Variants of derivation modes for which catalytic P systems with one catalyst are computationally complete

Artiom Alhazov, Rudolf Freund, Sergiu Ivanov, Sergey Verlan

Journal of Membrane Computing (2021) 3:233–245
https://doi.org/10.1007/s41965-021-00085-z

REGULAR PAPER

Variants of derivation modes for which catalytic P systems with one catalyst are computationally complete

Artiom Alhazov[1] · Rudolf Freund[2] · Sergiu Ivanov[3] · Sergey Verlan[4]

Received: 9 July 2021 / Accepted: 12 October 2021 / Published online: 15 November 2021

Abstract
Catalytic P systems are among the first variants of membrane systems ever considered in this area. This variant of systems also features some prominent computational complexity questions, and in particular the problem of using only one catalyst: is one catalyst enough to allow for generating all recursively enumerable sets of multisets? Several additional ingredients have been shown to be sufficient for obtaining even computational completeness with only one catalyst. Last year we could show that the derivation mode $max_{objects}$, where we only take those multisets of rules which affect the maximal number of objects in the underlying configuration one catalyst is sufficient for obtaining computational completeness without any other ingredients. In this paper we follow this way of research and show that one catalyst is also sufficient for obtaining computational completeness when using specific variants of derivation modes based on non-extendable multisets of rules: we only take those non-extendable multisets whose application yields the maximal number of generated objects or else those non-extendable multisets whose application yields the maximal difference in the number of objects between the newly generated configuration and the current configuration. A similar computational completeness result can even be obtained when omitting the condition of non-extendability of the applied multisets when taking the maximal difference of objects or the maximal number of generated objects. Moreover, we reconsider simple P system with energy control—both symbol and rule energy-controlled P systems equipped with these new variants of derivation modes yield computational completeness.

1 Introduction

Two decades ago, membrane systems were introduced in [35] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. The development of this fascinating area of biologically motivated computing models is documented in two textbooks, see [36] and [37]. For actual information see the P systems webpage [39] and the issues of the Bulletin of the International Membrane Computing Society and of the Journal of Membrane Computing.

One basic feature of P systems already presented in [35] is the maximally parallel derivation mode, i.e., using non-extendable multisets of rules in every derivation step. The result of a computation can be extracted when the system halts, i.e., when no rule is applicable any more. Catalysts are special symbols which allow only one object to evolve in its context (in contrast to promoters) and in their basic variant never evolve themselves, i.e., a catalytic rule is of the form $ca \rightarrow cv$, where c is a catalyst, a is a single object and v is a multiset of objects. In contrast, non-catalytic rules in catalytic P systems are non-cooperative rules of the form $a \rightarrow v$.

From the beginning, the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges regarding (catalytic) P systems. In [21] it has already been shown that two catalysts are enough for generating any

✉ Rudolf Freund
rudi@emcc.at

Artiom Alhazov
artiom@math.md

Sergiu Ivanov
sergiu.ivanov@ibisc.univ-evry.fr

Sergey Verlan
verlan@u-pec.fr

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science, Academiei 5, Chişinău, MD 2028, Moldova

[2] Faculty of Informatics, TU Wien, Favoritenstraße 9–11, 1040 Wien, Austria

[3] IBISC, Université Évry, Paris-Saclay, 23, boulevard de France, 91034 Évry, France

[4] Univ. Paris Est Creteil, LACL, 94010 Creteil, France

recursively enumerable set of multisets, without any additional ingredients like a priority relation on the rules as used in the original definition. As already known from the beginning, without catalysts only regular (semi-linear) sets can be generated when using the standard maximal derivation mode and the standard halting mode, i.e., a result is extracted when the system halts with no rule being applicable any more. As shown, for example, in [26], using various additional ingredients, i.e., additional control mechanisms, one catalyst can be sufficient: 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 [1–14, 17–20, 23, 24, 26–29, 32, 33].

On the other hand, for such catalytic P systems with only one catalyst and using the standard maximally parallel derivation mode and the standard halting mode, a lower bound has been established in [30]: P systems with one catalyst can simulate partially blind register machines, i.e., they can generate more than just semi-linear sets.

In [6], we returned to the idea of using a priority relation on the rules, but took only a very weak form of such a priority relation: we only required that overall in the system catalytic rules have weak priority over non-catalytic rules. This means that the catalyst c must not stay idle if the current configuration contains an object a with which it may cooperate in a rule $ca \to cv$; all remaining objects evolve in the maximally parallel way with non-cooperative rules. On the other hand, if the current configuration does not contain an object a with which the catalyst c may cooperate in a rule $ca \to cv$, c may stay idle and *all* objects evolve in the maximally parallel way with non-cooperative rules. Even without using more than this weak priority of catalytic rules over the non-catalytic (non-cooperative) rules, we could establish computational completeness for catalytic P systems with only one catalyst. Moreover, starting from a result established in [6], an even stronger result using a similar construction as in [6] has been established in [9] where we show computational completeness for catalytic P systems with only one catalyst using the derivation mode $max_{objects}$, i.e., we only take those multisets of rules which affect the maximal number of objects in the underlying configuration.

