax) = \{R \in Appl(\Pi, C, set) \mid \text{there is no } R' \in Appl(\Pi, C, set) \text{ such that } R' \supset R\}$$

The $smax$-derivation mode corresponds to the min_1-mode with the discrete partitioning of rules (each rule forms its own partition), see [9].

As already introduced for multisets of rules in [6], we now consider the variant where the maximal number of rules is chosen. In the derivation mode $max_{rules}max$ only a maximal multiset of rules is allowed to be applied. But it can also be seen as the variant of the basic mode max where we just take a multiset of applicable rules with the maximal number of rules in it, hence, we will also call it the max_{rules} derivation mode. Formally we have:

$$Appl(\Pi, C, max_{rules}) = \{R \in Appl(\Pi, C, asyn) \mid \text{there is no } R' \in Appl(\Pi, C, asyn) \text{ such that } |R'| > |R|\}$$

The derivation mode $max_{rules}smax$ is a special variant where only a maximal set of rules is allowed to be applied. But it can also seen as the variant of the basic set mode where we just take a set of applicable rules with the maximal number of rules in it, hence, we will also call it the $smax_{rules}$ derivation mode. Formally we have:

$$Appl(\Pi, C, smax_{rules}) = \{R \in Appl(\Pi, C, set) \mid \text{there is no } R' \in Appl(\Pi, C, set) \text{ such that } |R'| > |R|\}$$

We also consider the derivation modes $max_{objects}max$ and $max_{objects}smax$ where from the multisets of rules in $Appl(\Pi, C, max)$ and from the sets of rules in $Appl(\Pi, C, smax)$, respectively, only those are taken which affect the maximal number of objects. As with affecting the maximal number of objects, such

multisets and such sets of rules are non-extendable anyway, we will also use the notations $max_{objects}$ and $smax_{objects}$.

As usual, with all these variants of derivation modes as defined above, we consider halting computations. We may generate or accept or even compute functions or relations. The inputs/outputs may be multisets or strings, defined in the well-known way.

For any derivation mode γ,

$$\gamma \in \{sequ, asyn, max, smax\} \cup \{max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\},$$

the families of number sets ($Y = N$) and Parikh sets ($Y = Ps$) $Y_{\gamma,\delta}(\Pi)$, generated ($\delta = gen$) or accepted ($\delta = acc$) by P systems with at most m membranes and rules of type X, are denoted by $Y_{\gamma,\delta}OP_m(X)$.

4 Computational Completeness Proofs also Working for Set Derivation Modes

In this section we list several variants of P systems where the computational completeness proofs also work for the set derivation modes even being taken literally from the literature.

4.1 P Systems with Cooperative Rules

We first consider *simple P systems with cooperative rules* having only one membrane (the skin membrane), which also serves as input and output membrane, and cooperative rules of the form $u \to v$. Only specifying the relevant parts, we may write $\Pi = (O, w_1, R_1)$ where

- O is the alphabet of objects,
- w_1 is the finite multiset of objects over O present in the skin membrane at the beginning of a computation,
- R_1 is a finite set of cooperative rules.

For a rule $u \to v \in R_1$, $|uv|$ is called its *size*.

Theorem 1. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a simple P system $\Pi = (O, w_1 = l_0, R_1)$ with cooperative rules of size at most 3 working in any of the derivation modes γ,*

$$\gamma \in \{max, max_{rules}, max_{objects}, smax, smax_{rules}, smax_{objects}\},$$

and simulating the computations of M such that

$$|R_1| \leq 1 \times |ADD^1(R)| + 2 \times |ADD^2(R)| + 5 \times |SUB(R)|.$$

Proof. Let $M = (m, B, l_0, l_h, R)$ be an arbitrary register machine. We now construct a simple P system with cooperative rules of size at most 3 simulating M. The number in register r is represented by the corresponding number of symbol objects o_r.

A deterministic ADD-instruction $p : (ADD(r), q)$ is simulated by the rule $p \to o_r q$.

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the two rules $p \to o_r q$ and $p \to o_r s$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated by the following rules:

1. $p \to p'p''$;
2. $p' \to \tilde{p}$, $p''o_r \to \bar{p}$
   (executed in parallel if register r is not empty);
3. $\tilde{p}p'' \to s$ (if register r has been empty),
   $\tilde{p}\bar{p} \to q$ (if register r has not been empty).

In the case of a deterministic register machine, the simulation by the P system is deterministic, too.

We observe that the construction works for every maximal derivation mode, even if only sets of rules are taken into account. □

4.2 Catalytic and Purely Catalytic P Systems

We now investigate proofs elaborated for catalytic and purely catalytic P systems working in the *max*-mode for the derivation modes *smax*, max_{rules}, $smax_{rules}$, $max_{objects}$, and $smax_{objects}$.

Based on the proof construction elaborated in [2] we state the following result:

Theorem 2. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a simple catalytic P system $\Pi = (O, C, w_1, R_1)$ working in any of the derivation modes γ,*

$$\gamma \in \{max, smax, max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\},$$

and simulating the computations of M such that

$$|R_1| \leq 1 \times |ADD^1(R)| + 2 \times |ADD^2(R)| + 5 \times |SUB(R)| + 5 \times m + 1.$$

Proof. We first check the construction for simulating a register machine $M = (d, B, l_0, l_h, R)$ by a catalytic P system Π, with $m \leq d$ being the number of decrementable registers, elaborated in [2] for the *max*-mode, and argue why it works for the derivation mode *smax* and the other maximal (set) derivation modes, too.

