Universal P Systems: One Catalyst Can Be Sufficient

Rudolf Freund[1] and Gheorghe Păun[2]

[1] Technische Universität Wien, Institut für Computersprachen
Favoritenstr. 9, A-1040 Wien, Austria
rudi@emcc.at

[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, ghpaun@gmail.com

Summary. Whether P systems with only one catalyst can already be universal, is still an open problem. Here we establish universality (computational completeness) by using specific variants of additional control mechanisms. At each step using only multiset rules from one set of a finite number of sets of rules allows for obtaining computational completeness with one catalyst and only one membrane. If the targets are used for choosing the multiset of rules to be applied, for getting computational completeness with only one catalyst more than one membrane is needed. If the available sets of rules change periodically with time, computational completeness can be obtained with one catalyst in one membrane. Moreover, we also improve existing computational completeness results for P systems with mobile catalysts and for P systems with membrane creation.

1 Introduction

P systems with catalytic rules were already considered in the originating papers for membrane systems, see [9]. In [3] two catalysts were shown to be sufficient for getting universality/computational completeness (throughout this paper, with these notions we will indicate that all recursively enumerable sets of (vectors of) non-negative integers can be generated). Since then, it has become one of the most challenging open problems in the area of P systems, whether or not one catalyst might already be enough to obtain computational completeness.

Using additional control mechanisms as, for example, priorities or promoters/inhibitors, P systems with only one catalyst can be shown to be computationally completene, e.g., see Chapter 4 of [11]. On the other hand, additional features for the catalyst may be taken into account; for example, we may use bi-stable catalysts (catalysts switching between two different states) or mobile catalysts (catalysts able to cross membranes). Moreover, additional membrane features may be used, for example, membrane creation or controlling the membrane permeability by means of the operations δ and τ.

P systems with membrane creation were introduced in [8], showing both their universality and efficiency (the Hamiltonian path problem is solved in linear time in a semi-uniform way; this result was improved in [4], where a polynomial solution to the Subset Sum problem in a uniform way is provided). For proving universality, in [8] (Theorem 2) P systems starting with one membrane, having four membranes at some time during the computation, using one catalyst, and also controlling the membrane permeability by means of the operations δ (deleting the surrounding membrane) and τ (increasing the thicknes of the surrounding membrane, i.e., making it impermeable for objects to pass through) are needed. However, as already shown in [10], P systems with one catalyst and using the operations δ and τ are universal, i.e., the membrane creation facility is not necessary for getting universality in this framework. Here we improve the result shown in [8] from two points of view: (i) the control of membrane permeability is not used, and (ii) the maximal number of membranes used during a computation is two.

P systems with mobile catalysts were introduced in [5], and their universality was proved with using three membranes and target indications of the forms *here*, *out*, and in_j. We here improve this result by replacing the target indications in_j with the weaker one *in*.

Recently, several variants of P systems using only one catalyst together with control mechanisms for choosing the rules applicable in a computation step have been considered: for example, in [6] the rules are labeled with elements from an alphabet H and in each step a maximal multiset of rules having the same label from H is applied. In this paper, we will give a short proof for the universality of these *P systems with label selection* with only one catalyst in a single membrane. As a specific variant, for each membrane we can choose the rules according to the target indications, and we will prove universality for these *P systems with target selection* with only one catalyst, but needing more than one membrane (such systems with only one membrane lead to the still open problem of catalytic P systems with one catalyst).

Regular control languages were considered already in [6] for the maximally parallel derivation mode, whereas in [1] universality was proved for the sequential mode: there even only non-cooperative rules were needed in one membrane for time-varying P systems to obtain universality (in time-varying systems, the set of available rules varies periodically with time, i.e., the regular control language is of the very specific form $W = (U_1 \dots U_p)^*$, allowing to apply rules from a set U_i in the computation step $pn + i$, $n \geq 0$; p is called the *period*), but a bounded number of steps without applying any rule had to be allowed. We here prove that *time-varying P systems* using the maximally parallel derivation mode in one membrane with only one catalyst are computationally complete with a period of six and the usual halting when no rule can be applied.

