Theoretical Computer Science 920 (2022) 95–112

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Variants of derivation modes for which purely catalytic P systems are computationally complete

Artiom Alhazov [a], Rudolf Freund [b,*], Sergiu Ivanov [c], Marion Oswald [d]

[a] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Str. Academiei 5, Chişinău, MD 2028, Republic of Moldova
[b] TU Wien, A-1040 Wien, Austria
[c] IBISC, Univ. Évry, Paris-Saclay University, 23, boulevard de France, 91034 Évry, France
[d] Institute of Logic and Computation, Faculty of Informatics, TU Wien, Favoritenstraße 9–11, A-1040 Wien, Austria

ARTICLE INFO

Article history:
Received 27 August 2021
Received in revised form 15 February 2022
Accepted 8 March 2022
Available online 21 March 2022
Communicated by N. Jonoska

Keywords:
Catalysts
Computational completeness
Derivation modes
Purely catalytic P systems

ABSTRACT

Catalytic P systems and purely catalytic P systems are among the first variants of membrane systems ever considered in this area. These variants of systems also feature some prominent computational complexity questions, and in particular the problem if only one catalyst in catalytic P systems and two catalysts in purely catalytic P systems are enough to allow for generating all recursively enumerable sets of multisets. Several additional ingredients have been shown to be sufficient for obtaining such results.

Previously, we could show that using 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 catalytic P systems. In this paper we investigate the question whether we can obtain a similar result for purely catalytic P systems, i.e., we show that two catalysts in purely catalytic P systems are enough to allow for generating all recursively enumerable sets of multisets when using specific variants of the maximally parallel derivation mode: we take only those applicable multisets of rules which (i) generate the maximal number of objects, or (ii) yield the maximal difference in the number of objects between the newly generated configuration and the current configuration.

In addition, we also consider non-extendable multisets of rules which (i) generate the minimal number of objects, or (ii) yield the minimal difference in the number of objects between the newly generated configuration and the current configuration.

In all cases, we have also shown that register machines with n decrementable registers can be simulated by simple purely catalytic P systems working in any of these derivation modes using only n catalysts. Hence, simple purely catalytic P systems working in any of these derivation modes are computationally complete.

1. Introduction

Membrane systems were introduced more than two decades ago, see [29], 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 [30] and [31]. For actual information see the P sys-

* Corresponding author.
E-mail addresses: artiom@math.md (A. Alhazov), rudi@emcc.at (R. Freund), sergiu.ivanov@ibisc.univ-evry.fr (S. Ivanov), marion@emcc.at (M. Oswald).

https://doi.org/10.1016/j.tcs.2022.03.007

tems webpage [33] 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 [29] 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 objects 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. On the other hand, non-catalytic rules in catalytic P systems are non-cooperative rules of the form $a \rightarrow v$. In this paper, we focus on purely catalytic P systems, which use catalytic rules only.

From the beginning (see [29] and [30]), the question how many catalysts are needed for obtaining computational completeness has been one of the most intriguing challenges regarding catalytic and purely catalytic P systems. In [19] it has already been shown that for (purely) catalytic P systems two (three) catalysts are enough for generating any 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 (see [30]), 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 [16,22–24], using various additional ingredients, i.e., additional control mechanisms, one (two) catalyst(s) can be sufficient for (purely) catalytic P systems: In P systems with label selection or controlled P systems (as introduced in [27]), 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, the concept of anti-matter, where an object is annihilated by the corresponding anti-matter object, is investigated in [1,2,12]. Using sets of rules in variants of the maximally parallel derivation mode usually yields similar results as when using multisets of rules, but when all rules in a membrane have to agree in the same target for the results, even without catalysts computational completeness can be obtained, see [13]. An overview of results obtained by using control mechanisms as mentioned above for P automata is given in [17].

The influence of using partial halting, i.e., of stopping a computation already when a part of the system cannot evolve any more, is investigated in [11,21]. Descriptional complexity especially with respect to the number of rules in most cases of P systems working in different variants of the maximally parallel derivation mode can be reduced by using the concept of toxic objects, see [3], whose appearance immediately discards a computation. For an overview on specific results when different derivation modes and halting conditions are used, the reader is referred to [18].