For all d registers, n_i copies of the symbol o_i are used to represent the value n_i in register i, $1 \leq i \leq d$. For each of the m decrementable registers, we take a catalyst c_i and two specific symbols d_i, e_i, $1 \leq i \leq m$, for simulating SUB-instructions on these registers. For every $p \in B$, we use p, and also its variants $\bar{p}, \hat{p}, \tilde{p}$ for $p \in B_{SUB}$, where B_{SUB} denotes the set of labels of SUB-instructions.

For $r < m$, $r \oplus_m 1$ simply is $r + 1$, whereas for $r = m$ we define $m \oplus_m 1 = 1$; w_0 stands for additional input present at the beginning.

Usually, every catalyst c_i, $i \in \{1, \ldots, m\}$, is kept busy with the symbol d_i using the rule $c_i d_i \to c_i$, as otherwise the symbols d_i would have to be trapped by the rule $d_i \to \#$, and the trap rule $\# \to \#$ then enforces an infinite non-halting computation. We use the following shortcuts:

$$\begin{aligned} D_m &= \prod_{i \in \{1,\ldots,m\}} d_i, \\ D_{m,r} &= \prod_{i \in \{1,\ldots,m\} \setminus \{r\}} d_i, \\ D'_{m,r} &= \prod_{i \in \{1,\ldots,m\} \setminus \{r, r \oplus_m 1\}} d_i. \end{aligned}$$

Only during the simulation of SUB-instructions on register r the corresponding catalyst c_r is left free for decrementing or for zero-checking in the second step of the simulation, and in the decrement case both c_r and its "coupled" catalyst $c_{r \oplus_m 1}$ are needed to be free for specific actions in the third step of the simulation.

$$\begin{aligned} \Pi &= (O, C, w_1 = c_1 \ldots c_m d_1 \ldots d_m p_1 w_0, R_1), \\ O &= C \cup D \cup E \cup \Sigma \cup \{\#\} \cup B \cup \{\bar{p}, \hat{p}, \tilde{p} \mid l \in B_{SUB}\}, \\ C &= \{c_i \mid 1 \le i \le m\}, \\ D &= \{d_i \mid 1 \le i \le m\}, \\ E &= \{e_i \mid 1 \le i \le m\}, \\ \Sigma &= \{o_i \mid 1 \le i \le d\}, \\ R_1 &= \{p \to o_r q D_m, p \to o_r s D_m \mid p : (ADD(r), q, s) \in R\} \\ &\quad \cup \{p \to \hat{p} e_r D_{m,r}, p \to \bar{p} D_{m,r}, \hat{p} \to \tilde{p} D'_{m,r}, \\ &\qquad \tilde{p} \to q D_m, \bar{p} \to s D_m \mid p : (SUB(r), q, s) \in R\} \\ &\quad \cup \{c_r o_r \to c_r d_r, c_r d_r \to c_r, c_{r \oplus_m 1} e_r \to c_{r \oplus_m 1} \mid 1 \le r \le m\}, \\ &\quad \cup \{d_r \to \#, c_r e_r \to c_r \# \mid 1 \le r \le m\} \\ &\quad \cup \{\# \to \#\}. \end{aligned}$$

The HALT-instruction labeled l_h is simply simulated by not introducing the corresponding state symbol l_h, i.e., replacing it by λ, in all rules defined in R_1.

Each ADD-instruction $p : (ADD(r), q, s)$, for $r \in \{1, \ldots, d\}$, can easily be simulated by the rules $p \to o_r q D_m$ and $p \to o_r s D_m$; in parallel, the rules $c_i d_i \to c_i$, $1 \le i \le m$, have to be carried out, as otherwise the symbols d_i would have to be trapped by the rules $d_i \to \#$.

Each SUB-instruction $p : (SUB(r), q, s)$, is simulated as shown in the table listed below (the rules in brackets [and] are those to be carried out in case of a wrong choice):

Simulation of the SUB-instruction $p : (SUB(r), q, s)$ if register r is not empty	register r is empty
$p \to \hat{p} e_r D_{m,r}$	$p \to \bar{p} D_{m,r}$
$c_r o_r \to c_r d_r$ $[c_r e_r \to c_r \#]$	c_r should stay idle
$\hat{p} \to \tilde{p} D'_{m,r}$	$\bar{p} \to s D_m$
$c_r d_r \to c_r$ $[d_r \to \#]$	$[d_r \to \#]$
$\tilde{p} \to q D_m$	
$c_{r \oplus_m 1} e_r \to c_{r \oplus_m 1}$	

In the first step of the simulation of each instruction (ADD-instruction or SUB-instruction) due to the introduction of D_m in the previous step (we also start with that in the initial configuration) every catalyst c_r is kept busy by the corresponding symbol d_r, $1 \leq r \leq m$.

We finally observe that the catalytic P system constructed above not only works correctly with the maximally parallel derivation mode max, but also for the derivation modes max_{rules} and $max_{objects}$ as well as for the set derivation modes $smax$, $smax_{rules}$, and $smax_{objects}$. Each maximality condition guarantees that the rules are applied in a correct way in every step.