In this paper we now continue the research started in [9] and investigate several variants of derivation modes based on non-extendable multisets of rules and taking only those for which the difference of objects between the underlying configuration and the configuration after the application of the multisets of rules is maximal. We also consider the variants where the number of objects generated by the application of a multiset of rules is maximal.

Finally, for the variants with maximal number of objects we can also take these multisets of rules without requesting them to fulfill the condition of the multisets to be non-extendable.

In this context, we also reconsider P systems with energy control as first presented in [4] and then further developed for spiking neural P systems in [25]. Equipped with each of the newly defined derivation modes we obtain computational completeness for both symbol and rule energy-controlled P systems.

2 Definitions

For an alphabet V, by V^* we denote the free monoid generated by V under the operation of concatenation, i.e., containing all possible strings over 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 we refer to textbooks like [16] and [38].

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 [9].

Definition 1 A *register machine* is a construct

$$M = \left(m, B, l_0, l_h, P\right)$$

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:

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

  Increase the value of register r by one, and non-deterministically jump to instruction q or s.
- $p : (SUB(r), q, s)$, with $p \in B \setminus \{l_h\}, q, s \in B, 1 \leq r \leq m$.

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

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 [34]; 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: given a register machine $M = (m, B, l_0, l_h, P)$ with having 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:

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

2.2 Simple catalytic P systems

Taking into account the well-known flattening process, e.g., see [22], in this paper we only consider simple catalytic P systems, i.e., with the simplest membrane structure of only one membrane, and with only one catalyst:

Definition 2 A *simple catalytic P system* with only one catalyst is a construct

$$\Pi = (V, \{c\}, T, w, \mathcal{R})$$

where

- V is the alphabet of objects;
- $c \in V$ is the single catalyst;
- $T \subseteq (V \setminus \{c\})$;
- $w \in V^\circ$ is the multiset of objects initially present in the membrane region;

– $\mathcal{R}$ is a finite set of *evolution rules* over V; these evolution rules are of the forms $ca \to cv$ or $a \to v$, where c is a catalyst, a is an object from $V \setminus \{c\}$, and v is a multiset over $V \setminus \{c\}$.

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 [31] and extend them for the purposes of this paper.

Given a P system $\Pi = (V, \{c\}, T, w, \mathcal{R})$, 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 $C \xrightarrow{R} C'$ 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 $\Delta obj(C, R)$. In all cases, the catalysts are taken into account, too.

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 $\mathcal{R}$ 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:

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

As already introduced for multisets of rules in [15], we now consider the variant where the maximal number of rules is chosen. In the derivation mode $max_{rules}max$ only a maximal multiset of rules is allowed to be applied. But it can also be seen as the variant of the basic mode *max* where we just take a multiset of applicable rules with the maximal number of rules in it, hence, we will also call it the max_{rules} derivation mode. Formally we have:

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

We also 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.

Remark 2 The inherent possibility of mimicking the weak priority of catalytic rules over non-catalytic rules taken from the set of applicable non-extendable multisets of rules in simple catalytic P systems using the derivation mode $max_{objects}$ allowed us to show that simple catalytic P systems with only one catalyst are computationally complete when using the derivation mode $max_{objects}$, see [9] .

We now define new derivation modes starting from the non-extendable multisets of rules applicable to the current configuration C as for the derivation mode *max*, which instead of looking at the number of affected objects taken 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:

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

We illustrate the difference between these new derivation modes in the following example:

Example 1 To illustrate 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 following rules:

1. $a \to b$
2. $ca \to cd$

In case of the derivation mode $max_{GENobjects}max$, only the multiset of rules $\{ca \to cd, a \to b\}$ can be applied, as $\mathrm{Gen}(caa, \{ca \to cd\}) = 2$ and $\mathrm{Gen}(cab, \{a \to b\}) = 1$ and therefore $\mathrm{Gen}(caa, \{ca \to cd, a \to b\}) = 3$, whereas $\mathrm{Gen}(caa, \{a \to b, a \to b\}) = 2$. 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 a slightly different set of rules:

1. $a \to bb$
2. $ca \to cd$

Observing that $\mathrm{Gen}(caa, \{a \to bb\}) = 2$ and $\Delta obj(caa, \{a \to bb\}) = 1$, we obtain the following sets of applicable multisets of rules in the two derivation modes considered here in this example:

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

Finally, let us take the following set of rules:

1. $a \to \lambda$
2. $ca \to cd$

Observing that $\mathrm{Gen}(caa, \{a \to \lambda\}) = 0$ and $\Delta obj(caa, \{a \to \lambda\}) = -1$, we obtain the following sets of applicable multisets of rules in the two derivation modes considered here in this example:

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

As for $max_{objects}max$ we now can also consider the variants where the restriction $max_{objects}$ is imposed on the applicable multisets without first requiring them to be non-extendable:

$max_{GENobjects}$ 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:

$Appl(\Pi, C, max_{GENobjects})$
$= \{R \in Appl(\Pi, C, asyn) \mid$
there is no $R' \in Appl(\Pi, C, asyn)$
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 multisets of rules:

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

Example 2 To illlustrate the derivation modes $max_{GENobjects}$ and $max_{\Delta objects}$, consider a simple P system with the initial configuration caa and the following rules:

1. $a \to bb$
2. $ca \to cdd$

In case of the derivation mode $max_{GENobjects}$, only the multiset of rules $\{ca \to cdd, a \to bb\}$ can be applied, as $\mathrm{Gen}(caa, \{ca \to cdd\}) = 3$ and $\mathrm{Gen}(cab, \{a \to bb\}) = 2$ and therefore $\mathrm{Gen}(caa, \{ca \to cdd, a \to bb\}) = 5$, whereas $\mathrm{Gen}(caa, \{a \to bb, a \to bb\}) = 4$. Hence, the only possible derivation with the derivation mode $max_{GENobjects}$ is $caa \xrightarrow{\{ca\to cdd, a\to bb\}} cddbb$. In this special case,

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

On the other hand, with the derivation mode $max_{\Delta objects}$ both rules yield the same increase of one object, i.e.,

$$\Delta obj(caa, \{ca \to cdd\}) = \Delta obj(caa, \{a \to bb\}) = 1,$$

which yields all two non-extendable multisets of rules $\{ca \to cdd, a \to bb\}$ and $\{a \to bb, a \to bb\}$ to be applicable to the underlying configuration caa, i.e.,

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

2.4 Computations in a P system

The P system continues with applying multisets of rules according to the derivation mode until there remain no applicable rules in the single region of Π, i.e., as usual, with all these variants of derivation modes as defined above, we consider *halting computations*.

We may generate or accept or even compute functions or relations. The inputs/outputs may be multisets or strings, defined in the well-known way. When the system *halts*, in case of computing with multisets we consider the number of objects from T contained in the membrane region at the moment when the system halts as the *result* of the underlying computation of Π.

We would like to emphasize that as results we only take the objects from the terminal alphabet T, especially the catalyst is not counted to the result of a computation. On the other hand, with all the proofs given in this paper, except for the single catalyst no other garbage remains in the membrane region at the end of a halting computation.

As already mentioned earlier, the following result was shown in [30], establishing a lower bound for the computational power of catalytic P systems with only one catalyst:

Proposition 4 *Catalytic P systems with only one catalyst working in the derivation mode max have at least the computational power of partially blind register machines.*

Example 3 In [30] it was shown that the vector set

$$S = \{(n, m) \mid 0 \le n, n \le m \le 2^n\}$$

(which is not semi-linear) can be generated by a P system working in the derivation mode *max* with only one catalyst and 19 rules.

2.5 Two variants of energy control

We now recall the definitions from [4] introducing two variants of energy-controlled P systems. Yet in contrast to the general definitions given in [4] again we will restrict ourselves to simple P systems $\Pi = (V, \{c\}, T, w, \mathcal{R})$.

The first variant assigns fixed integer values of energy to each symbol in the system, i.e., instead of V we consider the set V_E consisting of pairs $[x, f(x)]$ with $x \in V$ and $f : V \to \mathbb{Z}$ being a function assigning a unique energy value to each symbol in V. We extend f in the natural way to strings or multisets over V. The energy balance of a rule $u \to v$ then is $f(v) - f(u)$. Such variants of P systems will be called *symbol energy-controlled* P systems.

In the second variant, the energy is directly assigned to the rules only. Such variants of P systems will be called *rule energy-controlled* P systems.

3 Computational completeness for simple P systems working in the derivation modes $max_{GENobjects}max$ and $max_{\Delta objects}max$

As already mentioned earlier, in [9] we have already shown that we can obtain computational completeness with simple P systems and only one catalyst when using the derivation mode $max_{objects}$ instead of max.

