The power of Parallelism in Membrane Computing

Rudolf Freund

Faculty of Informatics, TU Wien
Favoritenstraße 9–11, 1040 Wien, Austria
rudi@emcc.at

Abstract: Maximal parallelism has been an important feature of membrane systems from the beginning, i.e., only non-extendable multisets of rules are applied to the underlying configuration. During the last two decades many new variants of parallel derivation modes have been investigated, for example, non-extendable sets of rules are used instead of non-extendable multisets of rules. In many cases, computational completeness can be obtained. Recently, derivation modes applying multisets of rules affecting or generating the maximal number of objects or yielding the maximal difference between the objects in the current and the derived configuration have been shown to allow for computational completeness even when using only very restricted variants of rules.

1 Introduction

When membrane systems were introduced in [37] more than two decades ago, the application of non-extendable multisets of rules was one of the basic features of this bio-inspired model of computing. An introduction to this fascinating area is documented in two textbooks, see [38] and [39]. For actual information see the P systems webpage [42] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

The basic model of membrane systems (P systems, see [37] and [38]) can be seen as a multiset rewriting system where objects evolve in parallel in all the regions of a hierarchical membrane structure, but the resulting objects may also pass through the membranes surrounding a membrane region. In every derivation step a non-extendable multiset of rules is applied. The result of a computation is extracted when the system halts, i.e., when no rule is applicable any more. The multiset rewriting rules often can be restricted to non-cooperative rules of the form $a \to v$ and catalytic rules of the form $ca \to cv$, where c is a catalyst, a is a single object and v is a multiset of objects. Catalysts are special objects which allow only one object to evolve in its context, but in their basic variant never evolve themselves. P systems using only these two types of rules are called *catalytic*, and if only catalytic rules are allowed we speak of *purely catalytic* P systems.

In the context of catalytic and purely catalytic P systems, the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges from the beginning. Without catalysts only regular (semi-linear) sets can be generated when using the standard maximally parallel derivation mode and the standard halting mode. For catalytic P systems with only one catalyst a lower bound was established in [32]: P systems with one catalyst can simulate partially blind register machines, i.e., they can generate more than just semi-linear sets. At least when using additional control mechanisms, even one catalyst can be sufficient to obtain computational completeness, see [28]: for example, in P systems with label selection, only rules from one set of a finite number of sets of rules in each computation step are used; in time-varying P systems, the available sets of rules change periodically with time. For many other variants of P systems using specific control mechanism for the application of rules the interested reader is referred to the list of references, for example, see [1, 2, 3, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 20, 21, 22, 23, 26, 27, 28, 29, 30, 31, 34, 35].

In [24] it was shown that without any additional ingredients like a priority relation on the rules as used in the original definition computational completeness can be obtained by showing that register machines with n registers can be simulated by (purely) catalytic P systems with $(n+3)\,n+2$ catalysts.

In [8], the idea of using a priority relation on the rules was revived, but in a very weak form: overall in the system, catalytic rules have weak priority over non-catalytic rules. Even without using more than this weak priority of catalytic rules over the non-catalytic (non-cooperative) rules, computational completeness could be established for catalytic P systems with only one catalyst. Moreover, starting from this result, an even stronger result has been established in [12], where computational completeness for catalytic P systems with only one catalyst is shown when the derivation mode $max_{objects}$ is used, i.e., only those multisets of rules are taken which affect the maximal number of objects in the underlying configuration.

In this paper, after recalling some classic results, I will focus on the research which has been started in [12] and then has been continued in [11] and in [13]. I will consider several variants of derivation modes based on non-extendable multisets of rules and taking only those for which (i) the number of affected objects is maximal, (ii) the number of objects generated by the application of the

multiset of rules is maximal, or (iii) the difference of objects between the underlying configuration and the configuration after the application of the multiset of rules is maximal. In case of catalytic P systems, for all these variants one can also take such multisets of rules without requesting them to fulfill the condition to be non-extendable.

2 Definitions

The set of natural numbers $n \geq 0$ is denoted by $\mathbb{N}$. For any two natural numbers m, n, $m \leq n$, $[m..n]$ denotes the set of natural numbers $\{k \mid m \leq k \leq n\}$. Moreover, $m \oplus_n +1$ is defined as $m+1$ for $1 \leq m < n$ and $n \oplus_n +1 = 1$.

For an alphabet V, the free monoid generated by V under the operation of concatenation, i.e., the set containing all possible strings over V is denoted by V^*. The *empty string* is denoted by λ. A *multiset* M with underlying set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $M = (A, f)$ is a multiset then its *support* is defined as $supp(M) = \{x \in A \mid f(x) > 0\}$. A multiset is empty (respectively finite) if its support is the empty set (respectively a finite set). If $M = (A, f)$ is a finite multiset over A and $supp(M) = \{a_1, \ldots, a_k\}$, then it can also be represented by the string $a_1^{f(a_1)} \ldots a_k^{f(a_k)}$ over the alphabet $\{a_1, \ldots, a_k\}$, and, moreover, all permutations of this string precisely identify the same multiset M. The set of all multisets over V is denoted by V°. The cardinality of a set or multiset M is denoted by $|M|$.

For further notions and results in formal language theory I refer to textbooks like [19] and [40].

2.1 Register Machines

Register machines are well-known universal devices for computing on (or generating or accepting) sets of vectors of natural numbers. The following definitions and propositions are given as in [12].

Definition 1. *A* register machine *is a construct*

$$M = (m, B, l_0, l_h, P)$$

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:

- *$p : (ADD(r), q, s)$; $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  Increase the value of register r by one, and nondeterministically jump to instruction q or s.
- *$p : (SUB(r), q, s)$; $p \in B \setminus \{l_h\}$, $q, s \in B$, $1 \leq r \leq m$.*
  *If the value of register r is not zero then decrease the value of register r by one (*decrement *case) and jump to instruction q, otherwise jump to instruction s (*zero-test *case).*
- *$l_h : HALT$.*
  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. M is called* deterministic *if the ADD-instructions all are of the form $p : (ADD(r), q)$.*

Throughout the paper, B_{ADD} denotes the set of labels of ADD-instructions $p : (ADD(r), q, s)$ of arbitrary registers r, and $B_{SUB(r)}$ denotes the set of labels of all SUB-instructions $p : (SUB(r), q, s)$ of a decrementable register r. Moreover, for any $p \in B \setminus \{l_h\}$, $Reg(p)$ denotes the register affected by the ADD- or SUB-instruction labeled by p; for the sake of completeness, in addition $Reg(l_h) = 1$ is taken.

In the *accepting* case, a computation starts with the input of an l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction. Without loss of generality, we may assume all registers to be empty at the end of the computation.

In the *generating* case, a computation starts with all registers being empty and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

In the *computing* case, a computation starts with the input of an l-vector of natural numbers in its first l registers and by executing the first instruction of P (labeled with l_0); it terminates with reaching the $HALT$-instruction and the output of a k-vector of natural numbers in its last k registers. Without loss of generality, we may assume all registers except the last k output registers to be empty at the end of the computation.

For useful results on the computational power of register machines, we refer to [36]; for example, to prove our main theorem, we need the following formulation of results for register machines generating or accepting recursively enumerable sets of vectors of natural numbers with k components or computing partial recursive relations on vectors of natural numbers:

Proposition 1. *Deterministic register machines can accept any recursively enumerable set of vectors of natural numbers with l components using precisely $l+2$ registers. Without loss of generality, we may assume that at the end of an accepting computation all registers are empty.*

Proposition 2. *Register machines can generate any recursively enumerable set of vectors of natural numbers with k components using precisely $k+2$ registers. Without loss of generality, we may assume that at the end of a generating computation the first two registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

Proposition 3. *Register machines can compute any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no SUB-instruction is ever used.*

In all cases it is essential that the output registers never need to be decremented.

Remark 1. *For any register machine, without loss of generality we may assume that the first instruction is an ADD-instruction on register* 1*: in fact, given a register machine $M = (m,B,l_0,l_h,P)$ with having a another instruction as its first instruction, we can immediately construct an equivalent register machine M' which starts with an increment immediately followed by a decrement of the first register:*

$$\begin{aligned} M' &= \left(m,B',l_0',l_h,P'\right), \\ B' &= B \cup \{l_0',l_0''\}, \\ P' &= P \cup \{l_0' : (ADD(1),l_0'',l_0''),\ l_0'' : (SUB(1),l_0,l_0)\}. \end{aligned}$$

2.2 Simple P Systems

Taking into account the well-known flattening process, which means that computations in a P system with an arbitrary membrane structure can be simulated in a P system with only one membrane, e.g., see [25], in this paper we only consider simple (catalytic, purely catalytic) P systems, i.e., with the simplest membrane structure of only one membrane:

Definition 2. *A* simple P system *is a construct*

$$\Pi = (V,C,T,w,\mathscr{R})$$

where

- *V is the alphabet of objects;*
- *$C \subset V$ is the set of* catalysts*;*
- *$T \subseteq (V \setminus C)$ is the alphabet of* terminal *objects;*
- *$w \in V^\circ$ is the multiset of objects initially present in the membrane region;*
- *$\mathscr{R}$ is a finite set of evolution rules over V; these evolution rules are mutiset rewriting rules $u \to v$ with $u,v \in V^\circ$.*

The P system Π is called catalytic*, if $\mathscr{R}$ contains only non-cooperative rules of the form $a \to cv$ and catalytic rules of the form $ca \to cv$, where $c \in C$ is a catalyst, a is an object from $V \setminus C$, and v is a multiset over $V \setminus C$. The P system Π is called* purely catalytic*, if $\mathscr{R}$ contains only catalytic rules.*

The multiset in the single membrane region of Π constitutes a *configuration* of the P system. The *initial configuration* is given by the initial multiset w; in case of accepting or computing P systems the input multiset w_0 is assumed to be added to w, i.e., the initial configuration then is ww_0.

