Computational Completeness of P Systems Using 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, Chişinău, MD-2028, Moldova
E-mail: `artiom@math.md`

[2] Faculty of Informatics, TU Wien
Favoritenstraße 9-11, 1040 Wien, Austria
E-mail: `rudi@emcc.at`

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

Summary. We consider P systems only allowing rules to be used in at most one copy in each derivation step, especially the variant of the maximally parallel derivation mode where each rule may only be used at most once. Moreover, we also consider the derivation mode where from those sets of rules only those are taken which have the maximal number of rules. We check the computational completeness proofs of several variants of P systems and show that some of them even literally still hold true for the for these two new set derivation modes. Moreover, we establish two new results for P systems using target selection for the rules to be chosen together with these two new 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 another derivation mode we apply sets of rules which contain a maximal number of applicable rules (abbreviated max_{rule}).

Taking sets of rules instead of multisets is a quite natural restriction and it arises from different motivations, e.g., firing a maximal set of transitions in Petri Nets [5, 8] or optimizing an implementation of FPGA simulators [13]. A natural question arises concerning the power of set-based modes in contrast to multiset-based ones. The first attempt to go into this direction was done in [10] where it was shown that in some cases the computational completeness results established for the *max*-mode also hold for the *smax*-mode.

In this paper we continue this line of research and we show that for several variants of P systems the proofs for computational completeness for *max* can be taken over even literally for *smax* and eventually even for max_{rule}, but on the other hand there are also variants of P systems where the derivation modes *smax* and max_{rule} yield even stronger results than the *max*-mode.

2 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.

A (cell-like) P system is a construct

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

- O is the alphabet of objects,
- $C \subset O$ is the set of catalysts,
- μ is the membrane structure (with m membranes),
- $w_1, \dots, w_m$ are multisets of objects present in the m regions of μ at the beginning of a computation,
- $R_1, \dots, R_m$ 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 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 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.

In the *maximally parallel derivation mode* (abbreviated by *max*), in any computation step of Π we choose a multiset of rules from $\mathcal{R}$, defined as the union of the sets $R_1, \dots, R_m$, 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 $1, \dots, m$.

2.1 Set Derivation Modes

The basic set derivation mode is defined as the derivation mode where in each derivation step at must 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 $Appl(\Pi, C, set)$; following the notations elaborated in [7], we define the mode *smax* as follows:

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

The *smax*-derivation mode corresponds to the min_1-mode with the discrete partitioning of rules (each rule forms its own partition), see [7].

The derivation mode $max_{rule}smax$ 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 in it, hence, we will also call it the max_{rule} derivation mode. Formally we have:

$$\begin{aligned} Appl(\Pi, C, max_{rule}) = \{R \in Appl(\Pi, C, set) \mid & \text{ there is no } R' \in Appl(\Pi, C, set) \\ & \text{ such that } |R'| > |R|\} \end{aligned}$$

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

2.2 The History of the *smax*-Derivation Mode

In [13], a paper on fast P systems simulators using FPGA, the problem of the unbounded *max*-mode was considered as too difficult to be parallelized on this hardware. In the quest for an efficient solution, the authors proposed to restrict to the case of the maximal parallelism where each rule can be applied at most once. The most important advantage of this variant was that the multiset of applicable rules could be represented as a binary string, i.e., an encoding as a number. Moreover, the paper showed that in many interesting cases it is possible to represent the language of corresponding binary strings at each step by an automaton. Then the problem of the simulation of a P system could be solved as follows:

- Find the size S of the set of multisets of applicable rules (the size of the language of binary strings).
- Take a random number $k \in \{1..S\}$ and chose the string representing k.

  This algorithm allowed for obtain a speed-up of magnitude 10^5.

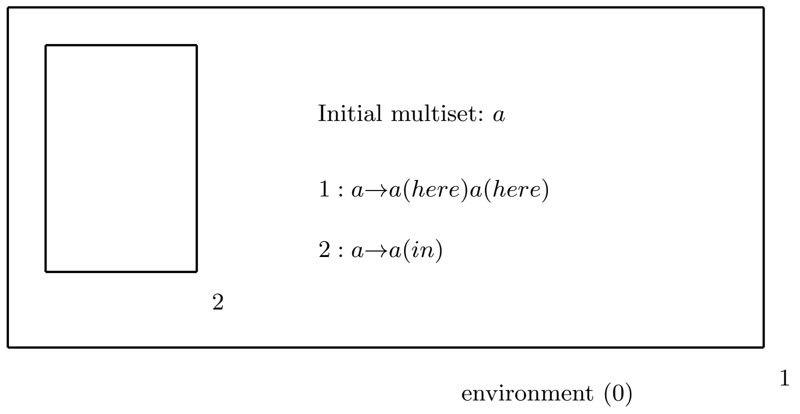
The advantages of the set-maximally parallel derivation mode over the unbounded maximally parallel derivation mode are:

- A compact representation of the applicable multisets of rules as binary strings/numbers is obtained.
- Most of the computational completeness results still hold.
- Simpler analysis of the behavior is possible.
- Only a bounded number of (multi)sets of rules has to be computed for each derivation step.

In [10], the *set-maximally parallel derivation mode* was called *flat maximal parallel derivation mode*, and, for example, P systems with promoters are shown to be computationally complete using this flat maximal parallel derivation mode with non-cooperative rules.