More recently, in a quite similar way as in graph-controlled systems, controlling the application of rules activating and blocking rules for the next steps based on the rules applied in a derivation step has been investigated in [5]. Limiting the number of specific objects obtained by the application of rules allows for reaching computational completeness even without catalysts, see [7,8]. On the other hand, one catalyst is needed for catalytic P systems when giving catalytic rules weak priority over non-cooperative rules, as shown in [6]. In [9], the additional computational completeness result for P systems with only one catalyst working in the maximally parallel derivation mode and affecting the maximal number of objects in the underlying configuration was elaborated.

Assigning energy values to objects or rules allows for controlling which rules have to be applied, see [4,15]. In a similar way, in [9] computational completeness has been shown for catalytic P systems with one catalyst where we 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.

In this paper we now continue the research started for catalytic P systems in [10] and continued in [9]: we investigate the following variants of the maximally parallel derivation mode for purely catalytic P systems: we take only those applicable multisets of rules which

1. generate the maximal number of objects, or
2. yield the maximal difference in the number of objects between the newly generated configuration and the current configuration or
3. generate the minimal number of objects, or
4. yield the minimal difference in the number of objects between the newly generated configuration and the current configuration.

For the variants with the maximal number of objects we can take those multisets of rules from the applicable multisets of rules which are non-extendable, but we can also take those sets with the maximal number of generated or maximal difference of objects without requesting the multisets of rules to fulfill the condition to be non-extendable.

For the variants with the minimal number of objects we only take the non-extendable applicable multisets of rules with the minimal number of generated or minimal difference of objects.

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 $0 \leq m < n$ and $n \oplus_n +1 = 1$.

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 [14] and [32].

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

$$M = (m, B, l_0, l_h, P) \text{ where}$$

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

Throughout the paper, B_{ADD} denotes the set of labels of *ADD*-instructions $p : (ADD(r), q, s)$ of arbitrary registers r, and $B_{ADD(r)}$ denotes the set of labels of all *ADD*-instructions $p : (ADD(r), q, s)$ of the specific register r. In the same way, B_{SUB} denotes the set of labels of all *SUB*-instructions $p : (SUB(r), q, s)$ of arbitrary decrementable registers r, and $B_{SUB(r)}$ denotes the set of labels of all *SUB*-instructions $p : (SUB(r), q, s)$ of the specific 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.

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.

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 [28]; 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 purely catalytic 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 [20], in this paper we only consider simple purely catalytic P systems, i.e., with the simplest membrane structure of only one membrane:

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

$$\Pi = (V, C, T, w, \mathcal{R}) \text{ where}$$

- 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;
- $\mathcal{R}$ is a finite set of *evolution rules* over V; these evolution rules are catalytic rules of the forms $ca \to cv$, 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 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 [26] 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 $\Delta obj(C, R)$.

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:

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

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 [10], 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:

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

As in purely catalytic P system we only have catalytic rules, which on the left-hand side of the rule have exactly two objects, in both cases we only have to consider the right-hand sides of the rules when comparing rules.

We now also consider the new idea of the minimal number of generated objects and the minimal difference between generated and consumed objects.

$min_{GENobjects}max$ 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 minimal:

$$\begin{aligned} Appl(\Pi, C, min_{GENobjects}max) = \{R \in Appl(\Pi, C, max) \mid \\ \text{there is no } R' \in Appl(\Pi, C, max) \\ \text{such that } \mathrm{Gen}(C, R) > \mathrm{Gen}(C, R')\}. \end{aligned}$$

$min_{\Delta objects}max$ a non-extendable multiset of rules R applicable to the current configuration C is only taken if the difference of $|C'| - |C|$ is minimal:

$$\begin{aligned} Appl(\Pi, C, min_{\Delta objects}max) = \{R \in Appl(\Pi, C, max) \mid \\ \text{there is no } R' \in Appl(\Pi, C, max) \\ \text{such that } \Delta obj(C, R) > \Delta obj(C, R')\}. \end{aligned}$$

As for the corresponding maximal variants, in both cases as defined above for the minimal numbers again we only have to consider the right-hand sides of the rules when comparing rules, because in purely catalytic P system we only have catalytic rules, which on the left-hand side of the rule have exactly two objects.