2 Prerequisites

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

A (finite) multiset over the (finite) alphabet V, $V = \{a_1, \cdots, a_n\}$, is a mapping $f : V \longrightarrow \mathbb{N}$ and represented by $\langle f\,(a_1), a_1\rangle \cdots \langle f\,(a_n), a_n\rangle$ or by any string x the Parikh vector of which with respect to $a_1, \cdots, a_n$ is $(f\,(a_1), \cdots, f\,(a_n))$. In the following we will not distinguish between a vector $(m_1, \cdots, m_n)$, its representation by a multiset $\langle m_1, a_1\rangle \cdots \langle m_n, a_n\rangle$ or its representation by a string x having the Parikh vector $\left(|x|_{a_1}, \cdots, |x|_{a_n}\right) = (m_1, \cdots, m_n)$. Fixing the sequence of symbols $a_1, \cdots, a_n$ in the alphabet V in advance, the representation of the multiset $\langle m_1, a_1\rangle \cdots \langle m_n, a_n\rangle$ by the string $a_1^{m_1} \cdots a_n^{m_n}$ is unique. The set of all finite multisets over an alphabet V is denoted by V°.

The family of regular and recursively enumerable string languages is denoted by REG and RE, respectively. For more details of formal language theory the reader is referred to the monographs and handbooks in this area as [2] and [12].

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

- $l_1 : (ADD\,(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  Increase the value of register j by one, and non-deterministically jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB\,(j), l_2, l_3)$, with $l_1 \in B \setminus \{l_h\}$, $l_2, l_3 \in B$, $1 \leq j \leq m$
  If the value of register j is zero then jump to instruction l_3, otherwise decrease the value of register j by one and jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stop the execution of the register machine.

A *configuration* of a register machine is described by the contents of each register and by the value of the current label, which indicates the next instruction to be executed. Computations start by executing the first instruction of P (labeled with l_0), and terminate with reaching the $HALT$-instruction.

Register machines provide a simple universal computational model [7]. In the generative case as we need it later, we start with empty registers, use the first two registers for the necessary computations and take as results the contents of the k registers 3 to $k+2$ in all possible halting computations; during a computation of M, only the registers 1 and 2 can be decremented. In the following, we shall call a specific model of P systems *computationally complete* or *universal* if and only if for any (generating) register machine M we can effectively construct an equivalent P system Π of that type simulating each step of M in a bounded number of steps and yielding the same results.

2.1 P Systems

The basic ingredients of a (cell-like) P system are the membrane structure, the objects placed in the membrane regions, and the evolution rules. The *membrane structure* is a hierarchical arrangement of membranes. Each membrane defines a *region/compartment*, the space between the membrane and the immediately inner membranes; the outermost membrane is called the *skin membrane*, the region outside is the *environment*, also indicated by (the label) 0. Each membrane can be labeled, and the label (from a set *Lab*) will identify both the membrane and its region. The membrane structure can be represented by a rooted tree (with the label of a membrane in each node and the skin in the root), but also by an expression of correctly nested labeled parentheses. The *objects* (multisets) are placed in the compartments of the membrane structure and usually represented by strings, with the multiplicity of a symbol corresponding to the number of occurrences of that symbol in the string. *The evolution rules* are multiset rewriting rules of the form $u \to v$, where u is a multiset of objects from a given set O and $v = (b_1, tar_1) \dots (b_k, tar_k)$ with $b_i \in O$ and $tar_i \in \{here, out, in\}$ or $tar_i \in \{here, out\} \cup \{in_j \mid j \in Lab\}$, $1 \leq i \leq k$. Using such a rule means "consuming" the objects of u and "producing" the objects $b_1, \dots, b_k$ of v; the *target indications here*, *out*, and *in* mean that an object with the target *here* remains in the same region where the rule is applied, an object with the target *out* is sent out of the respective membrane (in this way, objects can also be sent to the environment, when the rule is applied in the skin region), while an object with the target *in* is sent to one of the immediately inner membranes, non-deterministically chosen, wheras with in_j this inner membrane can be specified directly. In general, we omit the target indication *here*.

Formally, a (cell-like) P system is a construct

$$\Pi = (O, \mu, w_1, \dots, w_m, R_1, \dots, R_m, f)$$

where O is the alphabet of objects, 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 evolution rules, associated with the regions of μ, and f is the label of the membrane region from which the outputs are taken ($f = 0$ indicates that the output 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.* In *catalytic P systems* we use non-cooperative as well as *catalytic rules* which are of the form $ca \to cv$, where c is a special object which never evolves and never passes through a membrane (both these restrictions can be relaxed), but it just assists object a to evolve to the multiset v. In a *purely catalytic P system* we only allow catalytic rules. In both catalytic and purely catalytic P systems, we replace O by O, C in order to specify those objects from O which are the catalysts in the set C.

The evolution rules are used in the *non-deterministic maximally parallel* way, i.e., in any computation step of Π we choose a multiset of rules from 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 membrane regions $1, \dots, m$.