In this section we now study simple P systems with only one catalyst using the derivation modes $max_{GENobjects}max$ and $max_{\Delta objects}max$, respectively.

Theorem 1 *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 $\Pi = (V, \{c\}, T, w, \mathcal{R})$ 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 Propositions 1, 2, and 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. 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 our construction is that all the objects except the catalyst c and the output objects (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 objects 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 our 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 elaborated in Remark 1: 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, i.e., as in Remark 1, 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 $\Pi = (V, \{c\}, T, c(l_0, 1), \mathcal{R})$.

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

The construction includes the dummy object 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 $\mathrm{Gen}(C, R)$.

The objects a_r, $n + 1 \leq r \leq m$, represent the output registers. For the decrementable registers, we use the objects (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 an object can be used to decrement register r is when $i = r$, i.e., in the r-th step of the simulation cycle.

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

In the same way as the register objects a_r, the program objects (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 $p : (ADD(r), q, s)$:

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

The catalyst has to be used with the program object which otherwise would stay idle when the catalyst is used with a register object, 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 object for the catalyst, we obtain $\Delta obj(C, R) = 1$ because of the additional dummy object 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 object; in fact, if N is the total number of register objects for decrementable registers in the underlying configuration C, then with applying the set of rules R described so far we get $\mathrm{Gen}(C,R)= N+3$ in contrast to $\mathrm{Gen}(C,R')= N-1+3 = N+2$ 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:

$$c(p,n) \to c(q,1)(a_r,1), \quad c(p,n) \to c(s,1)(a_r,1). \tag{3}$$

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

$$c(p,n) \to c(q,1)a_r, \quad c(p,n) \to c(s,1)a_r. \tag{4}$$

As in both cases, together with the program object a new register object 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 object; again we have $\mathrm{Gen}(C,R)= N+3$ with this multiset of rules R in contrast to $\mathrm{Gen}(C,R')= N-1+3 = N+2$ 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 $p : (SUB(r),q,s)$:

$$c(p,i) \to c(p,i+1)d, \quad 1 \le i < r. \tag{5}$$

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

$$(p,r) \to (p,r+1), \quad c(a_r,r) \to ced. \tag{6}$$

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 objects 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 object (a_r,r), i.e., $\mathrm{Gen}(C,R)= N-1+3 = N+2$ in contrast to $\mathrm{Gen}(C,R')= N$.

If $r < n-1$:

$$\begin{aligned} &ce \to cdddd, && (p,r+1) \to (p,r+2)^-; \\ &c(p,r+1) \to c(p,r+2)^0 dd. \end{aligned} \tag{7}$$

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 object $(p,r+1)$ 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 objects evolve in the usual way; in total we get $\Delta obj(C,R)= 3$ and $\mathrm{Gen}(C,R)= N+6$;
2. the catalyst c takes the program object $(p,r+1)$ using the rule $c(p,r+1) \to c(p,r+2)^0 dd$, and all register objects evolve in the usual way; in total we get $\Delta obj(C,R)= 2$ and $\mathrm{Gen}(C,R)= N+4$;
3. the catalyst c takes a register object, the program object $(p,r+1)$ 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 $\mathrm{Gen}(C,R)= (N-1+3)+1 = N+3$.

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

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

$$\begin{aligned} &c(p,i)^- \to c(p,i+1)^- d, \quad r+2 \le i < n, \quad c(p,n)^- \to c(q,1)d, \\ &c(p,i)^0 \to c(p,i+1)^0 d, \quad r+2 \le i < n, \quad c(p,n)^0 \to c(s,1)d. \end{aligned} \tag{8}$$

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

If $r = n-1$:

$$ce \to cdddd,\ (p,n) \to (q,1),\quad c(p,n) \to c(s,1)dd. \tag{9}$$

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

If $r = n$:

$$ce \to cdddd,\quad (p,n+1) \to (q,2),\quad c(p,n+1) \to c(s,2)dd. \tag{10}$$

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 $l_h : HALT$. In an easy way, we can do this by introducing d instead of $(l_h, 1)$ or $(l_h, 2)$ as done for other labels. In this way, finally no program object 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 objects 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. □

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 theorem above we obtain the following result:

Corollary 1 *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 $max_{\Delta objects}max$ or in the derivation mode $max_{GENobjects}max$ 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 derivation mode $max_{\Delta objects}max$ or in the derivation mode $max_{GENobjects}max$ can generate any recursively enumerable set of (vectors of) natural numbers.*

For accepting register machines, in addition to the two working registers, we have at least one input register and therefore immediately infer the following result:

Corollary 2 *For any recursively enumerable set of d-vectors of natural numbers given by a register machine with $d+2$ decrementable registers we can construct an accepting simple P system with only one catalyst and 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 $d+2$ steps, and therefore such catalytic P systems with only one catalyst and working in the derivation mode $max_{\Delta objects}max$ or in the derivation mode $max_{GENobjects}max$ can accept any recursively enumerable set of (vectors of) natural numbers.*

In a similar way, assuming at least one input register to be present, we also infer a similar result for register machines computing partial recursive relations on natural numbers and therefore computational completeness in its widest sense:

Corollary 3 *For any partial recursive relation $f : \mathbb{N}^d \longrightarrow \mathbb{N}^k$ on vectors of natural numbers (i.e., with d components as input and k components as output) given by a register machine with $d+2$ decrementable registers we can construct a simple P system with only one catalyst and 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 $d+2$ steps, and therefore such catalytic P systems with only one catalyst and working in the derivation mode $max_{\Delta objects}max$ or in the derivation mode $max_{GENobjects}max$ can compute any partial recursive relation $f : \mathbb{N}^d \longrightarrow \mathbb{N}^k$ on vectors of natural numbers.*

4 Computational completeness for simple P systems working in the derivation modes $max_{GENobjects}$ and $max_{\Delta objects}$

In this section we show the rather astonishing results that simple P systems with only one catalyst using the derivation modes $max_{GENobjects}$ and $max_{\Delta objects}$ are computationally complete for themselves without a priori needing the condition that the applied multisets of rules are non-extendable, which condition can be mimicked using a suitable additional number of dummy objects on the right-hand side of the rules.

Theorem 2 *For any register machine with $n \geq 2$ 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.*

Proof We use a similar construction as in the proof of Theorem 1. To mimick the non-extendability of the multisets of rules to be applied, we simply add one more dummy object

d on the right-hand side of every rule constructed in that proof, except for the erasing rule $d \to \lambda$.

For example, the cycling rules for the register objects now are as follows:

$$(a_r, i) \to (a_r, i+1)d, 1 \le r < n; \quad (a_r, n) \to (a_r, 1)d. \tag{11}$$

No register object can stay idle, as the application of a rule given above increases $\Delta obj(C,R)$ by 1 and the number of generated objects by 2.

All the other arguments for $\Delta obj(C,R)$ and $\mathrm{Gen}(C,R)$ as explained in in the proof of Theorem 1 can be taken over as they are given there, with re-calculating the values for $\Delta obj(C,R)$ and for $\mathrm{Gen}(C,R)$, the number of generated objects, always adding 1 for every rule which is applied.

For example, we re-investigate the possible variants for the simulation of a *SUB*-instruction on register r, for $r < n-1$; in the following, N is the total number of register objects for decrementable registers in the underlying configuration C:

We start with

$$(p, r) \to (p, r+1)d, \quad c(a_r, r) \to cedd. \tag{12}$$

From the list of rules in Eq. (7) we obtain the following list of rules:

$$\begin{array}{ll} ce \to cd^5, & (p, r+1) \to (p, r+2)^- d; \\ c(p, r+1) \to c(p, r+2)^0 d^3. & \end{array} \tag{13}$$

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 $ce \to cd^5$, and to the program object $(p, r+1)$ the rule $(p, r+1) \to (p, r+2)^- d$ must be applied due to the fact that both derivation modes $max_{\Delta objects}$ and $max_{GENobjects}$ mimick the non-extendability of the applied multiset of rules by the additional object d on the right-hand side of the rules, i.e., every object which can evolve must evolve; all register objects evolve in the usual way; in total we get $\Delta obj(C,R) = N+5$ and $\mathrm{Gen}(C,R) = 2N+8$;
2. the catalyst c takes the program object $(p, r+1)$ using the rule $c(p, r+1) \to c(p, r+2)^0 d^3$, and all register objects evolve in the usual way; in total we get $\Delta obj(C,R) = N+3$ and $\mathrm{Gen}(C,R) = 2N+5$;
3. the catalyst c takes a register object, the program object $(p, r+1)$ evolves with the rule $(p, r+1) \to (p, r+2)^- d$, and all other register objects evolve in the usual way; in total we get $\Delta obj(C,R) = (N-1)+2+1 = N+2$ and $\mathrm{Gen}(C,R) = 2(N-1)+4+2 = 2N+4$.

In total, only variant 1 fulfills the condition given by the derivation mode $max_{\Delta objects}$ 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 $max_{GENobjects}$ with respect to the number of generated objects, $\mathrm{Gen}(C,R)$.