2.3 Examples

In the maximally parallel mode, we in addition need target or rule or label agreement to obtain $\left\{a^{2^n} \mid n \geq 0\right\}$, otherwise only $\{a^n \mid n \geq 1\}$ can be obtained.



target/ rule/ label agreement:
the same rule is used for all symbols a

Fig. 1. Example of a P system.

In the set-maximally parallel mode $smax$, we in addition need target or rule or label agreement to obtain $\{a^n \mid n \geq 1\}$, otherwise only $\{a\}$ can be obtained, because:

- If $2 : a \rightarrow a(in)$ is used in the first step, then a is obtained.
- If $1 : a \rightarrow a(here)a(here)$ is applied at least once, then from the second step on it has to be applied infinitely often, as only one copy of a can be sent into membrane 2 by the second rule $2 : a \rightarrow a(in)$.

The same arguments hold for the derivation mode max_{rule}.

3 Symport/Antiport P Systems

A *symport/antiport P system* is a construct

$$\Pi = (O, E, \mu, w_0, w_1, \dots, w_m, R_1, \dots, R_m, f_O, f_I) \text{ where}$$

- O is the alphabet of objects,
- $E \subseteq O$ is the set of objects being available in the environment in an unbounded number,
- μ is the membrane structure (with m membranes),
- w_0 is the finite multiset of objects over $O \setminus E$ present in the environment at the beginning of a computation,
- $w_1, \dots, w_m$ are the multisets of objects present in the m regions of μ at the beginning of a computation,
- $R_1, \dots, R_m$ are finite sets of symport and/or antiport rules, associated with the membranes of μ,
- f_O, f_I is the label of the membrane region from which the outputs are taken/the inputs are put in.

Every rule is of the form $(u, out; v, in)$ with $u, v \in O^*$ and $uv \neq \lambda$; if $u = \lambda$ or $v = \lambda$ then this rule is called a *symport rule*, otherwise it is called an *antiport rule*. The application of a rule $(u, out; v, in) \in R_i$ means sending out u from region i and taking v into it from the surrounding region.

For $(u, out; v, in)$, $\max\{|u|, |v|\}$ is called its *weight* and $|uv|$ is called its *size*; obviously, for symport rules weight and size are the same.

The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{sequ, asyn, max, smax, max_{rule}, \dots\}$, computed by symport/antiport P systems with at most m membranes, symport rules with maximal weight r as well as antiport rules with maximal weight w and maximal size s are denoted by $Y_{\gamma,\delta}OP_m(sym_r, anti_{w,s})$.

3.1 Accepting Antiport P Systems

Theorem 1. *For $Y \in \{N, Ps\}$, $\beta \in \{max, smax, max_{rule}\}$,*

$$Y_{\beta,acc}DOP_m(anti_{2,3}) = YRE.$$

Proof. Let $M = (m, B, l_0, l_h, P)$ be an arbitrary deterministic register machine. We now construct an antiport P system simulating M. The number in register r is represented by the corresponding number of symbol objects o_r.

- An ADD-instruction $p : (ADD(r), q)$ is simulated by the rule $(p, out; o_r q, in)$.
- A SUB-instruction $p : (SUB(r), q, s)$ is simulated by the following rules
  1. $(p, out; p'p'', in)$;
  2. $(p', out; \tilde{p}, in)$ as well as $(p''o_r, out; \bar{p}, in)$ which is executed in parallel if and only if the register is not empty;
  3. $(\tilde{p}p'', out; s, in)$ (if register was empty),
     $(\tilde{p}\bar{p}, out; q, in)$ (if register was not empty).

As can be seen immediately, in each step only different rules can be applied, each of them only once. Hence, the proof elaborated for the max-mode literally also works for the derivation modes *smax* and max_{rule} without any restrictions as well. □

4 P Systems with Anti-Matter

For any object *a (matter)*, we consider its anti-object *(anti-matter)* a^- and the corresponding (cooperative) *annihilation rule* $aa^- \to \lambda$. This rule is assumed to exist in all membranes.

In the following, we assume these annihilation rules to have (weak) priority over all other rules, i.e., other rules may only be applied if objects cannot be bound by an annihilation rule any more.

This type of rules is abbreviated by *antim/pri*, indicating matter/anti-matter annihilation rules having weak priority. For further results we refer to [1].

4.1 Matter/Anti-Matter Annihilation Rules Having Priority

The matter/anti-matter annihilation rules are so powerful that we only need the minimum number of catalysts, i.e., zero ($cat(0) = ncoo$).

Theorem 2. *[1] For any* $n \geq 1$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc, aut\}$, $\alpha \in \{acc, aut\}$, $Z \in \{Fun, Rel\}$, *and* $\beta \in \{max, smax, max_{rule}\}$,

$$Y_{\beta,\delta}OP_n\,(ncoo, antim/pri) = YRE \text{ and}$$
$$ZY_{\beta,\alpha}OP_n\,(ncoo, antim/pri) = ZYRE.$$

4.2 Deterministic Matter/Anti-Matter Accepting P Systems

In the accepting case, we can even simulate the actions of a deterministic register machine in a deterministic way, i.e., for each configuration of the system, there can be at most one multiset of rules applicable to it. Yet the proof exhibited in [1], even fulfills the condition that every rule is only applied at most once.