The membranes and the objects present in the compartments of a system at a given time form a *configuration*; starting from a given *initial configuration* and using the rules as explained above, we get *transitions* among configurations; a sequence of transitions forms a *computation.* A computation is *halting* if it reaches a configuration where no rule can be applied. With a halting computation we associate a *result*, in the form of the number of objects present in membrane f in the halting configuration. The set of vectors of non-negative integers and the set of (Parikh) vectors of non-negative integers obtained as results of halting computations in Π are denoted by $N(\Pi)$ and $Ps(\Pi)$, respectively.

The family of sets $Y(\Pi)$, $Y \in \{N, Ps\}$, computed by P systems with at most m membranes and cooperative rules and with non-cooperative rules is denoted by $YOP_m(coop)$ and $YOP_m(ncoo)$, respectively. It is well known that for any $m \geq 1$, $YREG = YOP_m(ncoo) \subset NOP_m(coop) = YRE$, see [9].

The family of sets $Y(\Pi)$, $Y \in \{N, Ps\}$, computed by (purely) catalytic P systems with at most m membranes and at most k catalysts is denoted by $YOP_m(cat_k)$ ($YOP_m(pcat_k)$); from [3] we know that, with the results being sent to the environment in order to avoid the discussion how to count the catalysts in the skin membrane, we have $YOP_1(cat_2) = YOP_1(pcat_3) = YRE$.

If we allow catalysts to move from one membrane region to another one, then we speak of *P systems with mobile catalysts.* The families of sets $N(\Pi)$ and $Ps(\Pi)$ computed by P systems with at most m membranes and k mobile catalysts is denoted by $NOP_m(mcat_k)$ and $PsOP_m(mcat_k)$, respectively.

For all the variants of P systems using rules of some type X as defined above, we may consider systems containing only rules of the form $u \to v$ where $u \in O$ and $v = (b_1, tar) \dots (b_k, tar)$ with $b_i \in O$ and $tar \in \{here, out, in\}$ or $tar \in \{here, out\} \cup \{in_j \mid j \in H\}$, $1 \leq i \leq k$, i.e., in each rule there is only one target for all objects b_i; if *catalytic rules* are considered, then we request the rules to be of the form $ca \to c(b_1, tar) \dots (b_k, tar)$. *P systems with target selection* contain only these forms of rules; moreover, in each computation step, for each membrane region i we choose a maximal non-empty (if it exists) multiset of rules from R_i having the same target indication tar (for different membranes these targets may be different). The families of sets $N(\Pi)$ and $Ps(\Pi)$ computed by P systems with target selection with at most m membranes and rules of type X are denoted by $NOP_m(X, ts)$ and $PsOP_m(X, ts)$, respectively.

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 and 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 family of sets $N(\Pi)$ and $Ps(\Pi)$ computed by P systems with label selection with at most m membranes and rules of type X is denoted by $NOP_m(X, ls)$ and $PsOP_m(X, ls)$, respectively.

Another method to control the application of the labeled rules is to use control languages (see [6] and [1]). A *controlled P system* 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 is a string language over 2^H from a family FL. Every successful computation in Π has to follow a control word $U_1 \ldots U_n \in W$: in transition step i, only rules with labels in U_i are allowed to be applied, and after the n-th transition, the computation halts; we may relax this end condition, and then we speak of *weakly controlled P systems.* If $W = (U_1 \ldots U_p)^*$, 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 $Y(\Pi)$, $Y \in \{N, Ps\}$, 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 $YOP_m(X, C(FL))$ $(YOP_m(X, wC(FL)))$ and $YOP_m(X, TV_p)$ $(YOP_m(X, wTV_p))$, respectively.

In the *P systems with membrane creation* considered in this paper, besides the catalytic rules $ca \to c(u, tar)$ and the non-cooperative rules $a \to (u, tar)$ we also use catalytic membrane creation rules of the form $ca \to c[\ u\ ]_i$ (in the context of c, from the object a a new membrane with label i containing the multiset u is generated) and membrane dissolution rules $a \to u\delta$ (we assume that no objects can be sent into a membrane which is going to be dissolved; with dissolving the membrane i by applying δ, all objects contained inside this membrane are collected in the region surrounding the dissolved membrane); in all cases, c is a catalyst, a is an object, u is a multiset, and tar is a target indication of the form $here$, out, and in_j. The family of sets $Y(\Pi)$, $Y \in \{N, Ps\}$, computed by such P systems with membrane creation and using at most k catalysts, with m initial membranes and having at most h membranes during its computations is denoted by $YP_{m,h}(cat_k, mcre)$.