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:

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

$max_{\Delta objects}$ 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:

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

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

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

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 $c_1c_2aa \xrightarrow{\{c_2a\to c_2bb\}} c_1c_2abb$.

In contrast, when taking the minimal derivation modes $min_{\Delta objects}max$ and $min_{GENobjects}max$, we only take non-extendable multisets of rules, i.e., both catalysts have to be used in the computation step; yet now for catalyst c_2 instead of the rule $c_2a \to c_2bb$ we have to take the rule $c_2a \to c_2b$. Hence, the only possible derivation with the minimal derivation modes is:

- for $min_{GENobjects}max$: $c_1c_2aa \xrightarrow{\{c_1a\to c_1, c_2a\to c_2b\}} c_1c_2b$;
- for $min_{\Delta objects}max$: $c_1c_2aa \xrightarrow{\{c_1a\to c_1, c_2a\to c_2b\}} c_1c_2b$.

Remark 2. Without requiring the non-extendability of the multisets to be applied, the conditions $min_{GENobjects}$ and $min_{\Delta objects}$ for themselves alone seem to be not very useful. In both cases, shrinking rules like $ce \to c$ have to be used in an extensive way, probably together with unit rules like $ca \to cb$ in the case of $min_{\Delta objects}$. If only growing rules like $ca \to cbd$ are applicable, both modes work like the sequential mode with only one rule having the chance to be applied. Hence, these derivation modes are not considered further in this paper.

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 catalysts are not counted to the result of a computation. On the other hand, with all the proofs given in this paper, except for the catalysts no other "garbage" remains in the membrane region at the end of a halting computation, i.e., we could even omit T.

3. Simple purely catalytic P systems working in the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, or $max_{GENobjects}$

In this section we show how the new derivation modes allow for simulating register machines by purely catalytic P systems with one catalyst less than in the original proofs given in [19], which are the first results obtained for purely catalytic P systems of that kind when using specific variants of derivation modes themselves without in addition using specific control mechanisms as, for example, in [23].

Theorem 4. *For any register machine with two decrementable registers we can construct a simple purely catalytic P system with only two 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. As in purely catalytic P systems we only have a bounded number of rules – bounded by the number of catalysts – which can be executed in one derivation step, we cannot apply the proof idea used in the proofs elaborated in [10], where in some sense 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 was mimicked when using the derivation mode $max_{objects}$, see [9].

Instead, we have to go back to the original construction as elaborated in [19], yet we also take into consideration ideas related to energy-controlled P systems as described in [4], using dummy objects like energy to control the correct application of rules. Now having the new variants of derivation modes, we can avoid the trap rules and even more, instead of three catalysts, we only need two catalysts to obtain the desired completeness result.

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

$$\Pi = (V, \{c_1, c_2\}, T, c_1c_2l_0l_0', \mathcal{R})$$

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 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 is elaborated for all the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, and $max_{GENobjects}$; only a few subtle technical details have 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 both catalysts are kept busy to guarantee that the simulation is executed in a correct way. The extensive use of the dummy object d guarantees that one of the rules using the catalysts c_1 and c_2 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. On the other hand, 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.

The only special detail which arises is that in the derivation mode $max_{\Delta objects}$, where, as in Example 1, the rules

$$c_1d \to c_1 \text{ and } c_2d \to c_2$$

erasing the "energy" object d can only be used at the end of a computation when no other rules can be applied any more.