A transition between configurations is governed by the application of the evolution rules, which is done in a given derivation mode. The application of a rule $u \to v$ to a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v.

2.3 Variants of Derivation Modes

The definitions and the corresponding notions used in this subsection follow the definitions and notions elaborated in [33] as well as in [12] and [13].

Given a P system $\Pi = (V,C,T,w,\mathscr{R})$, the set of multisets of rules applicable to a configuration C is denoted by $Appl(\Pi,C)$; this set also equals the set $Appl(\Pi,C,asyn)$ of multisets of rules applicable in the *asynchronous derivation mode* (abbreviated *asyn*).

Given a multiset R of rules in $Appl(\Pi,C)$, we write $C \stackrel{R}{\to} C'$ if C' is the result of applying R to C. The number of objects affected by applying R to C is denoted by $\mathrm{Aff}(C,R)$. The number of objects generated in C' by the right-hand sides of the rules applied to C with the multiset of rules R is denoted by $\mathrm{Gen}(C,R)$. The difference between the number of objects in C' and C is denoted by $\Delta obj(C,R)$.

The set $Appl(\Pi,C,sequ)$ denotes the set of multisets of rules applicable in the *sequential derivation mode* (abbreviated *sequ*), where in each derivation step exactly one rule is applied.

The standard parallel derivation mode used in P systems is the *maximally parallel derivation mode* (*max* for short). In the maximally parallel derivation mode, in any computation step of Π we choose a multiset of rules from $\mathscr{R}$ 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 configuration, i.e., in simple P systems we only take applicable multisets of rules which cannot be extended by further (copies of) rules and are to be applied to the objects in the single membrane region:

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

We first consider the derivation mode $max_{objects}max$ where from the multisets of rules in $Appl(\Pi,C,max)$ only those are taken which affect the maximal number of objects. As with affecting the maximal number of objects, such multisets of rules are non-extendable anyway, we will also use the notation $max_{objects}$. Formally we may write:

$$Appl(\Pi,C,max_{objects}max) = \{R \in Appl(\Pi,C,max) \mid \text{there is no } R' \in Appl(\Pi,C,max) \text{ such that } \mathrm{Aff}(C,R) < \mathrm{Aff}(C,R')\}$$

and

$$Appl(\Pi,C,max_{objects}) = \{R \in Appl(\Pi,C,asyn) \mid \text{there is no } R' \in Appl(\Pi,C,asyn) \text{ such that } \mathrm{Aff}(C,R) < \mathrm{Aff}(C,R')\}.$$

As already mentioned, both definitions yield the same multiset of rules.

In addition to these well-known derivation modes, in this paper we also consider several new variants of derivation modes as already introduced in [11], where instead of looking at the number of affected objects we take into account the number of generated objects and the difference of objects between the derived configuration and the current configuration, respectively.

$max_{GENobjects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the number of objects generated by the application of the rules in R to the configuration C is maximal with respect to the number of objects generated by the application of the rules in any other non-extendable multiset of rules R' to the configuration C:

$$Appl(\Pi,C,max_{GENobjects}max) = \{R \in Appl(\Pi,C,max) \mid \text{there is no } R' \in Appl(\Pi,C,max) \text{ such that } \mathrm{Gen}(C,R) < \mathrm{Gen}(C,R')\}.$$

$max_{\Delta objects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the difference $\Delta C = |C'| - |C|$ between the number of objects in the configuration C' obtained by the application of R and the number of objects in the underlying configuration C is maximal with respect to the differences in the number of objects obtained by applying any other non-extendable multiset of rules:

$$Appl(\Pi,C,max_{\Delta objects}max) = \{R \in Appl(\Pi,C,max) \mid \text{there is no } R' \in Appl(\Pi,C,max) \text{ such that } \Delta obj(C,R) < \Delta obj(C,R')\}.$$

Like for $max_{objects}max$ in comparison with $max_{objects}$ we now can also consider the variants of the other maximal derivation modes where we do not start with imposing the restriction of being non-extendable on the applicable multisets:

$max_{GENobjects}$ a multiset of rules R applicable to the current configuration C is only taken if the number of objects generated by the application of the rules in R to the configuration C is maximal with respect to the number of objects generated by the application of the rules in any other multiset of rules R' to the configuration C:

$$Appl(\Pi,C,max_{GENobjects}) = \{R \in Appl(\Pi,C,asyn) \mid \text{there is no } R' \in Appl(\Pi,C,asyn) \text{ such that } \mathrm{Gen}(C,R) < \mathrm{Gen}(C,R')\}.$$

$max_{\Delta objects}$ a multiset of rules R applicable to the current configuration C is only taken if the difference $\Delta C = |C'| - |C|$ between the number of objects in the configuration C' obtained by the application of R and the number of objects in the underlying configuration C is maximal with respect to the differences in the number of objects obtained by applying any other multiset of rules:

$$Appl(\Pi,C,max_{\Delta objects}) = \{R \in Appl(\Pi,C,asyn) \mid \text{there is no } R' \in Appl(\Pi,C,asyn) \text{ such that } \Delta obj(C,R) < \Delta obj(C,R')\}.$$

We illustrate the difference between these new derivation modes in the following examples, thereby emphasizing on catalytic and non-cooperative rules:

Example 1. *To illlustrate the derivation modes $max_{GENobjects}max$ as well as $max_{\Delta objects}max$, consider a simple P system with the initial configuration caa and the set of rules $\{a \to b, ca \to cd\}$.*

In case of the derivation mode $max_{GENobjects}max$, only the multiset of rules $\{ca \to cd, a \to b\}$ can be applied, as $Gen(caa,\{ca \to cd\}) = 2$ and $Gen(cab,\{a \to b\}) = 1$ and therefore $Gen(caa,\{ca \to cd, a \to b\}) = 3$, whereas $Gen(caa,\{a \to b, a \to b\}) = 2$. Hence, the only possible derivation with the derivation mode $max_{GENobjects}max$ is $caa \xrightarrow{\{ca\to cd, a\to b\}} cdb$. In this special case,

$$Appl(\Pi,caa,max_{GENobjects}max) = Appl(\Pi,caa,max_{objects}).$$

On the other hand, with the derivation mode $max_{\Delta objects}max$ both rules yield the same difference of 0, i.e., $\Delta obj(caa,\{ca \to cd\}) = \Delta obj(caa,\{a \to b\}) = 0$, which yields all two non-extendable multisets of rules $\{ca \to cd, a \to b\}$ and $\{a \to b, a \to b\}$ to be applicable to the underlying configuration caa, i.e.,

$$Appl(\Pi,caa,max_{\Delta objects}max) = Appl(\Pi,caa,max).$$

Now let us take the set of rules $\{a \to bb, ca \to cd\}$. Observing that $Gen(caa,\{a \to bb\}) = 2$ and $\Delta obj(caa,\{a \to bb\}) = 1$, we obtain the following sets of applicable multisets of rules:

$Appl(\Pi, caa, max_{GENobjects}max) =$
$\{\{a \to bb, a \to bb\}, \{a \to bb, ca \to cd\}\}$,
$Appl(\Pi, caa, max_{\Delta objects}max) = \{\{a \to bb, a \to bb\}\}$.

Finally, let us take the set of rules $\{a \to \lambda, ca \to cd\}$. *As* $Gen(caa, \{a \to \lambda\}) = 0$ *and* $\Delta obj(caa, \{a \to \lambda\}) = -1$, *we obtain the following sets of applicable multisets of rules:*
$Appl(\Pi, caa, max_{GENobjects}max) =$
$Appl(\Pi, caa, max_{\Delta objects}max) = \{\{a \to \lambda, ca \to cd\}\}$.

Example 2. *Consider a simple purely catalytic P system with the initial configuration* c_1c_2aa *and the following rules:*

1. $c_1a \to c_1$
2. $c_2a \to c_2b$
3. $c_2a \to c_2bb$

We immediately observe the following:

1. $Gen(c_1c_2aa, \{c_1a \to c_1\}) = 1$,
2. $Gen(c_1c_2aa, \{c_2a \to c_2b\}) = 2$,
3. $Gen(c_1c_2aa, \{c_2a \to c_2bb\}) = 3$.

In case of the derivation mode $max_{GENobjects}max$, *the multiset of rules* $\{c_1a \to c_1, c_2a \to c_2bb\}$ *has to be applied. Hence, the only possible derivation with the derivation mode* $max_{GENobjects}max$ *is* $c_1c_2aa \xrightarrow{\{c_1a \to c_1, c_2a \to c_2bb\}} c_1c_2bb$. *In this special case,*
$Appl(\Pi, c_1c_2aa, max_{GENobjects}max) =$
$Appl(\Pi, c_1c_2aa, max_{objects})$.

If we do not start from non-extendable multisets of rules, we obtain the same results, i.e., in the derivation mode $max_{GENobjects}$, *the multiset of rules* $\{c_1a \to c_1, c_2a \to c_2bb\}$ *has to be applied, and the only possible derivation with the derivation mode* $max_{GENobjects}$ *is* $c_1c_2aa \xrightarrow{\{c_1a \to c_1, c_2a \to c_2bb\}} c_1c_2bb$.