3 Computational Completeness of P Systems with Label Selection

Theorem 1. *$YOP_1(cat_1, ls) = YRE$, $Y \in \{N, Ps\}$.*

Proof. We only prove the inclusion $PsRE \subseteq PsOP_1(cat_1, ls)$. 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:

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

and the sets of labels in W and the rules for R_1 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 introduced, and for $r \in \{1, 2\}$, we introduce 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 wit $r > 2$ 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 empty we take the (labeled) rules

$$l_i : l_i \to l_k, \quad l_r : ca_r \to c, \quad l_d : cd \to c\#,$$

(if no symbol a_r is present, i.e., if the register r is empty, then the trap symbol # is introduced) and for the case that the register r is not empty, we introduce the rules

$$l_i' : l_i \to l_j, \quad l_r' : ca_r \to c\#$$

(if at least one symbol a_r is present, i.e., if the register r is not empty, then the trap symbol # is introduced); the corresponding sets of labels to be taken into W are $\{l_i, l_r, l_d\}$ and $\{l'_i, l'_r\}$, respectively. In both cases, the simulation of the SUB instruction works correctly, if we have made the right choice.

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

In sum, we have the equality $Ps\,(M) = Ps\,(\Pi)$, which completes the proof. $\square$

4 Computational Completeness of P Systems with Target Selection

Theorem 2. *$YOP_7\,(cat_1, ts) = YRE$, $Y \in \{N, Ps\}$.*

Proof. We only prove the inclusion $PsRE \subseteq PsOP_7\,(cat_1, ts)$. 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. The set of labels $B \setminus \{l_h\}$ is divided into three disjoint subsets:

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

moreover, we define $B_- = B_{-1} \cup B_{-2}$, $B'_- = \{l' \mid l \in B_-\}$, $B''_- = \{l'' \mid l \in B_-\}$, and $B' = B_+ \cup B_- \cup B'_- \cup B''_-$ as well as $A = \{a_1, \ldots, a_{n+2}\}$. We construct the following P system:

$$\Pi = (O, \{c\}, [\,[\;\;]_2 \ldots [\;\;]_7\,]_1, w_1, \ldots, w_7, R_1, \ldots, R_7, 0),$$
$$O = A \cup \{a'_1, a'_2\} \cup B' \cup \{c, d, \#\},$$

with $w_1 = l_0$, $w_2 = c$, and $w_i = \lambda$ for $3 \le i \le 7$. In order to make argumentation easier, in the following we refer to the membrane labels 1 to 7 according to the following table:

1	2	3	4	5	6	7
skin	$-$	0_1	0_2	$-_1$	$-_2$	$+$

The sets of rules now are constructed as follows:

A. The simulation of any instruction from I starts in the skin membrane with moving all objects except the output symbols a_r for $r > 2$ into an inner membrane; according to the definition, taking the target *in* means non-deterministically choosing one of the inner membranes, but the same membrane for all objects to be moved in. The output symbols a_r for $r > 2$ are sent out into the environment by $a_r \to (a_r, out)$, thus yielding the result of a halting computation as the number of symbols a_r sent out into the environment during this computation. Hence, in sum we get

$$R_1 = \{x \to (x, in) \mid x \in B_+ \cup B_- \cup \{a_1, a_2, a_1', a_2', \#\}\} \cup \left\{x \to (xd, in) \mid x \in B_-'\right\}$$
$$\cup \; \{a_r \to (a_r, out) \mid 3 \le r \le n+2\}\,.$$

B. For the simulation of an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k) \in I$ all non-terminal symbols (all symbols except a_r for $r > 2$) are expected to have been sent to membrane $+$:

$$\begin{aligned} R_+ = \;& \{l_i \to (l_j a_r, out), l_i \to (l_k a_r, out) \mid l_i : (\mathtt{ADD}(r), l_j, l_k) \in I\} \\ & \cup \; \{l \to (\#, out) \mid l \in B' \setminus B_+\} \\ & \cup \; \{x \to (x, out) \mid x \in \{a_1, a_2, \#\}\}\,. \end{aligned}$$

If the symbols arrive in membrane $+$ with a label $l \in B' \setminus B_+$, then the trap symbol # is generated and the computation will never halt. Sending out all terminal symbols a_r for $r > 2$ from the skin membrane can be done as a last step of a successful computation, but we may also choose to send out all those present there at a specific moment instead of immediately continuing the simulation of an instruction of the register machine. Hence, the simulation of an ADD instruction by Π takes at most three steps.