In case register r is empty, we have the following possibilities:

1. catalyst c takes the program object $(p, r+1)$ using the rule $c(p, r+1) \to c(p, r+2)^0 ddd$, and all register objects evolve in the usual way; in total we get $\Delta obj(C,R) = N+3$ and $\mathrm{Gen}(C,R) = 2N+5$;
2. the catalyst c takes a register object $(a_{r+1}, r+1)$ thereby generating edd, the program object $(p, r+1)$ evolves with the rule $(p, r+1) \to (p, r+2)^- d$, and all other register objects evolve in the usual way; this variant leads to $\Delta obj(C,R) = N-1+2+1 = N+2$ and $\mathrm{Gen}(C,R) = (2(N-1)+4)+2 = 2N+4$.

In case of the derivation mode $max_{\Delta objects}$, the erasing rule $d \to \lambda$ can never be applied together with the other rules, as it would diminish $\Delta obj(C,R)$. Hence, the garbage of objects d remains until the simulated register machine has halted, and then the dummy objects d are eliminated in a sequential way.

On the other hand, in case of the derivation mode $max_{GENobjects}$ the value of $\mathrm{Gen}(C,R)$ does not change with the application of any number of rules $d \to \lambda$, as $\mathrm{Gen}(C, \{d \to \lambda\}) = 0$. Remaining dummy objects d finally are eliminated in an asynchronous way.

We leave all the remaining technical details to the interested reader. □

5 Connection to energy-controlled P systems

Adapting the proofs elaborated in the preceding section we can immediately show equivalent results for energy-controlled P systems using the derivation mode $max_{energy}max$ instead of the derivation modes $max_{objects}max$.

Theorem 3 *For any register machine with at least two decrementable registers we can construct a simple symbol energy-controlled P system with only one catalyst and working in the derivation mode $max_{energy}max$ which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

Proof We can verbatim take over the proof given for Theorem 2 by taking V_E instead of V consisting of pairs $[x, 1]$

assigning the unique energy value 1 to each symbol in V. In that way, all arguments referring to the number of objects in a (multiset of) rule(s) can immediately be taken over as analog arguments referring to the amount of energy represented by the objects.

As an example, let us consider the rule $ce \to cdddd$ which yields $\Delta objects = 5 - 2 = 3$ for the difference of the number of objects and as well $\Delta energy = 3$ for the difference of energy.

On the other hand, for the rule $d \to \lambda$ we get $\Delta objects = -1$ for the difference of the number of objects and as well $\Delta energy = -1$. □

Corollary 4 *For any register machine with at least two decrementable registers we can construct a simple rule energy-controlled P system with only one catalyst and working in the derivation mode $max_{energy}max$ which can simulate every step of the register machine in n steps where n is the number of decrementable registers.*

Proof Again we take over the proof given for Theorem 2 and now define the energy assigned to a rule as the difference of the number of objects on the left-hand side and the number of objects on the right-hand side of a rule.

Like in the proof of Theorem 3, as an example, let us again consider the rule $ce \to cdddd$ which yields $\Delta objects = 3$ for the difference of the number of objects and thus the energy 3 for the energy to be assigned to this rule. As in [4], we may write

$$ce \to cdddd \ < 3 >$$

to specify that the rule $ce \to cdddd$ has energy 3.

On the other hand, for the rule $d \to \lambda$ we get $\Delta objects = -1$ for the difference of the number of objects and therefore:

$$d \to \lambda \ < -1 >$$

The arguments for the sum of energies assigned to the rules in the applied multiset of rules now run as for the number of objects.

Yet now in the case of rule energy-controlled P systems we can even avoid the use of the dummy object d: we simply may omit the object in every rule obtained in that way, but still keep the original amount of energy assigned to the rule; for example, instead of

$$ce \to cdddd \ < 3 >$$

we now may take

$$ce \to c \ < 3 >$$

although the difference of objects now would be -1.

Finally, of course, we have to eliminate the rule

$$d \to \lambda \ < -1 >$$

as it is not needed any more. □

6 Conclusion

In this paper we have continued our research on revisiting a classic problem of computational complexity in membrane computing: can catalytic P systems with only one catalyst already generate all recursively enumerable sets of multisets? This problem has been standing tall for many years, and nobody has yet managed to give it a positive or a negative answer. Already in [6] and in [9] we could show that adding the ingredient of weak priority of catalytic rules over non-catalytic rules or only taking the derivation mode $max_{objects}$ we can obtain computational completeness with only one catalyst.

In this paper, we have added some similar results how to obtain computational completeness with only one catalyst when using one of the newly defined derivation modes $max_{\Delta objects}max$ and $max_{GENobjects}max$ or $max_{\Delta objects}$ and $max_{GENobjects}$, respectively. We especially emphasize that computational completeness for $max_{\Delta objects}$ and $max_{GENobjects}$ can be obtained without requiring the chosen multisets of rules to be non-extendable.