In the same way, for the difference of generated and consumed objects we obtain:

1. $\Delta obj(c_1c_2aa, \{c_1a \to c_1\}) = -1$,
2. $\Delta obj(c_1c_2aa, \{c_2a \to c_2b\}) = 0$,
3. $\Delta obj(c_1c_2aa, \{c_2a \to c_2bb\}) = 1$.

As for the derivation mode $max_{\Delta objects}max$, *also for the derivation mode* $max_{GENobjects}max$ *we obtain that the multiset of rules* $\{c_1a \to c_1, c_2a \to c_2bb\}$ *has to be applied and that the only possible derivation with the derivation mode* $max_{\Delta objects}max$ *is* $c_1c_2aa \xrightarrow{\{c_1a \to c_1, c_2a \to c_2bb\}} c_1c_2bb$.

On the other hand, if we do not start from non-extendable multisets of rules, now the rule $c_1a \to c_1$ *must not be applied because it would decrease the number of objects, i.e., in the derivation mode* $max_{\Delta objects}$ *we obtain that the* – not *non-extendable* – *multiset of rules* $\{c_2a \to c_2bb\}$ *has to be applied, and the only possible derivation with the derivation mode* $max_{\Delta objects}$ *is* $c_1c_2aa \xrightarrow{\{c_2a \to c_2bb\}} c_1c_2abb$.

3 Some Classic Results

In this section I recall some classic results for P systems being computationally complete:

Theorem 4. *Every (computation of a) register machine M can be simulated by a simple P system* Π. *If M is deterministic, then the simulation in* Π *is deterministic, too.*

Proof. Let $M = (m, B, l_0, l_h, P)$ be an arbitrary register machine with the first n registers being the decrementable ones and the registers $n+1, \ldots, m$ being the output registers.

Then we construct an equivalent simple P system $\Pi = (V, T, l_0, \mathscr{R})$ as follows (as we do not use catalysts, we omit C in the definition of Π):

$$\begin{array}{rcl} V & = & \{a_r \mid 1 \le r \le m\} \cup B \\ & \cup & \{p, p', p'', \tilde{p}, \bar{p} \mid p : (SUB(r), q, s) \in P\}, \\ T & = & \{a_r \mid n+1 \le r \le m\}, \\ \mathscr{R} & = & \{p \to a_r q, p \to a_r s \mid p : (ADD(r), q, s) \in P\} \\ & \cup & \{p \to p'p'', p' \to \tilde{p}, p''a_r \to \bar{p}, \\ & & \tilde{p}\bar{p} \to q, \tilde{p}p'' \to s \mid \\ & & p : (SUB(r), q, s) \in P\} \cup \{l_h \to \lambda\}. \end{array}$$

The contents of a register r is represented by the corresponding number of copies of the symbol a_r.

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

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

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

The HALT-instruction $l_h : HALT$ is simulated by the rule $l_h \to \lambda$. □

Theorem 5. *(see [5]) Every (computation of a) register machine M withat least two decrementable registers can be simulated by a simple catalytic P system* Π *and a simple purely catalytic P system* Π'.

Proof. Let $M = (m, B, l_0, l_h, P)$ be an arbitrary register machine with the first n registers being the decrementable ones and the registers $n+1, \ldots, m$ being the output registers. According to Remark 1, the first instruction is assumed to be an *ADD*-instruction on register 1. Moreover, without loss of generality we may assume that to output registers no *SUB*-instructions are applied and that at the end of a halting computation all registers which allow for *SUB*-instructions are empty.

As in the original construction given in [24], n catalysts are used, each of these catalysts being needed

for decrementing one of the registers allowing for *SUB*-instructions; moreover, the idea of "paired catalysts" is taken over, i.e., the catalyst c_r works together with the catalyst $c_{r\oplus_n 1}$ (this is the reason why we need at least two decrementable registers). The contents of a register r is represented by the corresponding number of copies of the symbol a_r.

The following abbreviations for specific multisets (written as strings) are used:

$$\begin{aligned} D_{n,r} &= \Pi_{j\in[1..n]\setminus\{r\}} d_j \text{ and} \\ D'_{n,r} &= \Pi_{j\in[1..n]\setminus\{r,r\oplus_n 1\}} d_j. \end{aligned}$$

The equivalent simple catalytic P system

$$\Pi = (V, C, T, l_0 D_{n,1}, \mathscr{R})$$

(observe that we start with an *ADD*-instruction on register 1) now is constructed as follows:

$$\begin{aligned} V &= \{a_r \mid 1 \le r \le m\} \cup B \cup C \cup D \\ &\cup \{p, p', \tilde{p}, \bar{p} \mid p : (SUB(r), q, s) \in P\}, \\ C &= \{c_r \mid 1 \le r \le n\}, \\ D &= \{d_r \mid 1 \le r \le n\}, \\ T &= \{a_r \mid n+1 \le r \le m\}, \\ \mathscr{R} &= \{c_r d_r \to c_r \mid 1 \le r \le n\} \cup \{c_1 l_h \to c_1\} \\ &\cup \{c_r p \to c_r a_r q D_{n,Reg(q)}, c_r p \to c_r a_r s D_{n,Reg(s)} \\ &\quad \mid p : (ADD(r), q, s) \in P\} \\ &\cup \{c_r d_r \to c_r \mid 1 \le r \le n\} \\ &\cup \{c_r p \to c_r \bar{p} D_{n,r},\ c_r p \to c_r p' D'_{n,r}, \\ &\quad c_r a_r \to c_r a'_r D'_{n,r},\ c_r \bar{p} \to c_r \#, \\ &\quad c_{r\oplus_n 1} p' \to c_{r\oplus_n 1} s D_{n,Reg(s)}, \\ &\quad c_r a'_r \to c_r d_{r\oplus_n 1}, c_{r\oplus_n 1} \bar{p} \to c_{r\oplus_n 1} \tilde{p} D'_{n,r}, \\ &\quad c_r \tilde{p} \to c_r q D_{n,Reg(q)} \\ &\quad \mid p : (SUB(r), q, s) \in P\} \\ &\cup \{x \to \# \mid x \in \{\#\} \cup \{d_j, a'_j \mid 1 \le j \le n\} \\ &\quad \cup \{p, p', \tilde{p} \mid p \in B_{SUB(r)}, 1 \le r \le n\}\}. \end{aligned}$$

For each catalyst c_r we use a "dummy" symbol d_r, which keeps the catalyst c_r busy whenever needed enforcing c_r to use the rule $c_r d_r \to c_r$ to keep the simulation alive. When one of the instructions works on a specific register r, then only the catalyst c_r and in case of *SUB*-instructions also catalyst $c_{r\oplus_k 1}$ is involved, whereas the catalysts corresponding to the other registers j have to be kept busy by the corresponding rule $c_j d_j \to c_j$. For each *SUB*-instruction labeled by the "program" symbol p also the variants $p', \tilde{p}, \bar{p}$ are used.

The HALT-instruction $l_h : HALT$ is simulated by the rule $c_1 l_h \to c_1$ (observe that $Reg(l_h) = 1$ is assumed).

Each *ADD*-instruction $j : (ADD(r), k, l)$ is simulated by the two rules $c_r p \to c_r a_r x D_{k,Reg(x)}$, $x \in \{q, s\}$.

Each *SUB*-instruction $j : (SUB(r), k, l)$ is simulated in at most four steps as shown in the table given below:

Simulation of the *SUB*-instruction $p : (SUB(r), q, s$ if register r is not empty	register r is empty
$c_r p \to c_r \bar{p} D_{n,r}$	$c_r p \to c_r p' D'_{n,r}$
$c_{r\oplus_n 1} d_{r\oplus_n 1} \to c_{r\oplus_n 1}$	$c_{r\oplus_n 1} d_{r\oplus_n 1} \to c_{r\oplus_n 1}$
$c_r a_r \to c_r a'_r D'_{n,r}$	c_r remains idle
$c_{r\oplus_n 1} d_{r\oplus_n 1} \to c_{r\oplus_n 1}$	$c_{r\oplus_n 1} p' \to c_{r\oplus_n 1} s D_{n,Reg(s)}$
$c_r a'_r \to c_r d_{r\oplus_n 1}$	
$c_{r\oplus_n 1} \bar{p} \to c_{r\oplus_n 1} \tilde{p} D'_{n,r}$	
$c_r \tilde{p} \to c_r q D_{n,Reg(q)}$	
$c_{r\oplus_n 1} d_{r\oplus_n 1} \to c_{r\oplus_n 1}$	

The trap rules $x \to \#$ guarantee that all the symbols x are used in a correct way in the rules listed above for the simulation of the register machine instructions. As soon as the trap symbol # has been introduced, the derivation finally will enter an infinite loop with the rule $\# \to \#$. In the case of catalytic P systems, the only non-cooperative rules are these trap rules.

In case the assumption about register r being not empty is wrong, then instead of the rule $c_r a_r \to c_r a'_r D'_{n,r}$ the trap rule $c_r \bar{p} \to c_r \#$ must be used. On the other hand, in case the assumption about register r being empty is wrong, then catalyst c_r will not stay idle, but will be used with the rule $c_r a_r \to c_r a'_r D'_{n,r}$ instead. Yet then in the third step in sum $2n-1$ objects to be handled by only n catalysts will be present in the configuration, which is impossible and therefore will lead to the introduction of the trap symbol #.