C. The simulation of a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is carried out in two steps for the zero test, i.e., when the register r is empty, using (the rules in) membrane 0_r, and in five steps for decrementing the number of symbols a_r, first using membrane $-_r$ to mark the corresponding symbols a_r into a_r' and then using the catalyst c in membrane $-$ to erase one of these primed objects; the marking procedure is necessary to guarantee that the catalyst erases the correct object. For $r \in \{1, 2\}$, we define the following sets of rules:

$$\begin{aligned} R_{0_r} = \;& \{l_i \to (l_k, out), a_r \to (\#, out) \mid l_i : (\mathtt{SUB}(r), l_j, l_k) \in I\} \\ & \cup \; \{l \to (\#, out) \mid l \in B' \setminus B_{-r}\} \\ & \cup \; \{x \to (x, out) \mid x \in \{a_{3-r}, \#\}\}\,. \end{aligned}$$

If the number of objects a_r is not zero, i.e., if the register r is not empty, the introduction of the trap symbol # causes the computation to never halt. On the other hand, if we want to decrement the register, we have to guarantee that exactly one symbol a_r is erased:

$$\begin{aligned} R_{-_r} = \;& \{l_i \to (l_i', out) \mid l_i \in B_{-r}\} \cup \{a_r \to (a_r', out)\} \\ & \cup \; \{l \to (\#, out) \mid l \in B' \setminus B_{-r}\} \\ & \cup \; \{x \to (x, out) \mid x \in \{a_{3-r}, \#\}\}\,. \end{aligned}$$

The whole multiset of objects, with the primed versions of l_i and the a_r, via the skin membrane now has to enter membrane $-$; here the dummy symbol d guarantees that the catalyst cannot do nothing if no primed symbol a_r' has arrived; again the generation of # causes the computation to not halt anymore:

$$\begin{aligned} R_- = \;& \left\{l_i' \to l_j'', c a_r' \to c, l_i'' \to \#, l_i'' \to (l_j, out) \mid l_i : (\mathtt{SUB}(r), l_j, l_k) \in I\right\} \\ & \cup \; \{cd \to c\#, d \to (\lambda, out)\} \cup \{a_r' \to (a_r, out) \mid r \in \{1, 2\}\}\,, \\ & \cup \; \left\{l \to (\#, out) \mid l \in B' \setminus B_-''\right\} \\ & \cup \; \{x \to (x, out) \mid x \in \{a_{3-r}, \#\}\}\,. \end{aligned}$$

In R_-, for correctly continuing the simulation of a SUB instruction, exactly two steps have to be carried out:

In the first step, the target indication *here* has to be used with applying the two rules $l_i' \to l_j''$ and $ca_r' \to c$ (eliminating exactly one copy of a_r', i.e., decrementing register r) and leaving all other objects unchanged; if instead the target indication *out* were chosen, the forced application of the rule $l_i' \to (\#, out)$ would yield the trap symbol #. In the second step, the target indication *out* has to be chosen and the rules $l_i'' \to (l_j, out)$, $d \to (\lambda, out)$, and $a_r' \to (a_r, out)$ are to be applied; if instead the target indication *here* were chosen again, the forced application of the rule $l_i'' \to \#$ would yield the trap symbol #.

Whenever a trap symbol is generated in one of the inner membranes, we get an infinite computation, as in R_1 we have the rule $\# \to (\#, in)$ and in every inner membrane we have the rule $\# \to (\#, out)$.

We finally observe that a computation in Π halts if and only if the final label l_h appears (and then stays in the skin membrane) and no trap symbol # is present, hence, we conclude $Ps(M) = Ps(\Pi)$. □

To eventually reduce the number of inner membranes remains as a challenging task for future research.

5 Computational Completeness of Time-Varying P Systems

Theorem 3. *$NOP_1(cat_1, \alpha TV_6) = NRE$, $Y \in \{N, Ps\}$, $\alpha \in \{\lambda, w\}$.*

Proof. We only prove the inclusion $PsRE \subseteq PsOP_1(cat_1, TV_6)$. 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 $A = \{a_1, \dots, a_{n+2}\}$ and divide the set of labels $B \setminus \{l_h\}$ into three disjoint subsets:

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

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):

$$\Pi = (O, \{c\}, [\ ]_1, l_0, R_1 \cup \cdots \cup R_6, R_1 \cup \cdots \cup R_6, (R_1 \ldots R_6)^*, 0),$$
$$O = A \cup \{a_1', a_2'\} \cup B' \cup \{c, h, \#\}.$$

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 step, the ADD instructions are simulated, i.e., for each $l_i : (\mathtt{ADD}(r), l_j, l_k) \in I$ we take

$cl_i \to ca_r\tilde{l}_j$, $cl_i \to ca_r\tilde{l}_k$ (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); in order to obtain the output in the environment, for $r \geq 3$, a_r has to be replaced by (a_r, out);

$cl \to cl^-$, $cl \to cl^0$ initiate the simulation of a SUB instruction for register 1 labeled by $l \in B_{-1}$;

$cl \to c\hat{l}$ 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 needed for simulating SUB instructions and eventually generated two steps before.