Theorem 3. *For any $n \geq 1$,$Y \in \{N, Ps\}$, and $\beta \in \{max, smax, max_{rule}\}$,*

$$\begin{array}{l} Y_{\beta,detacc}OP_n\,(ncoo, antim/pri) = YRE \text{ and} \\ FunY_{\beta,detacc}OP_n\,(ncoo, antim/pri) = FunYRE. \end{array}$$

Proof. We only show how the SUB-instructions of a register machine $M = (m, B', l_0, l_h, P)$ can be simulated in a deterministic way without introducing a trap symbol and therefore causing infinite loops by them:

Let $B = \{l \mid l : (SUB\,(r)\,, l', l'') \in P\}$ and, for every register r,

$$\begin{aligned} \tilde{M}_r &= \left\{\tilde{l} \mid l : (SUB\,(r)\,, l', l'') \in P\right\}, \\ \tilde{M}_r{}^- &= \left\{\tilde{l}^- \mid l : (SUB\,(r)\,, l', l'') \in P\right\}, \\ \hat{M}_r &= \left\{\hat{l} \mid l : (SUB\,(r)\,, l', l'') \in P\right\}, \\ \hat{M}_r{}^- &= \left\{\hat{l}^- \mid l : (SUB\,(r)\,, l', l'') \in P\right\}. \end{aligned}$$

We now take the rules $a_r{}^- \to \tilde{M}_r{}^-\hat{M}_r$ and the annihilation rules $a_r a_r{}^- \to \lambda$ for every register r as well as $\hat{l}\hat{l}^- \to \lambda$ and $\tilde{l}\tilde{l}^- \to \lambda$ for all $l \in B$. Then a SUB-instruction $l_1 : (SUB\,(r)\,, l_2, l_3)$, with $l_1 \in B$, $l_2, l_3 \in B'$, $1 \leq r \leq m$, is simulated by

$$\begin{array}{l} l_1 \to \bar{l}_1 a_r{}^-, \\ \bar{l}_1 \to \hat{l}_1{}^-(\tilde{M}_r \setminus \{\tilde{l}_1\}), \\ \hat{l}_1{}^- \to l_2(\tilde{M}_r{}^- \setminus \{\tilde{l}_1{}^-\}), \text{ and} \\ \tilde{l}_1{}^- \to l_3(\hat{M}_r{}^- \setminus \{\hat{l}_1{}^-\}). \end{array}$$

The symbol $\hat{l}_1{}^-$ generated by the second rule is eliminated again and replaced by $\tilde{l}_1{}^-$ if $a_r{}^-$ is not annihilated.

Again, the proof elaborated for the *max*-mode literally also works for the derivation modes *smax* and max_{rule} without any restrictions as well. □

5 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 *smax*-mode.

5.1 Computational Completeness of Catalytic P Systems

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 [3] for the *max*-mode, and argue why it works for the smax-mode, 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 $l \in B$, we use p_l, and also its variants $\bar{p}_l, \hat{p}_l, \tilde{p}_l$ for $l \in B_{SUB}$, where B_{SUB} denotes the set of labels of SUB-instructions.

$$\begin{aligned}
\Pi &= (O, C, \mu = [\ ]_1, w_1 = c_1 \dots c_m d_1 \dots d_m p_1 w_0, R_1, f = 1),\\
O &= C \cup D \cup E \cup \Sigma\\
&\quad \cup \{\#\} \cup \{p_l \mid l \in B\} \cup \{\bar{p}_l, \hat{p}_l, \tilde{p}_l \mid l \in B_{SUB}\},\\
C &= \{c_i \mid 1 \leq i \leq m\},\\
D &= \{d_i \mid 1 \leq i \leq m\},\\
E &= \{e_i \mid 1 \leq i \leq m\},\\
\Sigma &= \{o_i \mid 1 \leq i \leq d\},\\
R_1 &= \{p_j \to o_r p_k D_m, p_j \to o_r p_l D_m \mid j : (ADD(r), k, l) \in R\}\\
&\quad \cup \{p_j \to \hat{p}_j e_r D_{m,r}, p_j \to \bar{p}_j D_{m,r}, \hat{p}_j \to \tilde{p}_j D'_{m,r},\\
&\qquad\quad \bar{p}_j \to p_k D_m, \tilde{p}_j \to p_k D_m \mid j : (SUB(r), k, l) \in R\}\\
&\quad \cup \{c_r o_r \to c_r d_r, c_r d_r \to c_r, c_{r \oplus_m 1} e_r \to c_{r \oplus_m 1} \mid 1 \leq r \leq m\},\\
&\quad \cup \{d_r \to \#, c_r e_r \to c_r \# \mid 1 \leq r \leq m\}\\
&\quad \cup \{\# \to \#\}.
\end{aligned}$$

Here $r \oplus_m 1$ for $r < m$ 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, $i \in \{1, \dots, m\}$, 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.

In the *smax*-derivation mode only one trap rule $\# \to \#$ will be carried out, but this is the only difference!

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.

For the simulation of instructions, we use:

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

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

Each ADD-instruction $j : (ADD(r), k, l)$, for $r \in \{1, \dots, d\}$, can easily be simulated by the rules $p_j \to o_r p_k D_m$ and $p_j \to o_r p_l D_m$; in parallel, the rules $c_i d_i \to c_i$, $1 \leq i \leq m$, 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 $j : (SUB(r), k, l)$, 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 $j : (SUB(r), k, l)$ if

register r is not empty	register r is empty
$p_j \to \hat{p}_j e_r D_{m,r}$	$p_j \to \bar{p}_j D_{m,r}$
$c_r o_r \to c_r d_r$ $[c_r e_r \to c_r \#]$	c_r should stay idle
$\hat{p}_j \to \tilde{p}_j D'_{m,r}$	$\bar{p}_j \to p_k D_m$
$c_r d_r \to c_r$ $[d_r \to \#]$	$[d_r \to \#]$
$\tilde{p}_j \to p_k D_m$	
$c_{r \oplus_m 1} e_r \to c_{r \oplus_m 1}$	

In the first step of the simulation of each instruction (ADD-instruction, SUB-instruction, and even HALT-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, $1 \leq r \leq m$.