In the purely catalytic case, one additional catalyst c_{d+1} is needed for all the non-cooperative rules given above. These trap rules, and only those, are associated with this catalyst c_{d+1}; for example, the trap rule $\# \to \#$ now is replaced by the rule $c_{d+1}\# \to c_{d+1}\#$. □

The construction given in the proof above works for both catalytic and purely catalytic P systems. An improved version for catalytic P systems with respect to the number of rules needed for simulating *SUB*-instructions was presented at the Workshop on Membrane Computing 2015 (Satellite Workshop of UCNC 2015 in Auckland), see [4]. As this improved result for catalytic P systems is the best known so far with respect to the number of rules needed for simulating *SUB*-instructions, this specific construction is recalled in the proof of the following theorem:

Theorem 6. *For any register machine $M = (m, B, l_0, l_h, P)$, with $n \le m$ being the number of decrementable registers, we can construct a simple catalytic P system $\Pi = (V, C, T, w, \mathscr{R})$ simulating the computations of M such that*

$$\begin{aligned} |\mathscr{R}| &\le ADD^1(P) + 2 \times ADD^2(P) + \\ &\quad 5 \times SUB(P) + 5 \times m + 1, \end{aligned}$$

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

Proof. Again a register machine $M = (m, B, l_0, l_h, P)$ with $n \leq m$ decrementable registers is simulated by a catalytic P system $\Pi = (V, C, T, w, \mathscr{R})$.

For each of the n decrementable registers, we take a catalyst c_r and two specific symbols d_r, e_r, $1 \leq r \leq n$, for simulating *SUB*-instructions on these registers.

$$\begin{array}{rcl}
V & = & C \cup D \cup E \cup \Sigma \cup \{\#\} \cup \{p \mid p \in B\} \\
 & \cup & \{p', \bar{p}, \tilde{p} \mid l \in B_{SUB}\}, \\
C & = & \{c_r \mid 1 \leq r \leq n\}, \\
D & = & \{d_r \mid 1 \leq r \leq n\}, \\
E & = & \{e_r \mid 1 \leq r \leq n\}, \\
\Sigma & = & \{a_r \mid 1 \leq r \leq m\}, \\
T & = & \{a_r \mid 1 \leq r \leq n\}, \\
\mathscr{R} & = & \{l_h \to \lambda\} \\
 & \cup & \{p \to a_r q D_n, q \to a_r s D_n \\
 & & \mid p : (ADD(r), q, s) \in P\} \\
 & \cup & \{p \to \bar{p} e_r D_{n,r}, p \to p' D_{n,r}, \\
 & & \bar{p} \to \tilde{p} D'_{n,r}, p' \to s D_n, \tilde{p} \to q D_n \\
 & & \mid p : (SUB(r), q, s) \in P\} \\
 & \cup & \{c_r a_r \to c_r d_r, c_r d_r \to c_r, c_{r \oplus_n 1} e_r \to c_{r \oplus_n 1} \\
 & & \mid 1 \leq r \leq n\}, \\
 & \cup & \{d_r \to \#, c_r e_r \to c_r \# \mid 1 \leq r \leq n\} \\
 & \cup & \{\# \to \#\}.
\end{array}$$

The initial configuration is

$$w = c_1 \ldots c_n d_1 \ldots d_n p_1 w_0$$

where w_0 stands for additional input present at the beginning, for example, for the given input in case of accepting systems.

Usually, every catalyst c_r, $r \in [1..n]$, is kept busy with the symbol d_r using the rule $c_r d_r \to c_r$, as otherwise the symbols d_r would have to be trapped by the rule $d_r \to \#$, and the trap rule $\# \to \#$ then enforces an infinite non-halting computation. 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 $c_{r \oplus_n 1}$ are needed to be free for specific actions in the third step of the simulation.

For the simulation of instructions, we use the following shortcuts:

$$\begin{array}{rcl}
D_n & = & \prod_{i \in [1..n]} d_i, \\
D_{n,r} & = & \prod_{i \in [1..n] \setminus \{r\}} d_i, \\
D'_{n,r} & = & \prod_{i \in [1..n] \setminus \{r, r \oplus_n 1\}} d_i.
\end{array}$$

Each *ADD*-instruction $p : (ADD(r), q, s)$, for $r \in [1..n]$, is simulated by the rules $p \to a_r q D_n$ and $p \to a_r s D_m$; in parallel, the rules $c_r d_r \to c_r$, $1 \leq r \leq n$, have to be carried out, as otherwise the symbols d_r would have to be trapped by the rules $d_r \to \#$.

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

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

In the first step of the simulation of each instruction (*ADD*-instruction, *SUB*-instruction, and even HALT-instruction) due to the introduction of D_n 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, $1 \leq r \leq m$. Hence, this also guarantees that the zero-check on register r works correctly enforcing $d_r \to \#$ to be applied, as in the case of a wrong choice two symbols d_r are present.

The HALT-instruction $l_h : HALT$ is simulated by the rule $l_h \to \lambda$; observe that no objects a_r for $1 \leq r \leq n$ are present any more when l_h has appeared. □

Remark 2. *Exactly the same construction as elaborated above can be used when allowing for $n+2$ catalysts, with catalyst c_{n+1} being used with the state symbols and catalyst c_{n+2} being used with the trap rules. If only one additional catalyst c_{n+1} is allowed to be used with all the non-cooperative rules, a slightly more complicated simulation of SUB-instructions is needed, see [41], where for catalytic P systems*

$$\begin{array}{rcl}
|\mathscr{R}| & \leq & 2 \times ADD^1(P) + 3 \times ADD^2(P) + \\
 & & 6 \times SUB(P) + 5 \times m + 1,
\end{array}$$

and for purely for catalytic P systems

$$\begin{array}{rcl}
|\mathscr{R}| & \leq & 2 \times ADD^1(P) + 3 \times ADD^2(P) + \\
 & & 6 \times SUB(P) + 6 \times m + 1,
\end{array}$$

was shown.

Remark 3. *In case of deterministic register machines, especially in the accepting case, every sequence of consecutive ADD-instructions is bounded by a constant only depending on the given register machine. Hence, in this case the calculations for $|\mathscr{R}|$ in Theorem 6 and in Remark 2 can omit the ADD-instructions, because any fixed sequence of consecutive ADD-instructions can be included directly in the last simulation step of the preceding SUB-instruction, see the concept of generalized register machines as used in [17] and also cited in [41].*

Remark 4. *The use of the trap symbol # to enforce the computation never to stop is a typical element of many proofs to be found in the literature. Only very few variants of P systems are known so far which allow for a deterministic simulation of the SUB-instruction and in that way for a deterministic simulation of a deterministic register machine like the unrestricted variant considered in*

Theorem 4. Therefore I already now want to emphasize this important feature of most of the simple P systems investigated in the following sections to allow for an even deterministic simulation.

4 Catalytic P Systems with One Catalyst

In [12] it was shown that computational completeness can be obtained with simple P systems and only one catalyst when using the derivation mode $max_{objects}$ instead of *max*. Then in [11] computational completeness was established for simple P systems with only one catalyst using the derivation modes $max_{GENobjects}max$, $max_{\Delta objects}max$, $max_{GENobjects}$, and $max_{\Delta objects}$. The results exhibited in this section are optimal with respect to the number of catalysts for catalytic P systems working in these derivation modes, because such P systems when given multisets of non-cooperative rules can only generate semi-linear sets.

Theorem 7. *For any register machine with at least two decrementable registers we can construct a catalytic P system with only one catalyst and working in the derivation mode $max_{objects}$ which can simulate every step of the register machine in $n+1$ steps where n is the number of decrementable registers.*

Proof. We use the trick as elaborated in Remark 1 for getting a specific variant of register machines: without loss of generality, we start with an *ADD*-instruction on register 1, but we also change the register machine program in such a way that after a *SUB*-instruction on register n two intermediate instructions are introduced, i.e., as in Remark 1, we use an *ADD*-instruction on register 1 immediately followed by a *SUB*-instruction on register 1.

We then may simulate the resulting register machine fulfilling these additional constraints $M = (m, B, l_0, l_h, P)$ by a corresponding catalytic P system with one membrane and one catalyst $\Pi = (V, \{c\}, T, (l_0, 1), \mathscr{R})$. Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M, as stated in Proposition 1, Proposition 2, and Proposition 3, fulfills the condition that on the output registers we never apply any *SUB*-instruction. Moreover, we take the most general case of a register machine computing a partial recursive function on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using n decrementable registers, where without loss of generality, we may assume that at the end of a successful computation the first n registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used.

The main idea behind the construction is that all the symbols except the catalyst c and the output symbols (representing the contents of the output registers) go through a cycle of length $n+1$ where n is the number of decrementable registers of the simulated register machine. When the symbols are traversing the r-th section of the first n sections, they "know" that they are to probably simulate a *SUB*-instruction on register r of the register machine M.

$$\begin{aligned} V &= \{a_r \mid n+1 \le r \le m\} \\ &\cup \{(a_r, i) \mid 1 \le r \le n, 1 \le i \le n+1\} \\ &\cup \{(p, i) \mid p \in B_{ADD}, 1 \le i \le n+1\} \\ &\cup \{(p, i) \mid p \in B_{SUB(r)}, \\ &\quad 1 \le i \le r+1, 1 \le r \le n\} \\ &\cup \{(p, i)^-, (p, i)^0 \mid p \in B_{SUB(r)}, \\ &\quad r+2 \le i \le n+1, 1 \le r \le n-1\} \\ &\cup \{c, e, \#\}, \\ T &= \{a_r \mid n+1 \le r \le m\}. \end{aligned}$$

The alphabet V of symbols includes register symbols (a_r, i) for every decrementable register r of the register machine and only the register symbol a_r for each of the k output registers r, $m-k+1 \le r \le m$.