The only difference is that in the set derivation modes smax-mode, $smax_{rules}$, and $smax_{objects}$ only one trap rule $\# \rightarrow \#$ will be carried out! □

For the purely catalytic case, one additional catalyst c_{m+1} is needed to be used with all the non-cooperative rules. Unfortunately, in this case a slightly more complicated simulation of SUB-instructions is needed, a result established in [17], where for catalytic P systems

$$|R_1| \leq 2 \times |ADD^1(R)| + 3 \times |ADD^2(R)| + 6 \times |SUB(R)| + 5 \times m + 1,$$

and for purely catalytic P systems

$$|R_1| \leq 2 \times |ADD^1(R)| + 3 \times |ADD^2(R)| + 6 \times |SUB(R)| + 6 \times m + 1$$

is shown. We observe that again the construction works for every maximal derivation mode, even if only sets of rules are taken into account.

4.3 Computational Completeness of (Purely) Catalytic P Systems with Additional Control Mechanisms

In this subsection we mention results for (purely) catalytic P systems with additional control mechanisms, in that way reaching computational completeness with only one (two) catalyst(s).

P Systems with Label Selection

For all the variants of P systems of type X, we may consider labeling all the rules in the sets $R_1, \ldots, R_m$ in a one-to-one manner by labels from a set H and taking a set W containing subsets of H. In any transition step of a *P system with label selection* Π we first select a set of labels $U \in W$ and then, in the given derivation mode, we apply a non-empty multiset R of rules such that all the labels of these rules from R are in U.

The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and

$$\gamma \in \{sequ, asyn, max, smax, max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\},$$

computed by P systems with label selection with at most m membranes and rules of type X are denoted by $Y_{\gamma,\delta}OP_m(X, ls)$.

Theorem 3. *$Y_{\gamma,\delta}OP_1\,(cat_1, ls) = Y_{\gamma,\delta}OP_1\,(pcat_2, ls) = YRE$ for any $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and any (set) derivation mode γ,*

$$\gamma \in \{max, smax, max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\}\,.$$

The proof given in [8] for the maximally parallel mode max can be taken over for the other maximal (set) derivation modes word by word; the only difference again is that in set derivation modes, in non-successful computations where more than one trap symbol # has been generated, the trap rule $\# \to \#$ is only applied once.

Controlled P Systems and Time-Varying P Systems

Another method to control the application of the labeled rules is to use control languages (see [3,11]). In a *controlled P system* Π, in addition we use a set H of labels for the rules in Π, and a string language L over 2^H (each subset of H represents an element of the alphabet for L) from a family FL. Every successful computation in Π has to follow a control word $U_1 \ldots U_n \in L$: in transition step i, only rules with labels in U_i are allowed to be applied (in the underlying derivation mode, for example, max or $smax$), and after the n-th transition, the computation halts; we may relax this end condition, i.e., we may stop after the i-th transition for any $i \leq n$, and then we speak of *weakly controlled P systems*. If $L = (U_1 \ldots U_p)^*$, Π is called a *(weakly) time-varying P system*: in the computation step $pn + i$, $n \geq 0$, rules from the set U_i have to be applied; p is called the *period*.

The family of sets $Y_{\gamma,\delta}\,(\Pi)$, $Y \in \{N, Ps\}$, computed by (weakly) controlled P systems and (weakly) time-varying P systems with period p, with at most m membranes and rules of type X as well as control languages in FL is denoted by $Y_{\gamma,\delta}OP_m\,(X, C\,(FL))$ $(Y_{\gamma,\delta}OP_m\,(X, wC\,(FL)))$ and $Y_{\gamma,\delta}OP_m\,(X, TV_p)$ $(Y_{\gamma,\delta}OP_m\,(X, wTV_p))$, respectively, for $\delta \in \{gen, acc\}$ and

$$\gamma \in \{sequ, asyn, max, smax\} \cup \{max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\}\,.$$

Theorem 4. *$Y_{\gamma,\delta}OP_1\,(cat_1, \alpha TV_6) = Y_{\gamma,\delta}OP_1\,(pcat_2, \alpha TV_6) = YRE$, for any $\alpha \in \{\lambda, w\}$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and*

$$\gamma \in \{max, smax, max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\}\,.$$

The proof given in [8] for the maximally parallel mode max again can be taken over for the other maximal (set) derivation modes word by word, e.g., see [4].

5 P Systems with Toxic Objects

In many variants of (catalytic) P systems, for proving computational completeness it is common to introduce a trap symbol # for the case that the derivation goes the wrong way as well as the rule $\# \to \#$ (or $c\# \to c\#$ with a catalyst c)

guaranteeing that the derivation will never halt. Yet most of these rules can be avoided if we specify a specific subset of *toxic* objects O_{tox}.

The P system with toxic objects is only allowed to continue a computation from a configuration C by using an applicable multiset of rules covering all copies of objects from O_{tox} occurring in C; moreover, if there exists no multiset of applicable rules covering all toxic objects, the whole computation having yielded the configuration C is abandoned, i.e., no results can be obtained from this computation.

For any variant of P systems, we add the set of *toxic* objects O_{tox} and in the specification of the families of sets of (vectors of) numbers generated by P systems with toxic objects using rules of type X we add the subscript tox to O, thus obtaining the families $Y_{\gamma,\delta}O_{tox}P_m(X)$, for any $m \geq 1$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and

$$\gamma \in \{max, smax, max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\}.$$

The following theorem stated in [1] only for the max-mode obviously holds for the other maximal (set) derivation modes, too.