Based on the construction elaborated in [3] and recalled above in sum we have obtained the following result:

Theorem 4. *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 catalytic P system*

$$\Pi = (O, C, \mu = [\ \ ]_1, w_1, R_1, f = 1)$$

working in the max- or the smax-derivation mode and simulating the computations of M such that

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

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R.

5.2 Computational Completeness of Purely Catalytic P Systems

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 [12], 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 for 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. Yet also this proof literally works for the *smax*-derivation mode as well, with the only exception that the trap rule $\# \to \#$ is carried out at most once.

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

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

6.1 P Systems with Label Selection

For all the variants of P systems of type X, we may consider to label all the rules in the sets $R_1, \ldots, R_m$ in a one-to-one manner by labels from a set H and to take a set W containing subsets of H. Then a *P system with label selection* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H, W, f)$$

where $\Pi' = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$ is a P system as defined above, H is a set of labels for the rules in the sets $R_1, \ldots, R_m$, and $W \subseteq 2^H$. In any transition step in Π we first select a set of labels $U \in W$ and then apply a non-empty multiset R of rules such that all the labels of these rules in R are in U in the maximally parallel way, i.e., the set R cannot be extended by any further rule with a label from U so that the obtained multiset of rules would still be applicable to the existing objects in the membrane regions $1, \ldots, m$. The families of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{sequ, asyn, max, smax, max_{rule}, \ldots\}$, computed by P systems with label selection with at most m membranes and rules of type X is denoted by $Y_{\gamma,\delta}OP_m(X, ls)$.

The proof of the following theorem is based on the proof given in [6] for the maximally parallel mode *max*; the proof can be taken over for the mode *smax* word by word; the only difference is that in non-successful computations where more than one trap symbol # has been generated, the trap rule $\# \to \#$ is only applied once.

Theorem 5. *$Y_{\gamma,\delta}OP_1(cat_1, ls) = Y_{\gamma\delta}OP_1(pcat_2, ls) = YRE$ for any $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{max, smax\}$.*

Proof. We only prove the inclusion $PsRE \subseteq Ps_{smax,gen}OP_1(cat_1, ls)$. Let us consider a register machine $M = (n+2, B, l_0, l_h, I)$ with only the first and the second register ever being decremented, and let $A = \{a_1, \ldots, a_{n+2}\}$ be the set of objects for representing the contents of the registers 1 to $n+2$ of M. We construct the following P system:

$$\begin{aligned}
\Pi &= (O, \{c\}, [\ \ ]_1, cdl_0, R_1, H, W, 0),\\
O &= A \cup B \cup \{d, \#\},\\
H &= \{l, l' \mid l \in B\} \cup \left\{l_{\langle x\rangle} \mid x \in \{1, 2, 1', 2', d, \#\}\right\},
\end{aligned}$$

and the rules for R_1 and the sets of labels in W are defined as follows:

A. Let $l_i : (\mathtt{ADD}(r), l_j, l_k)$ be an ADD instruction in I. If $r > 2$, then the (labeled) rules

$$l_i : l_i \to l_j\,(a_r, out)\,,\quad l'_i : l_i \to l_k\,(a_r, out)\,,$$

are used, and for $r \in \{1,2\}$, we take the rules

$$l_i : l_i \to l_j a_r,\quad l'_i : l_i \to l_k a_r.$$

In both cases, we define $\{l_i, l'_i\}$ to be the corresponding set of labels in W. The contents of each register r, $r \in \{1,2\}$, is represented by the number of objects a_r present in the skin membrane; any object a_r with $r \geq 3$ is immediately sent out into the environment.

B. The simulation of a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$, for $r \in \{1,2\}$, is carried out by the following rules and the corresponding sets of labels in W:

For the case that the register r, $r \in \{1,2\}$, is not empty we take the (labeled) rules

$$l_i : l_i \to l_j,\quad l_{\langle r\rangle} : ca_r \to c,\quad l_{\langle d\rangle} : cd \to c\#,$$

(if no symbol a_r is present, i.e., if the register r is empty, then the trap symbol $\#$ is introduced by the rule $l_{\langle d\rangle} : cd \to c\#$).

For the case that the register r is empty, we take the (labeled) rules

$$l'_i : l_i \to l_k,\quad l_{\langle r'\rangle} : ca_r \to c\#$$

(if at least one symbol a_r is present, i.e., if the register r is not empty, then the trap symbol $\#$ is introduced by the rule $l_{\langle r'\rangle} : ca_r \to c\#$).

The corresponding sets of labels to be taken into W are $\left\{l_i, l_{\langle r\rangle}, l_{\langle d\rangle}\right\}$ and $\left\{l'_i, l_{\langle r'\rangle}\right\}$, respectively. In both cases, the simulation of the SUB instruction works correctly if we have made the right choice.

C. As soon as the final label l_h is reached, we apply the rules

$$l_h : l_h \to \lambda,\quad l'_h : cd \to c$$

according to the set of labels $\{l_h, l'_h\}$ in W. In fact, neglecting the single catalyst c, we could even obtain a clean result in the skin membrane when leaving the result objects in the skin membrane instead of sending them out.