The construction includes the trap rule $\# \to \#$ which will always keep the system busy and prevent it from halting and thus from producing a result as soon as the trap symbol # has been introduced, yet the only rule introducing this trap symbol is the single rule $e \to \#$.

For letting the register symbols cycle with a period of $n+1$ the following rules are used:

$$\begin{aligned} &(a_r, i) \to (a_r, i+1), 1 \le r \le n; \\ &(a_r, n+1) \to (a_r, 1). \end{aligned} \tag{1}$$

For simulating *ADD*-instructions we need the following rules:

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

$$c(p, i) \to c(p, i+1),\ 1 \le i \le n. \tag{2}$$

If r is a decrementable register:

$$\begin{aligned} &c(p, n+1) \to c(q, 1)(a_r, 1), \\ &c(p, n+1) \to c(s, 1)(a_r, 1). \end{aligned} \tag{3}$$

If r is an output register:

$$\begin{aligned} &c(p, n+1) \to c(q, 1)a_r, \\ &c(p, n+1) \to c(s, 1)a_r \end{aligned} \tag{4}$$

The catalyst has to be used with the program symbol which otherwise would stay idle when the catalyst is used with a register symbol, and the multiset of rules applied in this way would use one object less and thus violate the condition of using the maximal number of objects.

For simulating *SUB*-instructions we need the following rules:

Decrement and zero-test $p : (SUB(r), q, s)$*:*

$$\begin{aligned} &c(p, i) \to c(p, i+1),\ 1 \le i < r \\ &(p, r) \to (p, r+1),\ c(a_r, r) \to ce. \end{aligned} \tag{5}$$

In case that register r is empty, i.e., there is no object (a_r, r), then the catalyst will stay idle in step r as there is no other object with which it could react.

If $r < n$:

$$\begin{aligned} &ce \to c,\ \ e \to \#, (p, r+1) \to (p, r+2)^-; \\ &c(p, r+1) \to c(p, r+2)^0. \end{aligned} \tag{6}$$

If in the first step of the simulation phase the catalyst did manage to decrement the register, it produced e. Thus, in the second simulation step, the catalyst has three choices:

1. the catalyst c correctly erases e, and to the program symbol $(p, r+1)$ the rule $(p, r+1) \to (p, r+2)^-$ must be applied due to the derivation mode *maxobjects*; all register symbols evolve in the usual way;

2. the catalyst c takes the program symbol $(p, r+1)$ using the rule $c(p, r+1) \to c(p, r+2)^0$, thus forcing the object e to be trapped by the rule $e \to \#$, and all register symbols evolve in the usual way;

3. the catalyst c takes a register object $(a_{r+1}, r+1)$, thus leaving the object e to be trapped by the rule $e \to \#$, the program symbol $(p, r+1)$ evolves with the rule $(p, r+1) \to (p, r+2)^-$, and all other register objects evolve in the usual way.

In fact, only variant 1 now fulfills the condition given by the derivation mode $max_{objects}$ and therefore is the only possible continuation of the computation if register r is not empty.

On the other hand, if register r is empty, no object e is generated, and the catalyst c has only two choices:

1. the catalyst c takes the program symbol $(p, r+1)$ using the rule $c(p, r+1) \to c(p, r+2)^0$, and all register symbols evolve in the usual way;

2. the catalyst c takes a register object $(a_{r+1}, r+1)$ thereby generating e, the program symbol $(p, r+1)$ evolves with the rule $(p, r+1) \to (p, r+2)^-$, and all other register objects evolve in the usual way; this variant leads to the situation that e will be trapped in step $r+2$, as otherwise the program symbol stays idle, thus violating the condition of the derivation mode $max_{objects}$. Hence, this variant in any case cannot lead to a halting computation due to the introduction of the trap symbol #. We mention that in case no register object $(a_{r+1}, r+1)$ is present we have to apply case 1 and thus have a correct computation step.

Both variants fulfill the condition for the derivation mode $max_{objects}$, but only variant 1 is not introducing the trap symbol # and therefore is the only reasonable continuation of the computation if register r is empty.

$$\begin{aligned} &c(p, i)^- \to c(p, i+1)^-,\ \ r+2 \le i \le n, \\ &c(p, n+1)^- \to c(q, 1), \\ &c(p, i)^0 \to c(p, i+1)^0,\ \ r+2 \le i \le n, \\ &c(p, n+1)^0 \to c(s, 1). \end{aligned} \tag{7}$$

The catalyst has to be used with the program symbol which otherwise would stay idle when the catalyst is used with a register symbol, and the multiset of rules applied in this way would use one object less and thus violate the condition of using the maximal number of objects.

If $r = n$:

$$\begin{aligned} &ce \to c,\ \ e \to \#, \\ &(p, n+1) \to (q, 1),\ \ c(p, n+1) \to c(s, 1). \end{aligned} \tag{8}$$

We observe that in this case during the first step of the next cycle we have to guarantee that in the zero-test case the catalyst must be used with the program symbol, hence, we will simulate an *ADD*-instruction on register 1, as the introduction of the symbol e in the wrong variant of the zero-test case must lead to introducing the trap symbol and not allowing e to be erased by the catalytic rule $ce \to c$.

The HALT-instruction $l_h : HALT$ is simulated by the rule $cl_h \to c$. □

As the number of decrementable registers in generating register machines needed for generating any recursively enumerable set of (vectors of) natural numbers is only two, the following result is an immediate consequence of the preceding theorem:

Corollary 1. *For any generating register machine with two decrementable registers we can construct a catalytic P system with only one catalyst and working in the derivation mode $max_{objects}$ which can simulate every step of the register machine in 3 steps, and therefore such catalytic P systems with only one catalyst and working in the derivation mode $max_{objects}$ can generate any recursively enumerable set of (vectors of) natural numbers.*

The following results even yield deterministic simulations of *SUB*-instructions of a register machine and thus can even avoid trapping.

Theorem 8. *(see [11]) For any register machine with at least two decrementable registers we can construct a simple catalytic P system with only one catalyst, working in the derivation mode $max_{\Delta objects}$max or in the derivation mode $max_{GENobjects}$max, which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

Proof. Given an arbitrary register machine $M = (m,B,l_0,l_h,P)$ we will construct a corresponding catalytic P system with one membrane and one catalyst $\Pi = (V,\{c\},T,w,\mathscr{R})$ simulating M. Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M, as stated in Proposition 1, Proposition 2, and Proposition 3, fulfills the condition that on the output registers we never apply any *SUB*-instruction.

The following proof is given for the most general case of a register machine computing any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used. In fact, the proof works for any number $n \geq 2$ of decrementable registers, no matter how many of them are the l input registers and the working registers, respectively.

The main idea behind the construction is that all the symbols except the catalyst c and the output symbols (representing the contents of the output registers) go through a cycle of length n where n is the number of decrementable registers of the simulated register machine. When the symbols are traversing the r-th section of the n sections, they "know" that they are to probably simulate a *SUB*-instruction on register r of the register machine M.

As in this construction the simulation of a *SUB*-instruction takes two steps, the second simulation step in the case of a *SUB*-instruction on register n is shifted to the first step of the next cycle. Yet in this case we have to guarantee that after a *SUB*-instruction on register n the next instruction to be simulated is not a *SUB*-instruction on register 1. Hence, we use a similar trick as already used in the proof of Theorem 7, i.e., we not only do not start with a *SUB*-instruction, but we also change the register machine program in such a way that after a *SUB*-instruction on register n two intermediate instructions are introduced, we use an *ADD*-instruction on register 1 immediately followed by a *SUB*-instruction on register 1, whose simulation will end at most in step n, as we have assumed $n \geq 2$.

The following construction is elaborated in such a way that it works both for the derivation mode $max_{\Delta objects}max$ and the derivation mode $max_{GENobjects}max$.

We now simulate the resulting register machine fulfilling these additional constraints $M = (m,B,l_0,l_h,P)$ by a corresponding simple P system with one catalyst:

$$\Pi = (V,\{c\},T,c(l_0,1),\mathscr{R}).$$

$$\begin{aligned} V &= \{a_r \mid n+1 \leq r \leq m\} \\ &\cup \{(a_r,i) \mid 1 \leq r \leq n, 1 \leq i \leq n\} \\ &\cup \{(p,i) \mid p \in B_{ADD}, 1 \leq i \leq n\} \\ &\cup \{(p,i) \mid p \in B_{SUB(r)}, 1 \leq i \leq r+1\} \\ &\cup \{(p,i)^-, (p,i)^0 \mid p \in B_{SUB(r)}, r+2 \leq i \leq n\} \\ &\cup \{c,e,d\} \\ T &= \{a_r \mid n+1 \leq r \leq m\}. \end{aligned}$$

The construction includes the dummy symbol d which is erased by the rule $d \to \lambda$. The effect of applying these rules due to the requirement of the chosen multisets of rules to be non-extendable will be ignored in the following calculations for $\Delta obj(C,R)$ and Gen(C,R).