Theorem 5. *$PsRE = Ps_{\gamma,gen}O_{tox}P_1([p]cat_2)$ for every*

$$\gamma \in \{max, smax, max_{rules}, smax_{rules}, max_{objects}, smax_{objects}\}.$$

In general, we can formulate the following "metatheorem":

Metatheorem: *Whenever a proof has been established for the derivation mode max and literally also holds true for the derivation mode smax, then the omitting of trap rules by using the concept of toxic objects works for both derivation modes in the same way.*

In the following sections, we now turn our attention to models of P systems where the derivation mode *smax* yields different, in fact, stronger results than the derivation mode *max*.

6 Atomic Promoters and Inhibitors

As shown in [16], P systems with non-cooperative rules and atomic inhibitors are not computationally complete when the maximally parallel derivation mode is used. P systems with non-cooperative rules and atomic promoters can at least generate $PsET0L$. On the other hand, already in [13], the computational completeness of P systems with applying maximal sets of non-cooperative rules and atomic promoters has been shown. In the following we recall our new proof from [4] for the simulation of a register machine where the overall number of promoters only depends on the number of decrementable registers of the register machine. Moreover, we also recall the proof of a new rather surprising result, establishing computational completeness of P systems with applying maximal sets of non-cooperative rules and atomic inhibitors, where the number of inhibitors again only depends on the number of decrementable registers of the simulated register machine. Finally, in both cases, if the register machine is deterministic, then the P system is deterministic, too.

6.1 Atomic Promoters

We now recall our new proof from [4] for the computational completeness of P systems with non-cooperative rules and atomic promoters when using any of the set derivation modes $smax, smax_{rules}, smax_{objects}$. The overall number of promoters only is $5m$ where m is the number of decrementable registers of the simulated register machine.

Theorem 6. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a simple P system with atomic inhibitors $\Pi = (O, w_1 = l_0, R_1)$ working in any of the set derivation modes $smax, smax_{rules}, smax_{objects}$ and simulating the computations of M such that*

$$|R_1| \leq 1 \times |ADD^1(R)| + 2 \times |ADD^2(R)| + 5 \times |SUB(R)| + 7 \times m.$$

The number of atomic inhibitors is $5m$. Finally, if the register machine is deterministic, then the P system is deterministic, too.

Proof. The numbers of objects o_r represent the contents of the registers r, $1 \leq r \leq d$; moreover, we denote $B_{SUB} = \{p \mid p : (SUB(r), q, s) \in R\}$.

$$\begin{aligned} O = &\{o_r \mid 1 \leq r \leq d\} \cup \{o'_r, c_r, c'_r, c''_r, c'''_r \mid 1 \leq r \leq m\} \\ &\cup (B \setminus \{l_h\}) \cup \{p', p'', p''' \mid p \in B_{SUB}\} \end{aligned}$$

The symbols from $\{o'_r, c_r, c'_r, c''_r, c'''_r \mid 1 \leq r \leq m\}$ are used as promoters.

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the two rules $p \to qo_r$ and $p \to so_r$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated in four steps as follows:

1. $p \to p'c_r$;
2. $p' \to p''c'_r$, $o_r \to o'_r \mid_{c_r}$, $c_r \to \lambda$;
3. $p'' \to p'''c'''_r$, $c'_r \to c''_r \mid_{o'_r}$, $o'_r \to \lambda$;
4. $p''' \to q \mid_{c''_r}$, $p''' \to s \mid_{c'_r}$, $c'_r \to \lambda \mid_{c'''_r}$, $c''_r \to \lambda$, $c'''_r \to \lambda$.

As final rule we could use $l_h \to \lambda$, yet we can omit this rule and replace every appearance of l_h in all rules as described above by λ. □

6.2 Atomic Inhibitors

We now show that even P systems with non-cooperative rules and atomic promoters using one of the derivation modes $smax, smax_{rules}, smax_{objects}$ can simulate any register machine needing only $2m+1$ inhibitors where m is the number of decrementable registers of the simulated register machine.

Theorem 7. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a P system with atomic*

inhibitors $\Pi = (O, w_1 = l_0, R_1)$ *working in any of the set derivation modes* $smax, smax_{rules}, smax_{objects}$ *and simulating the computations of* M *such that*

$$|R_1| \leq 1 \times |ADD^1(R)| + 2 \times |ADD^2(R)| + 5 \times |SUB(R)| + 3 \times m + 1.$$

The number of atomic inhibitors is $2m + 1$. *Finally, if the register machine is deterministic, then the P system is deterministic, too.*

Proof. The numbers of objects o_r represent the contents of the registers r, $1 \leq r \leq d$. The symbols d_r prevent the register symbols o_r, $1 \leq r \leq m$, from evolving.

$$O = \{o_r \mid 1 \leq r \leq d\} \cup \{o'_r \mid 1 \leq r \leq m\} \cup \{d_r \mid 0 \leq r \leq m\}$$
$$\cup (B \setminus \{l_h\}) \cup \{p', p'', \tilde{p} \mid p \in B_{SUB}\}$$

We denote $D = \prod_{i=1}^{m} d_i$ and $D_r = \prod_{i=1, i \neq r}^{m} d_i$.

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the two rules $p \to qo_rD$ and $p \to so_rD$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated in four steps as follows:

1. $p \to p'D_r$;
2. $p' \to p''Dd_0$; in parallel, the following rules are used:
   $o_r \to o'_r \,|_{\neg d_r}$; $d_k \to \lambda$, $1 \leq k \leq m$;
3. $p'' \to \tilde{p}D \,|_{\neg o'_r}$; $o'_r \to \lambda$, $d_0 \to \lambda$;
   again, in parallel the rules $d_k \to \lambda$, $1 \leq k \leq m$, are used;
4. $p'' \to qD \,|_{\neg d_0}$, $\tilde{p} \to sD$.