D. We also add the labeled rule $l_{\langle\#\rangle} : \# \to \#$ to R_1 and the set $\left\{l_{\langle\#\rangle}\right\}$ to W, hence, the computation cannot halt once the trap symbol $\#$ has been generated.

In sum, we observe that each computation step in M is simulated by exactly one computation step in Π; moreover, such a simulating computation in Π halts if and only if the corresponding computation in M halts (as soon as the label l_h appears, only the set of rules $\{l_h, l'_h\}$ can be applied, and afterwards no rule can be applied anymore in Π, of course, provided that no trap symbol is present).

If at some moment we make the wrong choice when trying to simulate a SUB instruction and have to generate the trap symbol #, the computation will never halt. Hence, we have shown $Ps\left(M\right) = Ps\left(\Pi\right)$, which completes the proof for the catalytic case.

For the purely catalytic case, all the non-cooperative rules are associated with the second catalyst, which immediately yields the corresponding purely catalytic P system with two catalysts. $\square$

6.2 Controlled P Systems and Time-Varying P Systems

Another method to control the application of the labeled rules is to use control languages (see [9] and [4]). A *controlled P system* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H, L, f)$$

where $\Pi' = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f)$ is a P system as defined above, H is a set of labels for the rules in the sets $R_1, \ldots, R_m$, and L is a string language 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 (but again in the maximally parallel way, i.e., we have to apply a multiset R of rules with labels in U_i which cannot be extended by any rule with a label in U_i such that the resulting multiset would still be applicable), 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 $L = \left(U_1 \ldots U_p\right)^*$, 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 $Y_{\gamma,\delta}\left(\Pi\right)$, $Y \in \{N, Ps\}$, 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 $Y_{\gamma,\delta}OP_m\left(X, C\left(FL\right)\right)$ $\left(Y_{\gamma,\delta}OP_m\left(X, wC\left(FL\right)\right)\right)$ and $Y_{\gamma,\delta}OP_m\left(X, TV_p\right)$ $\left(Y_{\gamma,\delta}OP_m\left(X, wTV_p\right)\right)$, respectively, for $\delta \in \{gen, acc\}$ and $\gamma \in \{sequ, asyn, max, smax, max_{rule}, \ldots\}$.

The proof of the following theorem again is taken over for the mode *smax* word by word as given in [6] for the maximally parallel mode *max*.

Theorem 6. *$Y_{\gamma,\delta}OP_1\left(cat_1, \alpha TV_6\right) = Y_{\gamma,\delta}OP_1\left(pcat_2, \alpha TV_6\right) = YRE$, for any $\alpha \in \{\lambda, w\}$, $Y \in \{N, Ps\}$, $\delta \in \{gen, acc\}$, and $\gamma \in \{max, smax\}$.*

Proof. We only prove the inclusion $PsRE \subseteq Ps_{smax,gen}OP_1\left(cat_1, TV_6\right)$. Let us consider a register machine $M = (n + 2, B, l_0, l_h, I)$ with only the first and the second register ever being decremented. Again, we define $A = \{a_1, \ldots, a_{n+2}\}$ and divide the set of labels $B \setminus \{l_h\}$ into three disjoint subsets:

$$B_+ = \{l_i \mid l_i : (\mathtt{ADD}(r), l_j, l_k) \in I\},$$
$$B_{-r} = \{l_i \mid l_i : (\mathtt{SUB}(r), l_j, l_k) \in I\},\ r \in \{1, 2\};$$

moreover, we define $B_- = B_{-1} \cup B_{-2}$ as well as

$$B' = \left\{l, \tilde{l}, \hat{l} \mid l \in B \setminus \{l_h\}\right\} \cup \left\{l^-, l^0, \bar{l}^-, \bar{l}^0, \mid l \in B_-\right\}.$$

The main challenge in the construction for the time-varying P system Π is that the catalyst has to fulfill its task to erase an object a_r, $r \in \{1, 2\}$, for both objects in the same membrane where all other computations are carried out, too; hence, at a specific moment in the cycle of period six, parts of simulations of different instructions have to be coordinated in parallel. The basic components of the time-varying P system Π are defined as follows (we here do not distinguish between a rule and its label):

$$\Pi = (O, \{c\}, [\ \ ]_1, cl_0, R_1 \cup \cdots \cup R_6, R_1 \cup \cdots \cup R_6, (R_1 \ldots R_6)^*, 0),$$
$$O = A \cup \{a_1', a_2'\} \cup B' \cup \{c, h, l_h, \#\}.$$

We now list the rules in the sets of rules R_i to be applied in computation steps $6n + i$, $n \geq 0$, $1 \leq i \leq 6$:

$\mathbf{R}_1$: in this first step of the cycle, especially all the ADD instructions are simulated, i.e., for each $l_i : (\mathtt{ADD}(r), l_j, l_k) \in I$ we take

$cl_i \to ca_r\tilde{l}_j$, $cl_i \to ca_r\tilde{l}_k$ for $r \in \{1, 2\}$ as well as $cl_i \to c(a_r, out)\tilde{l}_j$, $cl_i \to c(a_r, out)\tilde{l}_k$ for $3 \leq r \leq n + 2$ (in order to obtain the output in the environment, for $r \geq 3$ we have to take (a_r, out) instead of a_r); only in the sixth step of the cycle, from $\tilde{l}_j$ and $\tilde{l}_k$ the corresponding unmarked labels l_j and l_k will be generated;

$cl \to cl^-$, $cl \to cl^0$ initiate the simulation of a SUB instruction for register 1 labeled by $l \in B_{-1}$, i.e., we make a non-deterministic guess whether register r is empty (with introducing l^0) or not (with introducing l^-);

$cl \to c\hat{l}$ marks a label $l \in B_{-2}$ (the simulation of such a SUB instruction for register 2 will start in step 4 of the cycle);

$\# \to \#$ keeps the trap symbol $\#$ alive guaranteeing an infinite loop once $\#$ has been generated;

$h \to \lambda$ eliminates the auxiliary object h which eventually has been generated two steps before (h is needed for simulating the decrement case of SUB instructions).