The symbols a_r, $n+1 \leq r \leq m$, represent the output registers. For the decrementable registers, we use the symbols (a_r,i), $1 \leq r \leq n, 1 \leq i \leq n$, which go through a loop of n steps. The main idea now is that the only case when such a symbol can be used to decrement register r is when $i = r$, i.e., in the r-th step of the simulation cycle.

$$(a_r,i) \to (a_r,i+1), 1 \leq r < n; (a_r,n) \to (a_r,1). \quad (9)$$

In the same way as the register symbols a_r, the program symbols (p,i) representing the label p from B undergo the same cycle of length n.

For simulating *ADD*-instructions we need the following rules:

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

$$c(p,i) \to c(p,i+1)d, \ 1 \leq i < n. \quad (10)$$

The catalyst has to be used with the program symbol which otherwise would stay idle when the catalyst is used with a register symbol, and the difference of objects $\Delta obj(C,R')$ for this other non-extendable multiset of rules R' would be 0 whereas when using the program symbol for the catalyst, we obtain $\Delta obj(C,R) = 1$ because of the additional dummy symbol d.

In a similar way we can argue that in the case of the derivation mode $max_{GENobjects}max$ the number of generated objects is maximal when using the catalyst together with the program symbol; in fact, if N is the total number of register symbols for decrementable registers in the underlying configuration C, then with applying the set of rules R described so far we get Gen$(C,R) = N+3$ in contrast to Gen$(C,R') = N-1+3 = N+2$ where using the catalyst with the rule $c(a_r,r) \to ced$, as described below for the simulation of the *SUB*-Instruction, results in the multiset of rules R'.

If r is a decrementable register, we end the simulation using one of the following rules:

$$c(p,n) \to c(q,1)(a_r,1), c(p,n) \to c(s,1)(a_r,1). \quad (11)$$

If r is an output register, we end the simulation using one of the following rules introducing output symbols not to be changed any more:

$$c(p,n) \to c(q,1)a_r, c(p,n) \to c(s,1)a_r. \tag{12}$$

As in both cases, together with the program symbol a new register symbol is generated, we again have $\Delta obj(C,R) = 1$, thus guaranteeing that the catalyst must take (p,n) and cannot take (a_n,n) instead.

A similar argument again holds in the case of the derivation mode $max_{GENobjects}max$ as the number of generated objects is only maximal when using the catalyst together with the program symbol; again we have Gen$(C,R)= N+3$ with this multiset of rules R in contrast to Gen$(C,R')= N-1+3 = N+2$ when using the catalyst with the rule $c(a_r,r) \to ced$ results in the multiset of rules R'.

For simulating *SUB*-instructions we need the following rules:

Decrement and zero-test $p : (SUB(r),q,s)$*:*

$$c(p,i) \to c(p,i+1)d,\ 1 \le i < r. \tag{13}$$

For $1 \le i < r$, we again use the dummy symbol d to obtain $\Delta C = 1$ and thus also having one more object generated, to enforce the catalyst to take the program symbol.

$$(p,r) \to (p,r+1),\ c(a_r,r) \to ced. \tag{14}$$

In case that register r is empty, i.e., there is no object (a_r,r), then the catalyst will stay idle as in this step there is no other object with which it could react. In case that register r is not empty, i.e., there is at least one object (a_r,r), then one of these objects (a_r,r) must be used with the catalyst c as the rule $c(a_r,r) \to ced$ implies $\Delta obj(C,R)= 1$, whereas otherwise, if all register symbols are used with the rule $(a_r,r) \to (a_r,r+1)$, then $\Delta obj(C,R)= 0$.

In the same way we argue that with using the rule $c(a_r,r) \to ced$ we get one object generated more than if we use the rule $(a_r,r) \to (a_r,r+1)$ for that symbol (a_r,r), i.e., Gen$(C,R)= N-1+3 = N+2$ in contrast to Gen$(C,R')= N$.

If $r < n-1$:

$$\begin{aligned} &ce \to cdddd,\ (p,r+1) \to (p,r+2)^-, \\ &c(p,r+1) \to c(p,r+2)^0 dd. \end{aligned} \tag{15}$$

If in the first step of the simulation phase the catalyst did manage to decrement the register, it produced e. Thus, in the second simulation step, the catalyst has three choices:

1. the catalyst c correctly "erases" e using the rule $ce \to cdddd$, and to the program symbol $(p,r+1)$ the rule $(p,r+1) \to (p,r+2)^-$ must be applied due to the fact that both derivation modes $max_{\Delta objects}max$ and $max_{GENobjects}max$ only allow for non-extendable multisets of rules; all register symbols evolve in the usual way; in total we get $\Delta obj(C,R)= 3$ and Gen$(C,R)= N+6$;
2. the catalyst c takes the program symbol $(p,r+1)$ using the rule $c(p,r+1) \to c(p,r+2)^0 dd$, and all register symbols evolve in the usual way; in total we get $\Delta obj(C,R)= 2$ and Gen$(C,R)= N+4$;
3. the catalyst c takes a register object, the program symbol $(p,r+1)$ evolves with the rule $(p,r+1) \to (p,r+2)^-$, and all other register objects evolve in the usual way; in total we get $\Delta obj(C,R)= 1$ and Gen$(C,R)= (N-1+3)+1 = N+3$.

In total, only variant 1 fulfills the condition given by the derivation mode $max_{\Delta objects}max$ that $\Delta obj(C,R)$ is maximal, and therefore is the only possible continuation of the computation if register r is not empty.

A similar argument holds for the derivation mode $max_{Genobjects}max$ with respect to the number of generated objects $\Delta obj(C,R)$.

On the other hand, if register r is empty, no object e is generated, and the catalyst c has only two choices:

1. the catalyst c takes the program symbol $(p,r+1)$ using the rule $c(p,r+1) \to c(p,r+2)^0 dd$, and all register symbols evolve in the usual way; in total we get $\Delta obj(C,R)= 2$ and Gen$(C,R)= N+4$;
2. the catalyst c takes a register object $(a_{r+1},r+1)$ thereby generating ed, the program symbol $(p,r+1)$ evolves with the rule $(p,r+1) \to (p,r+2)^-$, and all other register objects evolve in the usual way; this variant leads to $\Delta obj(C,R)= 1$ and Gen$(C,R)= (N-1+3)+1 = N+3$.

In total, variant 1 is the only possible continuation of the computation if register r is empty.

$$\begin{aligned} &c(p,i)^- \to c(p,i+1)^- d,\ \ r+2 \le i < n, \\ &c(p,n)^- \to c(q,1)d, \\ &c(p,i)^0 \to c(p,i+1)^0 d,\ \ r+2 \le i < n, \\ &c(p,n)^0 \to c(s,1)d. \end{aligned} \tag{16}$$

Again the catalyst has to be used with the program symbol to get $\Delta obj(C,R)= 1$ and Gen$(C,R)= N+3$, which otherwise would stay idle when the catalyst is used with a register symbol, and the multiset of rules applied in this way would only yield $\Delta obj(C,R)= 0$ and Gen$(C,R')= N-1+3 = N+2$.

If $r = n-1$:

$$\begin{aligned} &ce \to cdddd,\ (p,n) \to (q,1), \\ &c(p,n) \to c(s,1)dd. \end{aligned} \tag{17}$$

In this case, we directly go to the first step of the next cycle.

If $r = n$:

$$\begin{aligned} &ce \to cdddd,\ (p,n+1) \to (q,2), \\ &c(p,n+1) \to c(s,2)dd. \end{aligned} \tag{18}$$

In this case, the second step of the simulation is already the first step of the next cycle, which means that in this case of $r = n$ the next instruction to be simulated is an *ADD*-instruction on register 1.

To complete the proof we have to implement the final *HALT*-instruction $l_h : HALT$ with the rule $c(l_h, 1) \to cdd$. In this way, finally no program symbol is present any more in the configuration. As we have assumed all decrementable registers to be empty when the register machine halts, this means the constructed simple P system will also halt after having erased the dummy symbols d in the next step.

We finally observe that the proof construction given above is even deterministic if the underlying register machine to be simulated is deterministic. □

In a similar way, the same result can even be shown for the derivation modes $max_{\Delta objects}$ and $max_{GENobjects}$, where the condition of non-extendability for the multisets of rules to be applied is not required.

Theorem 9. *For any register machine with at least two decrementable registers we can construct a simple catalytic P system with only one catalyst, working in the derivation mode $max_{\Delta objects}$ or in the derivation mode $max_{GENobjects}$, which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

As the number of decrementable registers in generating register machines needed for generating any recursively enumerable set of (vectors of) natural numbers is only two, from the theorems above we obtain the following result:

Corollary 2. *For any generating register machine with two decrementable registers we can construct a simple P system with only one catalyst and working in the derivation mode $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, or $max_{GENobjects}$ which can simulate every step of the register machine in 2 steps, and therefore such catalytic P systems with only one catalyst and working in the in the derivation mode $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, or $max_{GENobjects}max$ can generate any recursively enumerable set of (vectors of) natural numbers.*

The even more important achievement than the rather expected computational completeness established with (the proof of) Theorem 8 and Theorem 9 is the fact that with the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, and $max_{GENobjects}$ only one catalyst is needed to obtain computational completeness, which is the optimal result with respect to the number of catalysts, because with non-cooperative rules, only semilinear sets can be generated. Moreover, the simulation of a deterministic register machine is deterministic in the P system, too.