As final rule we could use $l_h \to \lambda$, yet we can omit this rule and replace every appearance of l_h in all rules as described above by λ. □

7 P Systems with Target Selection

In P systems with target selection, all objects on the right-hand side of a rule must have the same target, and in each derivation step, for each region a (multi)set of rules – non-empty if possible – having the same target is chosen. In [4] it was shown that for P systems with target selection in the derivation mode *smax* ***no*** catalyst is needed any more, and with $smax_{rules}$, we even obtain a deterministic simulation of deterministic register machines.

Theorem 8. *For any register machine* $M = (d, B, l_0, l_h, R)$, *with* $m \leq d$ *being the number of decrementable registers, we can construct a P system with non-cooperative rules working in the set derivation mode smax and simulating the computations of* M.

Proof. As usual, we take an arbitrary register machine M with d registers satisfying the following conditions: the output registers are $m+1, \cdots, d$, and they are never decremented; moreover, registers $1, \cdots, m$ are empty in any reachable halting configuration.

We now construct the following P system Π simulating M. The value of each register r is represented by the multiplicity of objects o_r in the skin.

The correct behavior of the objects associated with the simulated instruction of M is the following:

In the decrement case, we have in_{r+2}, *out*, in_2, idle, *out*, in_2, *here*, *out*, *here* (9 steps in total), whereas in the zero-test case, we have the same as before, except that the fourth and the fifth steps are *out* and *here* instead of idle and *out*, respectively.

In case of an increment instruction, we get *here*, *here*, *here*, *here*, in_2, *here*, *out*, *here* (8 steps in total). We remark that the first four steps are carried out in the skin, while the last four steps repeat the cases of zero-test and decrement.

For every decrementable register r, there is a rule sending o_r into region $r+2$. However, this rule may only be applied safely in the first step of the simulation of the SUB-instruction, as otherwise some other object will also enter the same region as # (either one of e, e', e'', $\hat{e}$, $\hat{e}'$, which in the following we will refer to as the *guards*, or an object associated to the label of the simulated instruction, which in the following we will call a *program symbol*) forcing an unproductive computation, see the rules in brackets in the tables below.

The "correct" target selection for the inner regions normally coincides with that of the program symbol (described above) and no rule is applied there if the program symbol is not there, with the following exceptions. In the first step of simulating an instruction, object e exits membrane 2, as it is the only rule applicable there in this step. In the last step of simulating an instruction, object $\bar{e}$ is rewritten into e in membrane 2, as it is the only rule applicable there in this step. In the fourth step of the decrement case, the program symbol is idle while object d is erased. The "correct" target selection for the skin coincides with that of the program symbol, and is *here* if the program symbol is missing in the skin.

Most trapping rules, given in brackets in the tables and listed in rule groups $R_{i,j,\#}$ below, are only needed to force the "correct" target selection. The exception are some rules in steps 4 and 5 of the simulation of SUB-instructions, needed for verifying that the decrement and the zero-test have been performed correctly (the guess is made at step 3 by the program symbol, and is reflected in its subscript). Indeed, the zero-test being chosen while d is present (signifying that the register has been decremented), causes a target conflict, and either p_0 or d in any case will be rewritten into #. On the other hand, if the decrement is chosen while d is absent (signifying that the register has been zero), then p_- will appear in the skin in step 4 instead of step 5, causing a target conflict, and either p'_- or e'' in any case will be rewritten into #.