Moreover, based on the results for simple P systems with only one catalyst using the derivation modes $max_{\Delta objects}max$, we could show the corresponding computational completeness for simple energy-controlled P systems working in the corresponding derivation mode $max_{energy}max$.

The results obtained in this paper can also be extended to P systems dealing with strings. Following the definitions and notions used in [30], computational completeness for computing with strings can be shown.

Acknowledgements The authors gratefully thank the referees for their useful comments. Rudolf Freund acknowledges the TU Wien supporting the open access publishing of this paper.

Funding Open access funding provided by TU Wien (TUW). Artiom Alhayov acknowledges project 20.80009.5007.22 "Intelligent information systems for solving ill-structured problems, processing knowledge and big data" by the National Agency for Research and Development.

Sergiu Ivanov is partially supported by the Paris region via the project DIM RFSI n°2018-03 "Modèles informatiques pour la reprogrammation cellulaire".

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Alhazov, A., Aman, B., Freund, R. (2014). P systems with anti-matter. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (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 https://doi.org/10.1007/978-3-319-14370-5_5
2. Alhazov, A., Aman, B., Freund, R., Păun, Gh. (2014). Matter and anti-matter in membrane systems. In: H. Jürgensen, J. Karhumäki, A. Okhotin (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 https://doi.org/10.1007/978-3-319-09704-6_7
3. Alhazov, A., Freund, R. (2014). P systems with toxic objects. In: M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, C. Zandron (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 https://doi.org/10.1007/978-3-319-14370-5_7
4. Alhazov, A., Freund, R., Ivanov, S. (2016). Variants of energy-controlled P systems. In: Proceedings of NIT 2016
5. Alhazov, A., Freund, R., & Ivanov, S. (2019). Variants of P systems with activation and blocking of rules. *Nat. Comput., 18*(3), 593–608. https://doi.org/10.1007/s11047-019-09747-5
6. Alhazov, A., Freund, R., Ivanov, S. (2020). Catalytic P systems with weak priority of catalytic rules. In: R. Freund (ed.) Proceedings ICMC 2020, September 14–18, 2020, pp. 67–82. TU Wien
7. Alhazov, A., Freund, R., Ivanov, S. (2020). P systems with limiting the number of objects in membranes. In: R. Freund (ed.) Proceedings ICMC 2020, September 14–18, 2020, pp. 83–98. TU Wien
8. Alhazov, A., Freund, R., & Ivanov, S. (2021). P systems with limited number of objects. *Journal of Membrane Computing, 3*, 1–9. https://doi.org/10.1007/s41965-020-00068-6
9. Alhazov, A., Freund, R., & Ivanov, S. (2021). When catalytic P systems with one catalyst can be computationally complete. *Journal of Membrane Computing*. https://doi.org/10.1007/s41965-021-00079-x
10. Alhazov, A., Freund, R., Ivanov, S., Verlan, S. (2017). (Tissue) P systems with vesicles of multisets. In: E. Csuhaj-Varjú, P. Dömösi, Gy. Vaszil (eds.) Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4–6, 2017, *EPTCS*, vol. 252, pp. 11–25. https://doi.org/10.4204/EPTCS.252.6
11. Alhazov, A., Freund, R., Leporati, A., Oswald, M., & Zandron, C. (2006). (Tissue) P systems with unit rules and energy assigned to membranes. *Fundam. Informaticae, 74*(4), 391–408.
12. Alhazov, A., Freund, R., Oswald, M., & Verlan, S. (2009). Partial halting and minimal parallelism based on arbitrary rule partitions. *Fundam. Inform., 91*(1), 17–34. https://doi.org/10.3233/FI-2009-0031
13. Alhazov, A., Freund, R., & Sosík, P. (2015). Small P systems with catalysts or anti-matter simulating generalized register machines and generalized counter automata. *Comput. Sci. J. Moldova, 23*(3), 304–328.
14. Alhazov, A., Freund, R., & Verlan, S. (2017). P systems working in maximal variants of the set derivation mode. In: A. Leporati, G. Rozenberg, A. Salomaa, C. Zandron (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 https://doi.org/10.1007/978-3-319-54072-6_6
15. Ciobanu, G., Marcus, S., & Păun, Gh. (2009). New strategies of using the rules of a P system in a maximal way. power and complexity. Romanian Journal of Information Science and Technology **12**(2), 21–37
16. Dassow, J., & Păun, Gh. (1989). Regulated Rewriting in Formal Language Theory. Springer
17. Freund, R. (2003). Energy-controlled P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (eds.) Membrane Computing, pp. 247–260. Springer
18. Freund, R. (2013). Purely catalytic P systems: two catalysts can be sufficient for computational completeness. In A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin (Eds.), *CMC14 Proceedings 14th International Conference on Membrane Computing, Chişinău, August 20–23, 2013* (pp. 153–166). Academy of Sciences of Moldova: Institute of Mathematics and Computer Science.
19. Freund, R. (2016). P automata: New ideas and results. In: H. Bordihn, R. Freund, B. Nagy, Gy. Vaszil (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
20. Freund, R. (2020). How derivation modes and halting conditions may influence the computational power of P systems. *Journal of Membrane Computing, 2*(1), 14–25. https://doi.org/10.1007/s41965-019-00028-9
21. Freund, R., Kari, L., Oswald, M., & Sosík, P. (2005). Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science, 330*(2), 251–266. https://doi.org/10.1016/j.tcs.2004.06.029.
22. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., Zandron, C. (2014). Flattening in (tissue) P systems. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, A. Salomaa (eds.) Membrane Computing, *Lecture Notes in Computer Science*, vol. 8340, pp. 173–188. Springer https://doi.org/10.1007/978-3-642-54239-8_13
23. Freund, R., & Oswald, M. (2007). Partial halting in P systems. *Int. J. Found. Comput. Sci., 18*(6), 1215–1225. https://doi.org/10.1142/S0129054107005261
24. Freund, R., Oswald, M. (2013). Catalytic and purely catalytic P automata: Control mechanisms for obtaining computational completeness. In: S. Bensch, F. Drewes, R. Freund, F. Otto (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
25. Freund, R., Oswald, M. (2017). Variants of spiking neural P systems with energy control. In: Proceedings of ICAROB 2017