$\mathbf{R}_2$: 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) and

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

$\# \to \#$;

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B_+$,

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

$\mathbf{R}_3$: 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);

$\# \to \#$;

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B_+$;

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

$\mathbf{R}_4$: in the fourth step, the simulation of SUB instructions on register 2 is initiated, i.e., we take

$c\hat{l} \to cl^-$, $c\hat{l} \to cl^0$ for all $l \in B_{-2}$;

$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B_+ \cup B_{-1}$;

$\# \to \#$,

$h \to \lambda$.

$\mathbf{R}_5$: 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
$l^- \to \bar{l}^- h$, $l^- \to \bar{l}^0 h$ for all $l \in B_{-2}$;
$c\tilde{l} \to c\tilde{l}$, $\tilde{l} \to \#$ for all $l \in B_+ \cup B_{-1}$;
$\# \to \#$.

R$_6$: the simulation of SUB instructions $l_i : (\texttt{SUB}(2), l_j, l_k) \in I$ 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);

$c\tilde{l} \to cl$, $\tilde{l} \to \#$ for all $l \in B_+ \cup B_{-1}$.

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 possibilty 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 $Ps(M) = Ps(\Pi)$. □

6 Computational Completeness of P Systems with Membrane Creation

Theorem 4. *$YOP_{1,2}(cat_1, mcre) = YRE$, $Y \in \{N, Ps\}$.*

Proof. We only prove the inclusion $PsRE \subseteq PsOP_{1,2}(cat_1, mcre)$. 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 $A = \{a_1, \ldots, a_{n+2}\}$. We construct the following P system:

$$\begin{aligned} \Pi &= \left(O, \{c\}, [\ \ ]_1, cdl_0, R_1, R_2, R_3, 0\right), \\ O &= A \cup \{l, l', l'' \mid l \in B\} \cup \{c, d, d', d'', \#\}, \end{aligned}$$

and the sets of rules are constructed as follows.

A. For each ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ in I, the rules

$$\begin{aligned} &\text{step 1: } l_i \to l_i',\ d \to d', \\ &\text{step 2: } l_i' \to a_r l_j,\ l_i' \to a_r l_k,\ d' \to d. \end{aligned}$$

are introduced in R_1 and obviously simulate an ADD instruction in two steps.

B. For each SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ in I, the following rules are introduced in R_1 and R_{r+1}, $r \in \{1,2\}$:

Step	R_1	R_{r+1}
1	$cl_i \to c[\ l_i\ ]_{r+1},\ d \to d'$	–
2	$ca_r \to c\,(a_r, in_{r+1}),\ d' \to (d', in_{r+1})$	$l_i \to l_i'$
3	–	$a_r \to \lambda\delta,\ l_i' \to l_i'',\ d' \to d''$
4	$cl_i'' \to cl_j,\ d'' \to d$	$l_i'' \to l_k,\ d'' \to d\delta$

A SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (with $r \in \{1,2\}$) is simulated according to the four steps suggested in the table given above:

In the first step, we create a membrane with the label $r+1$, where l_i is sent to, and simultaneously d becomes d'. In the next step, if any a_r exists, i.e., if register r is not empty, then one copy of a_r should enter the membrane $r+1$ just having been created in the preceding step. Note that the selection of the membrane (the use of in_{r+1} instead of in) is important: a_r has to go to the membrane created in the previous step, when $r+1$ has been specified by the label l_i. At the same time, d' enters the membrane $r+1$, and l_i becomes l_i' in this membrane. If the register r is empty, then the catalyst is doing nothing in this second step.