$$
\begin{aligned}
\Pi &= (O, \mu, w_1, \cdots, w_{m+2}, R_1, \cdots, R_{m+2}) \text{ where}\\
O &= \{o_r \mid 1 \le r \le d\} \cup \{\bar{p}, p, p' \mid p \in B\} \cup \{p'', \hat{p}, \tilde{p} \mid p \in B_{ADD}\}\\
&\quad \cup \{p', p_-, p'_-, p_0, p'_0, p''_0 \mid p \in B_{SUB}\} \cup \{\bar{e}, e, e', e'', \hat{e}, \hat{e}', d, \#\},\\
\mu &= [\,[\,]_2 \cdots [\,]_{m+2}]_1,\\
w_1 &= l_0,\ w_2 = e,\ w_{r+2} = \lambda,\ 1 \le r \le m,\\
R_1 &= \bigcup_{i=1}^{m+2} (R_{1,i,s} \cup R_{1,i,\#}),\\
R_i &= R_{i,1,s} \cup R_{i,1,\#} \cup R_{i,i,s} \cup R_{i,i,\#},\ 2 \le j \le m+2,\\
R_{1,1,s} &= \{e \to e', e' \to e'', e'' \to \hat{e}, \hat{e} \to \hat{e}', \hat{e}' \to \lambda\}\\
&\quad \cup \{p'_0 \to p''_0 \mid p \in B_{SUB}\} \cup \{\bar{p} \to p \mid p \in B\}\\
&\quad \cup \{p \to \tilde{p} o_r \mid p : (ADD(r), q, s) \in P\}\\
&\quad \cup \{\tilde{p} \to p', p' \to p'', p'' \to \hat{p} \mid p \in B_{ADD}\},\\
R_{1,2,s} &= \{p' \to (p_-, in_2), p' \to (p_0, in_2), p'_- \to (p'_-, in_2), p''_0 \to (p''_0, in_2)\\
&\quad \mid p \in B_{SUB}\} \cup \{\hat{p} \to (\hat{p}, in_2) \mid p \in B_{ADD}\} \cup \{d \to (d, in_2)\}\\
R_{1,r+2,s} &= \{o_r \to (o_r, in_{r+2})\} \cup \{p \to (p, in_{r+2})\\
&\quad \mid p : (SUB(r), q, s) \in P\},\ 1 \le r \le m,\\
R_{1,1,\#} &= \{p' \to \#, p''_0 \to \#, p'_- \to \# \mid p \in B_{SUB}\}\\
&\quad \cup \{\hat{p} \to \# \mid p \in B_{ADD}\} \cup \{\# \to \#\},\\
R_{1,2,\#} &= \{p'_0 \to (\#, in_2), e'' \to (\#, in_2) \mid p \in B_{SUB}\}\\
&\quad \cup \{\bar{p} \to (\#, in_2) \mid p \in B\},\\
R_{1,r+2,\#} &= \{x \to (\#, in_{r+2}\} \mid x \in \{e, e', e'', \hat{e}, \hat{e}'\}\\
&\qquad \cup \{p_0, p''_0, p'_- \mid p \in B_{SUB}\} \cup \{\bar{p} \mid p \in B\}\}\\
&\quad \cup \{p \to (\#, in_{r+2}) \mid p : (SUB(i), q, s) \in P,\ i \ne r\}\\
&\quad \cup \{p' \to (\#, in_{r+2}) \mid p \in B_{SUB}\},\ 1 \le r \le m,\\
R_{2,1,s} &= \{e \to (e, out)\} \cup \{\bar{p} \to (\bar{p}, out) \mid p \in B\}\\
&\quad \cup \{p_0 \to (p'_0, out), p_- \to (p'_-, out) \mid p \in B_{SUB}\},\\
R_{2,2,s} &= \{d \to \lambda, \bar{e} \to e\}\\
&\quad \cup \{p''_0 \to \bar{s}\bar{e}, p'_- \to \bar{q}\bar{e} \mid p : (SUB(r), q, s) \in P\}\\
&\quad \cup \{\hat{p} \to \bar{q}\bar{e}, \hat{p} \to \bar{s}\bar{e} \mid p : (ADD(r), q, s) \in P\},\\
R_{2,1,\#} &= \{d \to (\#, out), \# \to (\#, out)\},\\
R_{2,2,\#} &= \{p_0 \to \# \mid p \in B_{SUB}\} \cup \{\bar{p} \to \# \mid p \in B\},\\
R_{r+2,1,s} &= \{p \to (p', out) \mid p \in B_{SUB}\} \cup \{o_r \to (d, out)\},\ 1 \le r \le m,\\
R_{r+2,1,\#} &= \{\# \to (\#, out)\},\ R_{r+2,r+2,s} = R_{r+2,r+2,\#} = \emptyset,\ 1 \le r \le m.
\end{aligned}
$$

Simulation of a SUB-instruction (p:(SUB(r),q,s))

	$r+2$	1	2
1	- -	$o_r \to (o_r, in_{r+2})$ $p \to (p, in_{r+2})$ $[p \to (\#, in_{i+2}),\ i \neq r]$	$e \to (e, out)$
2	$p \to (p', out)$ $o_r \to (d, out)$	$e \to e'$ $[e \to (\#, in_{i+2})]$	-
3	-	$p' \to (p_-, in_2)$ $p' \to (p_0, in_2)$ $d \to (d, in_2)$ $[p' \to \#]$ $[p' \to (\#, in_{i+2})]$ $[e' \to (\#, in_{i+2})]$	-

	1,-	1,0	2,-	2,0
4	$e' \to e''$		$d \to \lambda$ $[p_- \to (p'_-, out)]$	$p_0 \to (p'_0, out)$ $[d \to (\#, out)]$ $[p_0 \to \#]$
5	$e'' \to \hat{e}$ $[p'_- \to (p'_-, in_2)]$ $[p'_- \to \#]$ $[e'' \to (\#, in_t)]$ [for $t \geq 2$]	$p'_0 \to p''_0$ $e'' \to \hat{e}$ $[p'_0 \to (\#, in_t)]$ $[e'' \to (\#, in_t)]$ [for $t \geq 2$]	$p_- \to (p'_-, out)$	-
6	$p'_- \to (p'_-, in_2)$ $[p'_- \to \#]$ $[p'_- \to (\#, in_{i+2})]$	$p''_0 \to (p''_0, in_2)$ $[p''_0 \to \#]$ $[p''_0 \to (\#, in_{i+2})]$	-	
7	$\hat{e} \to \hat{e}'$ $[\hat{e} \to (\#, in_{i+2})]$		$p'_- \to \bar{q}\bar{e}$	$p''_0 \to \bar{s}\bar{e}$
8	$\hat{e}' \to \lambda$ $[\hat{e}' \to (\#, in_{i+2})]$		$\bar{q} \to (\bar{q}, out)$ $[\bar{q} \to \#]$	$\bar{s} \to (\bar{s}, out)$ $[\bar{s} \to \#]$
9	$\bar{q} \to q$ $[\bar{q} \to (\#, in_t)]$	$\bar{s} \to s$ $[\bar{s} \to (\#, in_t)]$	$\bar{e} \to e$	

Nearly half of the steps in the preceding constructions is needed for releasing the auxiliary symbol e in the first step of a simulation from membrane 2, yet in our construction, e and its derivatives are needed to control the correct target selection in the skin membrane, and especially to keep the register objects o_r from moving into membrane $r+2$.