5 Purely Catalytic P Systems

The technique used for catalytic P systems in the proof of Theorem 8 cannot be taken over to purely catalytic P systems, where the number of rules to be used in every step is bounded by the number of catalysts. Hence, a similar technique as already known from the proof of the classic result given in Theorem 4 is used for proving Theorem 10, yet still the result to be obtained with the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, and $max_{GENobjects}$ is better than that one known for the derivation mode max, because one catalyst less is needed.

Theorem 10. *For any register machine with $n \geq 2$ decrementable registers we can construct a simple purely catalytic P system with only n catalysts, working in one of the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, or $max_{GENobjects}$, which can simulate any computation of the register machine.*

Proof. Given an arbitrary register machine $M = (m, B, l_0, l_h, P)$ with n decrementable registers we will construct a corresponding simple purely catalytic P system with n catalysts

$$\Pi = (V, \{c_k \mid 1 \leq k \leq n\}, T, w, \mathscr{R})$$

simulating M. Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M, as stated in Proposition 1, Proposition 2, and Proposition 3, fulfills the condition that on the output registers we never apply any *SUB*-instruction. Moreover, according to Remark 1 we may assume that the first instruction is an *ADD*-instruction on the first register. Finally, we assume the n decrementable registers to be the first ones.

The following proof again is elaborated for all the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, and $max_{GENobjects}$, with only a few subtle technical details to be mentioned additionally.

The main part of the proof is to show how to simulate the instructions of M in Π; in all cases we have to take care that the n catalysts are kept busy – using corresponding dummy objects d_r – in order to guarantee that the simulation is executed in a correct way; especially we have to guarantee that one of the rules using the catalysts $c_k, 1 \leq k \leq n$, must be used if possible, i.e., a catalyst can only stay idle if the underlying configuration does not contain any object which can evolve together with the catalyst. Again the priority between different rules for a catalyst is guarded by the number of objects on the right-hand side of the rules, which argument applies for all the derivation modes under consideration, as every rule in a purely catalytic P system has exactly two objects on its left-hand side.

During the simulation of all instructions, we use the following multisets:

$$D'_{n,r} \quad = \quad \prod_{i \in [1..n] \setminus \{r, r \oplus_n 1\}} d_i, \; 1 \leq r \leq n.$$

As the first instruction to be simulated is an *ADD*-instruction on the first register, we start with the initial multiset

$$w = l_0 l_0' D_{n,1}' \prod_{i \in [1..n]} c_i.$$

As usual, the number of objects a_r in a configuration represents the number stored in register r at that moment of the computation. Objects a_r for $r > n$ are never changed again, as they represent output registers.

$$\begin{aligned} V &= \{a_r \mid 1 \le r \le m\} \cup \{\hat{a}_r \mid 1 \le r \le n\} \\ &\cup \{p, p' \mid p \in B_{ADD} \cup \{l_h\}\} \\ &\cup \{p, p', \bar{p}, \hat{p} \mid p \in B_{SUB(r)}, 1 \le r \le n\} \\ &\cup \{c_k, d_k \mid 1 \le k \le n\} \cup \{d\}, \\ T &= \{a_r \mid n+1 \le r \le m\}. \end{aligned}$$

The dummy objects d_i, $1 \le i \le n$, are used to keep the corresponding catalyst c_i busy whenever it is not needed during the simulation of a *SUB*-instruction, which is accomplished by the following rule erasing d_i, but instead introducing the necessary amount of objects d to keep the catalyst c_i away from erasing a register object a_r:

$$c_i d_i \to c_i d^4,\ 1 \le k \le n.$$

Moreover, for erasing d we use the rules

$$c_k d \to c_k,\ 1 \le k \le n.$$

In the derivation mode $max_{\Delta objects}$ these erasing rules can only be used at the end of a computation when no other rules can be applied any more.

The remaining rules in the set $\mathscr{R}$ of catalytic rules can be captured from the description of how the simulation of the register machine instructions works as described in the following:

- $p : (ADD(r), q, s)$, with $p \in B_{ADD}$, $q, s \in B$, $1 \le r \le m$.

  An *ADD*-instruction can be simulated in one step by letting every catalyst make one evolution step:

  $$\begin{aligned} &c_{Reg(p)} p \to c_{Reg(p)} q q' a_r d D_{n,Reg(q)}' \text{ or} \\ &c_{Reg(p)} p \to c_{Reg(p)} s s' a_r d D_{n,Reg(s)}', \\ &c_{Reg(p) \oplus_n 1} p' \to c_2 d^4. \end{aligned}$$

  We recall that all other catalysts c_i with $i \in [1..n] \setminus \{Reg(p), Reg(p) \oplus_n 1\}$ are forced to apply the rule $c_i d_i \to c_i d^4$. The dummy objects d are used to guarantee that the rules given above, with in sum at least 5 objects on their right-hand sides, have priority over the rules $c_r a_r \to c_r \hat{a}_r d^2$, $1 \le r \le n$, with in sum only 4 objects on their right-hand sides.

- $p : (SUB(r), q, s)$, with $p \in B_{SUB}$, $q, s \in B$, $1 \le r \le n$.

  decrement case

  If the value of register r (denoted by $|reg(r)|$) is not zero then the number of register objects a_r is decreased by one using the corresponding rule $c_r a_r \to c_r \hat{a}_r d^2$ in the first step of the simulation. In sum three steps are needed for the simulation, see table below.

  zero-test case.

  If the value of register r is zero, then the corresponding catalyst is already free for eliminating the label object p so that already in the second step of the simulation the simulation of the next instruction s can be initiated. In sum only two steps are needed for the simulation of this case, see table below.

  The following table summarizes the rules to be used for the simulation of the *SUB*-instruction on register r, $1 \le r \le n$, i.e., we use the following rules; we emphasize that again the simulation is *deterministic*.

step	$\lvert reg(r) \rvert$	rule for c_r and $c_{r \oplus_n 1}$
1	> 0	$c_r a_r \to c_r \hat{a}_r d^2$
		$c_{r \oplus_n 1} p' \to c_{r \oplus_n 1} \bar{p} d^{10} D_{n,Reg(p)}'$
	$= 0$	$c_r p \to c_r d^2$
		$c_{r \oplus_n 1} p' \to c_{r \oplus_n 1} \bar{p} d^{10} D_{n,Reg(p)}'$
2	> 0	$c_r \bar{p} \to c_r \hat{p} d^3$
		$c_{r \oplus_n 1} p \to c_{r \oplus_n 1} d^9 D_{n,Reg(p)}'$
	$= 0$	$c_r d \to c_r$ $(^*)$
		$c_{r \oplus_n 1} \bar{p} \to c_{r \oplus_n 1} s s' d^6 D_{n,Reg(s)}'$
3	> 0	$c_r \hat{p} \to c_1 q q' d^2 D_{n,Reg(q)}'$
		$c_{r \oplus_n 1} \hat{a}_r \to c_{r \oplus_n 1} d^4$

  The the rule $c_r d \to c_r$ marked with $(^*)$ is only applied in the derivation modes $max_{\Delta objects} max$ and $max_{GENobjects} max$ as well as $max_{GENobjects}$, whereas in the derivation mode $max_{\Delta objects}$ it will not be applied as it would decrease the difference between generated and consumed objects.

- $l_h : HALT$.

  Taking into account that we have defined $Reg(l_h) = 1$, we take:

  $$\begin{aligned} c_1 l_h &\to c_1 dd \\ c_2 l_h' &\to c_2 dd \end{aligned}$$

After the register machine has halted (with the first n registers being empty), which is simulated by the rules

above, finally all dummy objects generated during the simulation steps before are deleted by using the rules

$$c_i d \to c_i,\ 1 \leq i \leq n.$$

Whereas in the derivation modes $max_{\Delta objects}max$ and $max_{GENobjects}max$ as well as $max_{GENobjects}$ some of these objects d can already be erased during the simulation of *SUB*-instructions, see above, in the derivation mode $max_{\Delta objects}$, these erasing rules are only executed at the end of the computation. These observations complete the proof. □

As a consequence, we obtain the following result:

Corollary 3. *Purely catalytic P systems working in any of the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, or $max_{GENobjects}$ are computationally complete, i.e., they can compute any partial recursive relation on natural numbers.*

Yet besides this computational completeness result, the even more relevant achievement of the result established with Theorem 10 is the fact that, when we compare with the results given in (the proof of) Theorem 5, with all these new derivation modes, i.e., $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, and $max_{GENobjects}$, only one catalyst for each decrementable register is needed, which is an improvement of needing one catalyst less than with the derivation mode *max*, and moreover the simulation is deterministic, hence, no trapping is needed.

6 Conclusion

In this overview paper I have collected several classic as well as many new results established just recently for simple P systems working in variants of the maximally parallel derivation mode allowing for computational completeness. In case of the parallel derivation modes (i) affecting or (ii) generating the maximal number of objects or (iii) yielding the maximal difference between the objects in the current and the derived configuration, in simple catalytic P systems only *one* catalyst is needed to obtain computational completeness, which is the optimal result with respect to the number of catalysts, because with non-cooperative rules only semi-linear sets can be obtained. In case of simple purely catalytic P systems at least one catalyst less is needed than in the classic proofs showing computational completeness.

Acknowledgements

I am very grateful to Gheorghe Păun for involving me from the beginning in this new area of membrane systems. Moreover, many results as presented above have been developed together with my other co-authors, especially with my colleague Marion Oswald at the TU Wien and the "Moldovan team" Artiom Alhazov, Sergiu Ivanov, and Sergey Verlan.