In the third step, in membrane $r+1$, l_i' becomes l_i'' and d' becomes d''. If a_r is not present in membrane $r+1$, nothing else happens there in this step; if a_r is present, it dissolves the membrane and disappears. Observe that in both cases $ca_r \to c\,(a_r, in_{r+1})$ will not be applicable (anymore) in R_1. Thus, we either have $cl_i''d''$ in the skin membrane (when the register has been non-empty), or we have only c in the skin membrane and $l_i''d''$ in the inner membrane $r+1$. In the first case, in the fourth step we use the rules $cl_i'' \to cl_j$ and $d'' \to d$ from R_1, which is the correct continuation of the simulation of the SUB instruction; in the latter case, we use $l_i'' \to l_k$ and $d'' \to d\delta$ in R_{r+1}. The inner membrane is dissolved, and in the skin membrane we get the objects cl_kd. In both cases, the simulation of the SUB instruction is correct and we return to a configuration as that we started with, hence the simulation of another instruction can start.

C. We also add the rules $a_r \to (a_r, out)$ for $3 \leq r \leq n+2$ and $\# \to \#$ to R_1.

In any moment, any copy of a terminal symbol a_r in the skin membrane is sent out to the environment. Once the trap symbol # has been introduced, the computation continues forever.

There is one interference between the rules of Π simulating the ADD and the SUB instructions of M. If in the second step of simulating a SUB instruction, instead of $d' \to (d', in_{r+1})$ we use $d' \to d$, then the case when register r is non-empty continues correctly, as the simulation lasts four steps, and in the end d is present in the skin membrane (the dissolution of membrane r is done by a_r). If the register r has been empty, l_i'' will become l_k in membrane $r+1$ and it will remain there until d' enters the membrane, changes to d'', and then dissolves it (as long as d, d' switch to each other in the skin membrane, the computation cannot halt). Thus, also in this case we return to the correct submultiset $cl_k d$ in the skin membrane.

Consequently, exactly the halting computations of M are simulated by the halting computations in Π; hence, $Ps(M) = Ps(\Pi)$. The observation that the maximal number of membranes in any computation of Π is two completes the proof. □

It remains as an open problem whether it is possible to use the target indication in only instead of the in_j.

7 Computational Completeness of P Systems with Mobile Catalysts

If the membrane creation rules are of the form $ca \to [\ cb\ ]_i$, then this implicitly means that the catalyst is moving from one region to another one. However, for mobile catalysts, the universality of such systems with only one catalyst has already been proved in [5], using three membranes and target indications of the forms *here*, *out*, and in_j. In this paper, we improve this result from the last point of view, making only use of the target indications *here*, *out*, and *in*. In fact, if in the proof of Theorem 2 we let the catalyst c move with all the other objects, then we immediately obtain a proof for $NOP_7(mcat_1) = NRE$ where even only the target indications *out* and *in* are used (but instead of three we need seven membranes).

Theorem 5. $YOP_3(mcat_1) = YRE$, $Y \in \{N, Ps\}$.

Proof. We only prove the inclusion $PsRE \subseteq PsOP_1(cat_1, ls)$. 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 $A = \{a_1, \dots, a_{n+2}\}$ be the set of objects for representing the contents of the registers 1 to $n+2$ of M. We construct the following P system:

$$\begin{aligned}\Pi &= (O, \{c\}, [\ [\ \ ]_2 [\ \ ]_3\ ]_1, cl_0, R_1, R_2, R_3, 0),\\ O &= A \cup \{l, l', l'', l''' \mid l \in B\} \cup \{c, \#\},\end{aligned}$$

and the sets of rules are constructed as follows:

A. Let $l_i : (\mathtt{ADD}(r), l_j, l_k)$ be an ADD instruction in I. If $r = 3$, then the rules $l_i \to l_j(a_3, out)$, $l_i \to l_k(a_3, out)$ are introduced in R_1; if $r \in \{1,2\}$, in R_1 we introduce the rules $l_i \to l_j(a_r, in)$, $l_i \to l_k(a_r, in)$, as well as the rules $a_{4-j} \to \#$ and $\# \to \#$ in R_{j+1}, $j \in \{1,2\}$. The contents of each register r, $r \in \{1,2\}$, is represented by the number of objects a_r present in membrane $r+1$; any object a_r, $3 \le r \le n+2$, is immediately sent out into the environment. If a_{4-j} is introduced in membrane j, $j \in \{1,2\}$, then the trap object $\#$ is produced and the computation never halts.

B. The simulation of a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ is carried out by the following rules (the simulation again has four steps, as in the proof of Theorem 4):

For the first step, we introduce the rule $cl_i \rightarrow (c, in)(l_i, in)$ in R_1 and the rule $l_i \rightarrow \#$ in both R_2 and R_3 (if c and l_i are not moved together into an inner membrane, then the trap object $\#$ is produced and the computation never halts).