Before giving the whole construction of the simple purely catalytic P system, we mention that the main basis for choosing the right number of the objects d on the right-hand side of the rules is based on the importance and role of the rules

$$c_1a_1 \to c_1\hat{a}_1dd \text{ and } c_2a_2 \to c_2\hat{a}_2dd$$

which should only be applicable in the first step of the simulation of a decrement instruction on register 1 and register 2, respectively. As usually done in corresponding proofs, 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 $r > 2$ are never changed again, as they represent output registers.

$$\begin{aligned} V &= \{a_r \mid 1 \le r \le m\} \cup \{\hat{a}_r \mid 1 \le r \le 2\} \\ &\cup \{p, p' \mid p \in B_{ADD} \cup \{l_h\}\} \\ &\cup \{p, p', \bar{p}, \hat{p} \mid p \in B_{SUB(r)}, 1 \le r \le 2\} \\ &\cup \{c_1, c_2, d\}, \\ T &= \{a_r \mid 3 \le r \le m\}. \end{aligned}$$

B_{ADD} and $B_{SUB(r)}$ denote the set of labels of ADD- and SUB-instructions of the register machine M.

The set $\mathcal{R}$ 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 $p \in B_{ADD}$, $q, s \in B$, $1 \le r \le m$.

Table 1
Simulation of *SUB*-instruction.

step	$\|reg(r)\|$	rule for c_r	rule for c_{3-r}	resulting objects
first	>0	$c_r a_r \to c_r \hat{a}_r d^2$	$c_{3-r} p' \to c_{3-r} \bar{p} d^{10}$	$p, \bar{p}, \hat{a}_r$
	$=0$	$c_r p \to c_r d^2$	$c_{3-r} p' \to c_{3-r} \bar{p} d^{10}$	$\bar{p}$
second	>0	$c_r \bar{p} \to c_r \hat{p} d^3$	$c_{3-r} p \to c_{3-r} d^9$	$\hat{a}_r, \hat{p}$
	$=0$	$c_r d \to c_r$ (*)	$c_{3-r} \bar{p} \to c_{3-r} s s' d^6$	s, s'
third	>0	$c_r \hat{p} \to c_1 q q' d^2$	$c_{3-r} \hat{a}_r \to c_{3-r} d^4$	q, q'

Table 2
Rules applicable in the first step.

(1)	$c_r a_r \to c_r \hat{a}_r d^2$,
(2)	$c_r p \to c_r d^2$,
(3)	$c_r d \to c_r$,
(4)	$c_{3-r} p' \to c_{3-r} \bar{p} d^{10}$,
(5)	$c_{3-r} a_{3-r} \to c_{3-r} \hat{a}_{3-r} d^2$,
(6)	$c_{3-r} d \to c_{3-r}$,
(7)	$c_{3-r} p \to c_{3-r} d^9$.

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

$$c_1 p \to c_1 q q' a_r d \text{ or } c_1 p \to c_1 s s' a_r d,$$
$$c_2 p' \to c_2 d^4.$$

The dummy objects d are used guarantee that the rules given above, with in sum 5 objects on their right-hand sides, have priority over the rules $c_1 a_1 \to c_1 \hat{a}_1 d^2$ and $c_2 a_2 \to c_2 \hat{a}_2 d^2$, respectively, with in sum only 4 objects on their right-hand sides.

- $p : (SUB(r), q, s)$, with $p \in B_{SUB}$, $q, s \in B$, $1 \le r \le 2$.

  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 1.

  The following table summarizes the rules to be used for the simulation of the SUB-instruction on register r, $r \in \{1, 2\}$, i.e., we use the following rules, resulting in different sets of objects depending on the value of $|reg(r)|$, thereby neglecting the objects d; we emphasize that the simulation is *deterministic*.
  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.