Auxiliary trap rules

$r+2$	1	2
$[\# \to (\#, out)]$	$[\# \to \#]$	$[\# \to (\#, out)]$

Simulation of an ADD-instruction (p:(ADD(r),q,s))

	1	2
1	$p \to \tilde{p}o_r$	$e \to (e, out)$
2	$\tilde{p} \to p'$	-
3	$e \to e'$ $p' \to p''$ $e' \to e''$	-
4	$p'' \to \hat{p}$ $e'' \to \hat{e}$	-
5	$\hat{p} \to (\hat{p}, in_2)$ $[\hat{p} \to \#]$	-
6	$\hat{e} \to \hat{e}'$	$\hat{p} \to \bar{x}\bar{e},\ x \in \{q, s\}$
7	$\hat{e}' \to \lambda$	$\bar{x} \to (\bar{x}, out)$ $[\bar{x} \to \#]$
8	$\bar{x} \to x$	$\bar{e} \to e$

Again we mention that any application of one of the rules given in brackets in the tables above leads to non-halting computations, not contributing to the result. □

We now show that taking the maximal sets of rules which are applicable, the simulation of SUB-instructions can even be carried out in a deterministic way.

Theorem 9. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a P system with non-cooperative rules*

$$\Pi = (O, \mu = [\ [\ ]_2 \dots [\ ]_{2m+1}\ ]_1, w_1, \lambda, \dots, \lambda, R_1 \dots R_{2m+1}, f = 1)$$

working in the derivation mode $smax_{rules}$ and simulating the computations of M such that

$$|R_1| \leq 1 \times |ADD^1(R)| + 2 \times |ADD^2(R)| + 4 \times |SUB(R)| + 10 \times m + 3.$$

Proof. The contents of the registers r, $1 \leq r \leq d$, is represented by the numbers of objects o_r, and for the decrementable registers we also use a copy of the symbol o'_r for each copy of the object o_r. This second copy o'_r is needed during the simulation of SUB-instructions to allow for distinguishing between the decrement and the zero-test case. For each r, $1 \leq r \leq m$, the two objects o_r and o'_r can only be affected by the rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$ sending them into the membrane $r + 1$ corresponding to register r (and at the same time erasing them; in fact, we could also leave them in the membrane unaffected forever as a garbage). These are already two rules, so any other combination of rules with different targets has to contain at least three rules.

One of the main ideas of the proof construction is that in the skin membrane the label p of an ADD-instruction is represented by the three objects p and e, e', and the label p of any SUB-instruction is represented by the eight objects $p, e, e', e'', d_r, d'_r, \tilde{d}_r, \tilde{d}_r{}'$. Hence, for each $p \in (B \setminus \{l_h\})$ we define $R(p) = pee'$ for $p \in B_{ADD}$ and $R(p) = pee'e''d_r d'_r \tilde{d}_r \tilde{d}_r{}'$ for $p \in B_{SUB}$ as well as $R(l_h) = \lambda$; as initial multiset w_1 in the skin membrane, we take $R(l_0)$.

$$O = \{o_r \mid 1 \le r \le d\} \cup \{o'_r \mid 1 \le r \le m\} \cup (B \setminus \{l_h\}) \\ \cup \left\{d_r, d'_r, \tilde{d}_r, \tilde{d}_r{}' \mid 1 \le r \le m\right\} \cup \{e, e', e''\}$$

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the rules $e \to \lambda$ and $e' \to \lambda$ as well as by the rules $p \to R(q)o_r$ and $p \to R(s)o_r$ for $m < r \le d$, and, in the case of the decrementable registers, for $1 \le r \le m$, by the rules $p \to R(q)o_r o'_r$ and $p \to R(s)o_r o'_r$. Any possible maximal combination of these rules yields a (multi)set of three rules and thus supersedes any combination of rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$, for some $1 \le r \le m$.

A SUB-instruction $p : (SUB(r), q, s)$ is simulated in two steps as follows:

1. In R_1, for the first step we take one of the following tuples of rules:
   $p \to (p, in_{r+1})$, $d_r \to (\lambda, in_{r+1})$, $d'_r \to (\lambda, in_{r+1})$, $\tilde{d}_r \to (\lambda, in_{r+1})$,
   $o_r \to (\lambda, in_{r+1})$, $o'_r \to (\lambda, in_{r+1})$;
   $p \to (p, in_{m+r+1})$, $d_r \to (\lambda, in_{m+r+1})$, $d'_r \to (\lambda, in_{m+r+1})$,
   $\tilde{d}_r \to (\lambda, in_{m+r+1})$, $\tilde{d}_r{}' \to (\lambda, in_{m+r+1})$;
   the application of the rules $o_r \to (\lambda, in_{r+1})$, $o'_r \to (\lambda, in_{r+1})$ in contrast to the application of the rule $\tilde{d}_r{}' \to (\lambda, in_{m+r+1})$ determines whether the first or the second tuple of rules has to be chosen. Here it becomes clear why we have to use the two register symbols o_r and o'_r, as we have to guarantee that the target $r + 1$ cannot be chosen if none of these symbols is present, as in this case then only four rules could be chosen in contrast to the five rules for the zero-test case. On the other hand, if some of these symbols o_r and o'_r are present, then six rules are applicable superseding the five rules which could be used for the zero-test case.
2. In the second step, the following three or four rules, again superseding any combination of rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$ for some $1 \le r \le m$, are used in the skin membrane:
   $e \to \lambda$, $e' \to \lambda$, $e'' \to \lambda$, and in the decrement case also the rule $\tilde{d}_r{}' \to \lambda$.
   In the second step, we either find the symbol p in membrane $r + 1$, if a symbol o_r together with its copy o'_r has been present for decrementing or in membrane $m + r + 1$, if no symbol o_r has been present (zero-test case).
   In the decrement case, the following rule is used in R_{r+1}: $p \to (R(q), out)$.
   In the zero-test case, the following rule is used in R_{m+r+1}: $p \to (R(s), out)$.