$\mathbf{R}_2$: in the second and the third step, the SUB instructions on register 1 are simulated, i.e., for all $l \in B_{-1}$ we start with

$ca_1 \to ca_1'$ (if present, exactly one copy of a_1 can be primed, but only if a label l^- for some l from B_{-1} is present) and

$l^- \to \bar{l}^- h$, $l^0 \to \bar{l}^0$ for all $l \in B_{-1}$;

all other labels $\tilde{l}$ for $l \in B$ block the catalyst c from erasing a copy of a_1 by forcing the application of the corresponding rules $c\tilde{l} \to c\tilde{l}$ for c in order to avoid the introduction of the trap symbol # by the enforced application of a rule $\tilde{l} \to \#$, i.e., we take

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$, and

$c\hat{l} \to c\hat{l}$, $\hat{l} \to \#$ for all $l \in B_{-2}$;

$\# \to \#$ keeps the computation alive once the trap symbol has been introduced.

$\mathbf{R}_3$: for all $l_i : (\mathtt{SUB}\,(1)\,, l_j, l_k) \in I$ we take

$c\bar{l}_i^0 \to c\tilde{l}_k$, $a_1' \to \#$, $\bar{l}_i^0 \to \#$ (zero test; if a primed copy of a_1 is present, then the trap symbol # is generated);

$\bar{l}_i^- \to \tilde{l}_j$, $ca_1' \to c$, $ch \to c\#$ (decrement; the auxiliary symbol h is needed to keep the catalyst c busy with generating the trap symbol # if we have taken the wrong guess when assuming the register 1 to be non-empty);

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$ (with these labels, we just pass through this step);

$c\hat{l} \to c\hat{l}$, $\hat{l} \to \#$ for all $l \in B_{-2}$ (these labels pass through this step to become active in the next step);

$\# \to \#$.

$\mathbf{R}_4$: in the fourth step, the simulation of SUB instructions on register 2 is initiated by using

$c\hat{l} \to cl^-$, $c\hat{l} \to cl^0$ for all $l \in B_{-2}$, i.e., we make a non-deterministic guess whether register r is empty (with introducing l^0) or not (with introducing l^-);

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$ (with all other labels, we only pass through this step);

$\# \to \#$,

$h \to \lambda$ (if h has been introduced by $l^- \to \bar{l}^- h$ in the second step for some $l \in B_{-1}$, we now erase it).

$\mathbf{R}_5$: in the fifth and the sixth step, the SUB instructions on register 2 are simulated, i.e., for all $l \in B_{-2}$ we start with

$ca_2 \to ca_2'$ (if present, exactly one copy of a_2 can be primed) and

$l^- \to \bar{l}^- h$, $l^0 \to \bar{l}^0$ for all $l \in B_{-2}$;

$c_1\tilde{l} \to c_1\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B$;

$\# \to \#$.

$\mathbf{R}_6$: the simulation of SUB instructions $l_i : (\mathtt{SUB}\,(2)\,, l_j, l_k) \in I$ on register 2 is finished by

$c\bar{l}_i^0 \to cl_k$, $a_2' \to \#$, $\bar{l}_i^0 \to \#$ (zero test; if a primed copy of a_2 is present, then the trap symbol # is generated);

$\bar{l}_i^- \to l_j$, $ca_2' \to c$, $ch \to c\#$ (decrement; the auxiliary symbol h is needed to keep the catalyst c busy with generating the trap symbol # if we have taken the wrong guess when assuming the register 2 to be non-empty; if it is not used, it can be erased in the next step by using $h \to \lambda$ in R_1);

$c\tilde{l} \to cl$, $\tilde{l} \to \#$ for all $l \in B$;

$\# \to \#$.

Without loss of generality, we may assume that the final label l_h in M is only reached by using a zero test on register 2; then, at the beginning of a new cycle, after a correct simulation of a computation from M in the time-varying P system Π no rule will be applicable in R_1 (another possibility would be to take $c\bar{l}_i^0 \to c$ instead of $c\bar{l}_i^0 \to cl_h$ in R_6).

At the end of the cycle, in case all guesses have been correct, the requested instruction of M has been simulated and the label of the next instruction to be simulated is present in the skin membrane. Only in the case that M has reached the final label l_h, the computation in Π halts, too, but only if during the simulation of the computation of M in Π no trap symbol # has been generated; hence, we conclude $Ps\,(M) = Ps\,(\Pi)$.

For the purely catalytic case, all the non-cooperative rules are associated with the second catalyst, which immediately yields the corresponding purely catalytic P system with two catalysts. □

7 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 $Y_{\gamma,gen}O_{tox}P_m\,(X)$, for any $m \geq 1$, $\gamma \in \{sequ, asyn, max, smax, max_{rule}\}$, and $Y \in \{N, Ps\}$.

The following theorem stated in [2] only for the *max*-mode obviously hols for the *smax*-mode, too.