References

[1] Alhazov, A., Aman, B., Freund, R.: P systems with anti-matter. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 66–85. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_5

[2] Alhazov, A., Aman, B., Freund, R., Păun, Gh.: Matter and anti-matter in membrane systems. In: Jürgensen, H., Karhumäki, J., Okhotin, A. (eds.) Descriptional Complexity of Formal Systems – 16th International Workshop, DCFS 2014, Turku, Finland, August 5–8, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8614, pp. 65–76. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_7

[3] Alhazov, A., Freund, R.: P systems with toxic objects. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8961, pp. 99–125. Springer (2014). https://doi.org/10.1007/978-3-319-14370-5_7

[4] Alhazov, A., Freund, R.: Small catalytic P systems. In: Dinneen, M. (ed.) Proceedings of the Workshop on Membrane Computing 2015 (WMC2015), (Satellite Workshop of UCNC2015), August 2015, CDMTCS Research Report Series, vol. 487, pp. 1–16. Centre for Discrete Mathematics and Theoretical Computer Science, Department of Computer Science, University of Auckland, Auckland, New Zealand (2015)

[5] Alhazov, A., Freund, R.: Variants of small universal P systems with catalysts. Fundam. Informaticae **138**(1–2), 227–250 (2015). https://doi.org/10.3233/FI-2015-1209

[6] Alhazov, A., Freund, R., Ivanov, S.: Variants of energy-controlled P systems. In: Proceedings of NIT 2016 (2016)

[7] Alhazov, A., Freund, R., Ivanov, S.: Variants of P systems with activation and blocking of rules. Nat. Comput. **18**(3), 593–608 (2019). https://doi.org/10.1007/s11047-019-09747-5

[8] Alhazov, A., Freund, R., Ivanov, S.: Catalytic P systems with weak priority of catalytic rules. In: Freund, R. (ed.) Proceedings ICMC 2020, September 14–18, 2020, pp. 67–82. TU Wien (2020)

[9] Alhazov, A., Freund, R., Ivanov, S.: P systems with limiting the number of objects in membranes. In: Freund, R. (ed.) Proceedings ICMC 2020, September 14–18, 2020, pp. 83–98. TU Wien (2020)

[10] Alhazov, A., Freund, R., Ivanov, S.: P systems with limited number of objects. Journal of Membrane Computing **3**, 1–9 (2021). https://doi.org/10.1007/s41965-020-00068-6

[11] Alhazov, A., Freund, R., Ivanov, S.: Variants of simple P systems with one catalyst being computationally complete. In: Vaszil, Gy. (ed.) Proceedings ICMC 2021 (2021)

[12] Alhazov, A., Freund, R., Ivanov, S.: When catalytic P systems with one catalyst can be computationally complete. Journal of Membrane Computing (2021). https://doi.org/10.1007/s41965-021-00079-x

[13] Alhazov, A., Freund, R., Ivanov, S., Oswald, M.: Variants of simple purely catalytic P systems with two catalysts. In: Vaszil, Gy. (ed.) Proceedings ICMC 2021 (2021)

[14] Alhazov, A., Freund, R., Ivanov, S., Verlan, S.: (Tissue) P systems with vesicles of multisets. In: Csuhaj-Varjú, E., Dömösi, P., Vaszil, Gy. (eds.) Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4-6, 2017. EPTCS, vol. 252, pp. 11–25 (2017). https://doi.org/10.4204/EPTCS.252.6

[15] Alhazov, A., Freund, R., Leporati, A., Oswald, M., Zandron, C.: (Tissue) P systems with unit rules and energy assigned to membranes. Fundam. Informaticae **74**(4), 391–408 (2006)

[16] Alhazov, A., Freund, R., Oswald, M., Verlan, S.: Partial halting and minimal parallelism based on arbitrary rule partitions. Fundam. Inform. **91**(1), 17–34 (2009). https://doi.org/10.3233/FI-2009-0031

[17] Alhazov, A., Freund, R., Sosík, P.: Small P systems with catalysts or anti-matter simulating generalized register machines and generalized counter automata. Comput. Sci. J. Moldova **23**(3), 304–328 (2015)

[18] Alhazov, A., Freund, R., Verlan, S.: P systems working in maximal variants of the set derivation mode. In: Leporati, A., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing – 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10105, pp. 83–102. Springer (2017). https://doi.org/10.1007/978-3-319-54072-6_6

[19] Dassow, J., Păun, Gh.: Regulated Rewriting in Formal Language Theory. Springer (1989)

[20] Freund, R.: Energy–controlled P systems. In: Păun, Gh., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) Membrane Computing, pp. 247–260. Springer (2003)

[21] Freund, R.: Purely catalytic P systems: Two catalysts can be sufficient for computational completeness. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu. (eds.) CMC14 Proceedings – The 14th International Conference on Membrane Computing, Chișinău, August 20–23, 2013, pp. 153–166. Institute of Mathematics and Computer Science, Academy of Sciences of Moldova (2013)

[22] Freund, R.: P automata: New ideas and results. In: Bordihn, H., Freund, R., Nagy, B., Vaszil, Gy. (eds.) Eighth Workshop on Non-Classical Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29–30, 2016. Proceedings. books@ocg.at, vol. 321, pp. 13–40. Österreichische Computer Gesellschaft (2016)

[23] Freund, R.: How derivation modes and halting conditions may influence the computational power of P systems. Journal of Membrane Computing **2**(1), 14–25 (2020). https://doi.org/10.1007/s41965-019-00028-9

[24] Freund, R., Kari, L., Oswald, M., Sosík, P.: Computationally universal P systems without priorities: two catalysts are sufficient. Theoretical Computer Science **330**(2), 251–266 (2005). https://doi.org/10.1016/j.tcs.2004.06.029

[25] Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C.: Flattening in (tissue) P systems. In: Alhazov, A., Cojocaru, S., Gheorghe, M., Rogozhin, Yu., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 8340, pp. 173–188. Springer (2014). https://doi.org/10.1007/978-3-642-54239-8_13

[26] Freund, R., Oswald, M.: Partial halting in P systems. Int. J. Found. Comput. Sci. **18**(6), 1215–1225 (2007). https://doi.org/10.1142/S0129054107005261

[27] Freund, R., Oswald, M.: Catalytic and purely catalytic P automata: control mechanisms for obtaining computational completeness. In: Bensch, S., Drewes, F., Freund, R., Otto, F. (eds.) Fifth Workshop on Non-Classical Models for Automata and Applications – NCMA 2013, Umeå, Sweden, August 13 – August 14, 2013, Proceedings. books@ocg.at, vol. 294, pp. 133–150. Österreichische Computer Gesellschaft (2013)

[28] Freund, R., Oswald, M., Păun, Gh.: Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. Fundam. Inform. **136**(1–2), 59–84 (2015). https://doi.org/10.3233/FI-2015-1144

[29] Freund, R., Păun, Gh.: How to obtain computational completeness in P systems with one catalyst. In: Neary, T., Cook, M. (eds.) Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9–11, 2013. EPTCS, vol. 128, pp. 47–61 (2013). https://doi.org/10.4204/EPTCS.128.13

[30] Freund, R., Păun, Gh., Pérez-Jiménez, M.J.: Polarizationless P systems with active membranes working in the minimally parallel mode. In: Akl, S.G., Calude, C.S., Dinneen, M.J., Rozenberg, G., Wareham, T. (eds.) Unconventional Computation, 6th International Conference, UC 2007, Kingston, Canada, August 13-17, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4618, pp. 62–76. Springer (2007). https://doi.org/10.1007/978-3-540-73554-0_8

[31] Freund, R., Rogozhin, Yu., Verlan, S.: P systems with minimal left and right insertion and deletion. In: Durand-Lose, J., Jonoska, N. (eds.) Unconventional Computation and Natural Computation – 11th International Conference, UCNC 2012, Orléan, France, September 3–7, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7445, pp. 82–93. Springer (2012). https://doi.org/10.1007/978-3-642-32894-7_9

[32] Freund, R., Sosík, P.: On the power of catalytic P systems with one catalyst. In: Rozenberg, G., Salomaa, A., Sempere, J.M., Zandron, C. (eds.) Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9504, pp. 137–152. Springer (2015). https://doi.org/10.1007/978-3-319-28475-0_10

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

[34] Freund, R., Verlan, S.: (Tissue) P systems working in the k-restricted minimally or maximally parallel transition mode. Nat. Comput. **10**(2), 821–833 (2011). https://doi.org/10.1007/s11047-010-9215-z

[35] Krithivasan, K., Păun, Gh., Ramanujan, A.: On controlled P systems. Fundam. Inform. **131**(3–4), 451–464 (2014). https://doi.org/10.3233/FI-2014-1025

[36] Minsky, M.L.: Computation. Finite and Infinite Machines. Prentice Hall, Englewood Cliffs, NJ (1967)

[37] Păun, Gh.: Computing with membranes. Journal of Computer and System Sciences **61**(1), 108–143 (2000). https://doi.org/10.1006/jcss.1999.1693

[38] Păun, Gh.: Membrane Computing: An Introduction. Springer (2002). https://doi.org/10.1007/978-3-642-56196-2

[39] Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): The Oxford Handbook of Membrane Computing. Oxford University Press (2010)

[40] Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer (1997). https://doi.org/10.1007/978-3-642-59136-5

[41] Sosík, P., Langer, M.: Small (purely) catalytic P systems simulating register machines. Theoretical Computer Science **623**, 65–74 (2016). https://doi.org/10.1016/j.tcs.2015.09.020

[42] The P Systems Website. `http://ppage.psystems.eu/`