The simulation of the SUB-instructions works deterministically, hence, although the P system itself is not deterministic syntactically, it works in a deterministic way if the underlying register machine is deterministic. □

In contrast to the derivation mode $smax_{rules}$ where we take the maximal sets of rules which are applicable, in the *smax*-derivation mode we may have several non-extendable sets of rules which are applicable to the current configuration although being of different sizes, which has made that proof much more difficult than in the case of $smax_{rules}$.

8 Conclusion and Future Work

It is not very surprising that many of the computational completeness proofs elaborated in the literature for the derivation mode *max* also work for the set derivation mode *smax* and usually even for the other (set) derivation modes max_{rules} and $smax_{rules}$ as well as for $max_{objects}$ and $smax_{objects}$, because many constructions just "break down" maximal parallelism to near sequentiality in order to work for the simulation of register machines. On the other hand, we also have shown that due to this fact some variants of P systems become even stronger with the modes *smax* and $smax_{rules}$. A comprehensive overview of variants of P systems we have already investigated can be found in [4], many more variants wait for future research.

References

1. Alhazov, A., Freund, R.: P systems with toxic objects. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) CMC 2014. LNCS, vol. 8961, pp. 99–125. Springer, Cham (2014). doi:10.1007/978-3-319-14370-5_7
2. Alhazov, A., Freund, R.: Small catalytic P systems. In: Dinneen, M.J. (ed.) Proceedings of the Workshop on Membrane Computing 2015 (WMC2015), (Satellite workshop of UCNC2015), CDMTCS Research Report Series, vol. CDMTCS-487, pp. 1–16. Centre for Discrete Mathematics and Theoretical Computer, Science Department of Computer Science, University of Auckland, Auckland, New Zealand (2015), August 2015
3. Alhazov, A., Freund, R., Heikenwälder, H., Oswald, M., Rogozhin, Y., Verlan, S.: Sequential P systems with regular control. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) CMC 2012. LNCS, vol. 7762, pp. 112–127. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36751-9_9
4. Alhazov, A., Freund, R., Verlan, S.: Computational completeness of P systems using maximal variants of the set derivation mode. In: Proceedings 14th Brainstorming Week on Membrane Computing, Sevilla, February 1–5, 2016 (2016)
5. Burkhard, H.: Ordered firing in Petri nets. Elektronische Informationsverarbeitung und Kybernetik **17**(2/3), 71–86 (1981)
6. Ciobanu, G., Marcus, S., Păun, G.: New strategies of using the rules of a P system in a maximal way. Power and Complexity. Rom. J. Inf. Sci. Technol. **12**(2), 21–37 (2009)
7. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. EATCS Monographs in Theoretical Computer Science, vol. 18. Springer, Heidelberg (1989)
8. Freund, R., Păun, G.: How to obtain computational completeness in P systems with one catalyst. In: Neary, T., Cook, M. (eds.) Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9–11, 2013. EPTCS, vol. 128, pp. 47–61 (2013)

9. Freund, R., Verlan, S.: A formal framework for static (Tissue) P systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 271–284. Springer, Heidelberg (2007). doi:10.1007/978-3-540-77312-2_17
10. Frisco, P., Govan, G.: P systems with active membranes operating under minimal parallelism. In: Gheorghe, M., Păun, G., Rozenberg, G., Salomaa, A., Verlan, S. (eds.) CMC 2011. LNCS, vol. 7184, pp. 165–181. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28024-5_12
11. Krithivasan, K., Păun, Gh., Ramanujan, A.: On controlled P systems. In: Valencia-Cabrera, L., García-Quismondo, M., Macías-Ramos, L., Martínez-del-Amor, M., Păun, Gh., Riscos-Núñez, A. (eds.) Proceedings 11th Brainstorming Week on Membrane Computing, Sevilla, February 4–8, 2013, pp. 137–151. Fénix Editora, Sevilla (2013)
12. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice Hall, Englewood Cliffs (1967)
13. Pan, L., Păun, G., Song, B.: Flat maximal parallelism in P systems with promoters. Theoret. Comput. Sci. **623**, 83–91 (2016)
14. Păun, G., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press, New York (2010)
15. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1–3. Springer, Heidelberg (1997)
16. Sburlan, D.: Further results on P systems with promoters/inhibitors. Int. J. Found. Comput. Sci. **17**(1), 205–221 (2006)
17. Sosík, P., Langer, M.: Small (purely) catalytic P systems simulating register machines. Theoret. Comput. Sci. **623**, 65–74 (2015)
18. The P Systems Website: http://ppage.psystems.eu, http://ppage.psystems.eu
19. Verlan, S., Quiros, J.: Fast hardware implementations of P systems. In: Csuhaj-Varjú, E., Gheorghe, M., Rozenberg, G., Salomaa, A., Vaszil, G. (eds.) CMC 2012. LNCS, vol. 7762, pp. 404–423. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36751-9_27