Theorem 7. *For* $\beta \in \{max, smax\}$*,*

$$PsRE = Ps_{\beta,gen}O_{tox}P_1([p]cat_2).$$

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 omitting 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*.

8 Atomic Promoters and Inhibtors

As shown in [11], 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 [10], the computational completeness of P systems with non-cooperative rules and atomic promoters has been shown. In the following we will establish a new proof 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 show a new pretty surprising result, establishing computational completeness of P systems with non-cooperative rules and atomic inhibitors, and 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.

8.1 Atomic Promoters

We now establish our new proof for the computational completeness of P systems with non-cooperative rules and atomic promoters when using the derivation mode *smax*; the overall number of promoters only is $5m$ where m is the number of decrementable registers of the simulated register machine.

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 atomic inhibitors*

$$\Pi = (O, \mu = [\ \ ]_1, w_1 = l_0, R_1, f = 1)$$

working in the smax- or max_{rule}-derivation mode and simulating the computations of M such that

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

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R; moreover, 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 $p \to q o_r$ and $p \to s o_r$.

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. □

8.2 Atomic Inhibtors

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

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 atomic inhibitors*

$$\Pi = (O, \mu = [\ \ ]_1, w_1 = l_0, R_1, f = 1)$$

a P system with atomic inhibitors $\Pi = (O, \mu = [\ \ ]_1, w_1 = l_0, R_1, f = 1)$ working in the smax- or max_{rule}-derivation mode and simulating the computations of M such that

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

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R; moreover, 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.

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

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 $p \to q o_r D$ and $p \to s o_r D$.

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

1. $p \to p' D_r$;
2. $p' \to p'' D d_0$; in parallel, the following rules are used:
   $o_r \to o'_r \mid_{\neg d_r}$, $d_k \to \lambda$, $1 \le k \le m$;
3. $p'' \to \tilde{p} D \mid_{\neg o'_r}$; $o'_r \to \lambda$, $d_0 \to \lambda$;
   again, in parallel the rules $d_k \to \lambda$, $1 \le k \le m$, are used;
4. $p'' \to q D \mid_{\neg d_0}$, $\tilde{p} \to s D$.

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. □

9 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. We show that for P systems with target selection in the derivation mode *smax no* catalyst is needed any more, and with max_{rule}, we even obtain a deterministic simulation of deterministic register machines.

Theorem 10. *For any register machine $M = (d, B, l_0, l_h, R)$, with $m \le d$ being the number of decrementable registers, we can construct a P system with non-cooperative rules working in the smax-derivation mode 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. Clearly, these conditions do not restrict the generality. We construct the following P system Π simulating M.

The correct behavior of the object associated to 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.

The value of each register r is represented by the multiplicity of objects o_r in the skin. 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 we will in the following refer to as the *guards*, or an object associated to the label of the simulated instruction, which we will in the following 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.

$$
\begin{aligned}
\Pi &= (O, \mu, w_1, \cdots, w_{m+2}, R_1, \cdots, R_{m+2}) \text{ where} \\
O &= \{o_r \mid 1 \leq r \leq d\} \cup \{\bar{p}, p \mid p \in B\} \cup \{p', p''\hat{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 \leq r \leq 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 \leq j \leq m+2, \\
R_{1,1,s} &= \{e \to e', e' \to e'', e'' \to \hat{e}, \hat{e} \to \hat{e}', 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 \leq r \leq m, \\
R_{1,1,\#} &= \{p' \to \#, p''_0 \to \#, p'_- \to \# \mid p \in B_{SUB}\} \cup \{\hat{p} \to \# \mid p \in B_{ADD}\} \\
&\quad \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}'\} \\
&\quad\ \cup \{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 \neq r\} \\
&\quad \cup \{p' \to (\#, in_{r+2}) \mid p \in B_{SUB}\},\ 1 \leq r \leq 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\} \cup \{\mid p \in B\} \\
&\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 \leq r \leq m \\
R_{r+2,r+2,\#} &= \{\# \to (\#, out)\},\ R_{r+1,r+1,s} = R_{r+1,r+1,\#} = \emptyset.
\end{aligned}
$$

Most trapping rules, given in brackets in the tables below and listed in rule groups $R_{i,j,\#}$ above, 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, if the zero-test is chosen while d is present (signifying that the register was decremented), causing a target conflict: either p_0 or d will be anyway rewritten into #. However, if the decrement is chosen while d is absent (signifying that the register was zero), then p_- will appear in the skin in step 4 instead of step 5, causing a target conflict: either p'_- or e'' will be anyway rewritten into #.

Below we present the tables describing the simulation of instructions of M. An application of one of the rules given in brackets leads to non-halting computations, not contributing to the result.

$$(p : (SUB(r), q, s))$$

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

	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 $t > 1$]	$p'_0 \to p''_0$ $e'' \to \hat{e}$ $[p'_0 \to (\#, in_t)]$ $[e'' \to (\#, in_t)]$ [for $t > 1$]	$p_- \to (p'_-, out)$	-
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	$\bar{q} \to q$ $[\bar{q} \to (\#, in_t)]$	$\bar{s} \to s$ $[\bar{s} \to (\#, in_t)]$	$\bar{e} \to e$	

$(p : (ADD(r), q, s))$

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

Auxiliary rules

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

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 $r + 2$. □

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 11. *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*

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

working in the max_{rule}-derivation mode and simulating the computations of M such that

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

where $ADD^1(R)$ denotes the number of deterministic ADD-instructions in R, $ADD^2(R)$ denotes the number of non-deterministic ADD-instructions in R, and $SUB(R)$ denotes the number of SUB-instructions in R.