In the second step, R_{r+1} has to use the rule $cl_i \rightarrow cl_i'$. This checks whether c and l_i have been moved together into the right membrane $r+1$; if this is not the case, then the rule $cl_i \rightarrow cl_i'$ is not available and the rule $l_i \rightarrow \#$ must be used, which causes the computation to never halt.

Thus, after the second step, we know whether both c and l_i (l_i') are in the correct membrane $r+1$. The rules $ca_r \rightarrow (c, out)$ and $l_i' \rightarrow l_i''$are introduced in R_{r+1} in order to perform the third step of the simulation. If there is any copy of a_r in membrane $r+1$ (i.e., if register r is not empty), then the catalyst exits, while also removing a copy of a_r. Simultaneously, l_i' becomes l_i''. Hence, if the register r has been non-empty, we now have c in the skin membrane and l_i'' in membrane $r+1$; if register r has been empty, we have both c and l_i'' in membrane $r+1$. We introduce the rules $cl_i'' \rightarrow (c, out)(l_k, out)$, $l_i'' \rightarrow (l_i''', out)$, in R_{r+1} and the rules $cl_i''' \rightarrow cl_j$, $l_i''' \rightarrow \#$, $\# \rightarrow \#$ in R_1. If c is inside membrane $r+1$, we get cl_k in the skin membrane, which is the correct continuation for the case when the register is empty. If c is not in membrane $r+1$, then l_i'' exits alone thereby becoming l_i''', and, together with c, which waits in the skin membrane, introduces l_j, which is a correct continuation, too. If the rule $l_i'' \rightarrow (l_i''', out)$ is used although c is inside membrane $r+1$, then in the skin membrane we have to use the rule $l_i''' \rightarrow \#$ and the computation never halts.

In all cases, the simulation of the SUB instruction works correctly, and we return to a configuration with the catalyst and a label from H in the skin region.

In sum, we have the equality $Ps(M) = Ps(\Pi)$, which completes the proof. $\square$

8 Final Remarks

Although we have exhibited several new universality results for P systems using only one catalyst together with some additional control mechanism, the orignal problem of characterizing the sets of non-negative integers generated by P systems with only one catalyst still remains open. A similar challenging problem is to consider *purely catalytic* P systems with only two catalysts: with only one catalyst, we obtain the regular sets; as shown in [3], three catalysts are enough to obtain universality. With two catalysts and some additional control mechanism, universality can be obtained, too; for example, the proof of Theorem 1 for P systems with label selection for the rules can easily be adapted for purely catalytic P systems, i.e., $NOP_1(pcat_2, ls) = NRE$. For the other variants of additional control mechanisms, the case of purely catalytic P systems with two catalysts remains for future research.

Acknowledgements. The work of Gheorghe Păun has been supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200, co-financed by FEDER funds.

References

1. A. Alhazov, R. Freund, H. Heikenwälder, M. Oswald, Yu. Rogozhin, S. Verlan, Sequential P systems with regular control. In: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, G. Vaszil (Eds.): *Membrane Computing - 13th International Conference, CMC 2012*, Budapest, Hungary, August 28-31, 2012, Revised Selected Papers, LNCS 7762, Springer, 2013, 112–127.
2. Jürgen Dassow, Gheorghe Păun: *Regulated Rewriting in Formal Language Theory.* Springer, 1989.
3. Rudolf Freund, Lila Kari, Marion Oswald, Petr Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330**, 2005, 251–266.
4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: P systems with membrane creation and rule input, to appear.
5. S.N. Krishna, A. Păun: Results on catalytic and ecolution-communication P systems. *New Generation Computing*, 22 (2004), 377–394.
6. K. Krithivasan, Gh. Păun, A. Ramanujan: On controlled P systems. *Fundamenta Informaticae*, to appear.
7. Marvin L. Minsky: *Computation: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
8. M. Mutyam, K. Krithivasan: P systems with membrane creation: universality and efficiency. In: M. Margenstern, Y. Rogozhin (Eds.): *Proc. MCU 2001*, LNCS 2055, Springer, 2001, 276–287.
9. Gh. Păun: Computing with membranes. *J. Comput. Syst. Sci.*, 61 (2000), 108–143 (see also TUCS Report 208, November 1998, `www.tucs.fi`).
10. Gh. Păun: Computing with membranes - a variant. *Intern. J. Found. Computer Sci.*, 11, 1 (2000), 167–182.
11. Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.
12. Grzegorz Rozenberg, Arto Salomaa (Eds.): *Handbook of Formal Languages,* 3 volumes. Springer, 1997.
13. The P Systems Website: `http://ppage.psystems.eu`.