26. Freund, R., Oswald, M., & Păun, Gh. (2015). Catalytic and purely catalytic P systems and P automata: control mechanisms for obtaining computational completeness. *Fundam. Inform., 136*(1–2), 59–84. https://doi.org/10.3233/FI-2015-1144.
27. Freund, R., Păun, Gh. (2013). How to obtain computational completeness in P systems with one catalyst. In: T. Neary, M. Cook (eds.) Proceedings Machines, Computations and Universality 2013, MCU 2013, Zürich, Switzerland, September 9–11, 2013. *EPTCS*, vol. 128, pp. 47–61 https://doi.org/10.4204/EPTCS.128.13
28. Freund, R., Păun, Gh., Pérez-Jiménez, M.J. (2007). Polarizationless P systems with active membranes working in the minimally parallel mode. In: S.G. Akl, C.S. Calude, M.J. Dinneen, G. Rozenberg, T. Wareham (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 https://doi.org/10.1007/978-3-540-73554-0_8
29. Freund, R., Rogozhin, Yu., Verlan, S. (2012). P systems with minimal left and right insertion and deletion. In: J. Durand-Lose, N. Jonoska (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 https://doi.org/10.1007/978-3-642-32894-7_9
30. Freund, R., Sosík, P. (2015). On the power of catalytic P systems with one catalyst. In: G. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron (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 https://doi.org/10.1007/978-3-319-28475-0_10
31. Freund, R., Verlan, S. (2007). A formal framework for static (tissue) P systems. In: G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa (eds.) Membrane Computing, *Lecture Notes in Computer Science*, vol. 4860, pp. 271–284. Springer https://doi.org/10.1007/978-3-540-77312-2_17
32. Freund, R., & Verlan, S. (2011). (Tissue) P systems working in the k-restricted minimally or maximally parallel transition mode. *Nat. Comput., 10*(2), 821–833. https://doi.org/10.1007/s11047-010-9215-z
33. Krithivasan, K., Păun, Gh., & Ramanujan, A. (2014). On controlled P systems. Fundam. Inform. **131**(3–4), 451–464 https://doi.org/10.3233/FI-2014-1025
34. Minsky, M. L. (1967). *Computation*. Englewood Cliffs: Finite and Infinite Machines. Prentice Hall.
35. Păun, Gh. (2000). Computing with membranes. *Journal of Computer and System Sciences, 61*(1), 108–143. https://doi.org/10.1006/jcss.1999.1693
36. Păun, Gh. (2002). *Membrane Computing: An Introduction. Springer*. https://doi.org/10.1007/978-3-642-56196-2
37. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.) (2010). The Oxford Handbook of Membrane Computing. Oxford University Press
38. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer (1997). https://doi.org/10.1007/978-3-642-59136-5
39. The P Systems Website. http://ppage.psystems.eu/