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 be able to distinguish between the decrement and the zero test case. For each r, 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 membrane 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 $R(l_0)$.

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

An ADD-instruction $p : (ADD(r), q, s)$ is simulated by the rules $p \to R(q)o_r$ and $p \to R(s)o_r$ as well as the rules $e \to \lambda$ and $e' \to \lambda$. This combination of three rules supercedes any combination of rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$, for some $1 \leq r \leq m$.

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 tuple 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 superceding the five rules which could be used for the zero test case.
2. In the second step, the following three or four rules, again superceding any combination of rules $o_r \to (\lambda, in_{r+1})$ and $o'_r \to (\lambda, in_{r+1})$ for some $1 \leq r \leq m$, 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 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}: $p \to (R(s), out)$.

We finally point out that the simulation of the SUB-instructions works deterministically, hence, although the P system itself is not deterministic syntacticly, it works in a deterministic way if the underlying register machine is deterministic. □

10 Conclusion and Future Work

It is not very surprising that the proofs we have checked in the preceding sections also work for the derivation mode *smax*, as many constructions elaborated for the derivation mode *max* 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 max_{rule}.

- There are many models of P systems for which the maximally parallel derivation mode has been used, especially for showing computational completeness.
- As we have seen by careful inspection of several proofs for computational completeness, many results established with using the maximally parallel derivation mode literally hold true as well for the derivation modes *smax* and max_{rule}.
- Many other constructions working in the maximally parallel derivation mode have to be checked carefully if they work for the derivation modes *smax* and max_{rule}, too.
- For some proofs having been established in the maximally parallel derivation mode we might need completely new proofs or proof techniques for the set-maximally parallel derivation mode; one such example is the proof for P systems with target selection.
- Some variants of P systems become even stronger with the mode *smax*; as already pointed out by Gheorghe Păun, P systems with non-cooperative rules and atomic promoters are computationally complete with the *smax*-mode, also see [10], and in this paper we have shown a new proof for this computational completeness result and even shown a similar result for P systems with non-cooperative rules and atomic inhibitors.
- On the other hand, eventually, some results stablished in the maximally parallel derivation mode are not valid any more for the set-maximally parallel derivation mode.

References

1. A. Alhazov, B. Aman, R. Freund, and Gh. Păun. Matter and anti-matter in membrane systems. In *Proceedings of the Twelfth Brainstorming Week on Membrane Computing*, pages 1–26, 2014.
2. A. Alhazov and R. Freund. P systems with toxic objects. In M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, and C. Zandron, editors, *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 99–125. Springer, 2014.
3. A. Alhazov and R. Freund. Small catalytic P systems. In M. J. Dinneen, editor, *Proceedings of the Workshop on Membrane Computing 2015 (WMC2015), (Satellite workshop of UCNC2015), August 2015*, volume CDMTCS-487 of *CDMTCS Research Report Series*. Centre for Discrete Mathematics and Theoretical Computer, ScienceDepartment of Computer Science University of Auckland, Auckland, New Zealand, 2015.
4. A. Alhazov, R. Freund, H. Heikenwälder, M. Oswald, Yu. Rogozhin, and S. Verlan. Sequential P systems with regular control. In E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, and Gy. Vaszil, editors, *Membrane Computing - 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*, volume 7762 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2013.
5. H. Burkhard. Ordered firing in petri nets. *Elektronische Informationsverarbeitung und Kybernetik*, 17(2/3):71–86, 1981.
6. R. Freund and Gh. Păun. How to obtain computational completeness in P systems with one catalyst. In T. Neary and M. Cook, editors, *Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9-11, 2013*, volume 128 of *EPTCS*, pages 47–61, 2013.
7. R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing. 8th International Workshop, WMC 2007 Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2007.
8. P. Frisco and G. Govan. P systems with active membranes operating under minimal parallelism. In M. Gheorghe, Gh. Păun, G. Rozenberg, A. Salomaa, and S. Verlan, editors, *Membrane Computing - 12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers*, volume 7184 of *Lecture Notes in Computer Science*, pages 165–181. Springer, 2011.
9. K. Krithivasan, Gh. Păun, and A. Ramanujan. On controlled P systems. In L. Valencia-Cabrera, M. García-Quismondo, L. Macías-Ramos, M. Martínez-del-Amor, Gh. Păun, and A. Riscos-Núñez, editors, *Proceedings 11th Brainstorming Week on Membrane Computing, Sevilla, 4–8 February 2013*, pages 137–151. Fénix Editora, Sevilla, 2013.
10. L. Pan, Gh. Păun, and B. Song. Flat maximal parallelism in P systems with promoters. *Theoretical Computer Science*, 2015, to appear.
11. D. Sburlan. Further results on P systems with promoters/inhibitors. *International Journal of Foundations of Computer Science*, 17(1):205–221, 2006.
12. P. Sosík and M. Langer. Small catalytic P systems simulating register machines. *Theoretical Computer Science*, accepted, 2015.

13. S. Verlan and J. Quiros. Fast hardware implementations of P systems. In E. Csuhaj-Varjú, M.Gheorghe, G. Rozenberg, A. Salomaa, and Gy. Vaszil, editors, *Membrane Computing. 13th International Conference, CMC 2012, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers*, volume 7762 of *Lecture Notes in Computer Science*, pages 404–423. Springer, 2013.