  In the first step, the current configuration contains the program symbols p and p', hence, there are seven rules which possibly can be applied, depending on the contents of registers r and $3-r$, see Table 2.
  We now calculate the number of generated objects and the difference between the number of objects for each possible combination of rules for the catalysts c_r and c_{3-r}. In most cases the difference between sum of generated objects and difference of objects is 4, but when rule (3) and/or (6) is involved and the derivation mode is $max_{\Delta objects}$, then these rules (3) and (6) are never executed at this stage of the derivation, which then yields the number between brackets $[,]$. The combination of rules (2) and (7) is not taken into account, as both rules require the symbol p.
  Which rule catalyst c_r has to use depends on the contents of register r: in case register r is not empty, i.e., a copy of the symbol a_r is present in the current configuration, the rule (1) $c_r a_r \to c_r \hat{a}_r d^2$ has to be used, as it generates 4 objects, whereas the rule (2) $c_r p \to c_r d^2$, to be used in case the register is empty, i.e., if no copy of the symbol a_r is present in the current configuration, only generates 3 objects. In both cases, these rules have to be applied together with rule $c_{3-r} p' \to c_{3-r} \bar{p} d^{10}$, as already described in Table 1. Any other possible combination of rules in all cases would lead to less objects being generated, see Table 3.
  In the second step, in case the register has been empty, only the additional program symbol is present, and it has to be used with the catalyst c_{3-r} using the rule $c_{3-r} \bar{p} \to c_{3-r} s s' d^6$, because it generates 9 objects, whereas if the object $\bar{p}$ were used with the catalyst c_{3-r} and the rule $c_r \bar{p} \to c_r \hat{p} d^3$ together with the rule $c_{3-r} a_{3-r} \to c_{3-r} \hat{a}_{3-r} d^2$, in sum only 9 objects would be obtained. All possibly applicable rules as well as all possible combinations of rules and the

Table 3
Rule combinations applicable in the first step.

rules	generated objects	difference of objects
(1) and (4)	16	12
(1) and (5)	8	4
(1) and (6)	5[4]	3 [2]
(1) and (7)	14	10
(2) and (4)	15	11
(2) and (5)	7	3
(2) and (6)	4	2 [3]
(3) and (4)	13	9 [10]
(3) and (5)	5	1 [2]
(3) and (6)	2[0]	-2[0]
(3) and (7)	11	7 [8]

Table 4
Rules applicable in the second step, $|reg(r)| = 0$.

(3)	$c_r d \to c_r$,
(9)	$c_r \bar{p} \to c_r \hat{p} d^3$,
(5)	$c_{3-r} a_{3-r} \to c_{3-r} \hat{a}_{3-r} d^2$,
(6)	$c_{3-r} d \to c_{3-r}$,
(8)	$c_{3-r} \bar{p} \to c_{3-r} s s' d^6$.

Table 5
Rule combinations applicable in the second step, $|reg(r)| = 0$.

rules	generated objects	difference of objects
(3) and (5)	5	1 [2]
(3) and (6)	2 [0]	-2 [0]
(3) and (8)	10	6 [7]
(9) and (5)	9	5
(9) and (6)	6	2 [3]

Table 6
Rules applicable in the second step, $|reg(r)| > 0$.

(1)	$c_r a_r \to c_r \hat{a}_r d^2$,
(2)	$c_r p \to c_r d^2$,
(3)	$c_r d \to c_r$,
(9)	$c_r \bar{p} \to c_r \hat{p} d^3$,
(5)	$c_{3-r} a_{3-r} \to c_{3-r} \hat{a}_{3-r} d^2$,
(6)	$c_{3-r} d \to c_{3-r}$,
(7)	$c_{3-r} p \to c_{3-r} d^9$,
(8)	$c_{3-r} \bar{p} \to c_{3-r} s s' d^6$,
(10)	$c_{3-r} \hat{a}_r \to c_{3-r} d^4$.

corresponding numbers of generated objects and differences of objects are collected in Tables 4 and 5; observe that the combination of rules (9) and (8) is not possible, as both need the symbol $\bar{p}$.
Now we consider the case when register r has not been empty, i.e., the special symbols p, $\bar{p}$, and $\hat{a}_r$ are present in the current configuration; hence, all the rules as indicated in Table 6 probably are applicable, which yields the possible rule combinations exhibited in Table 7, where for the sake of conciseness, we omit all combinations with rules (3) and (6), as now always rules generating more objects are available for both catalysts in every case.
In the third step, only necessary when the register r has not been empty, only the combination of the rules $c_r \hat{p} \to c_1 q q' d^2$ and $c_{3-r} \hat{a}_r \to c_{3-r} d^4$ yields the maximal number of generated objects, as the application of one or both of the other applicable rules $c_r a_r \to c_r \hat{a}_r d^2$ and $c_{3-r} a_{3-r} \to c_{3-r} \hat{a}_{3-r} d^2$ in any case would yield (at least) one object less.

- $l_h : HALT$.
  When a computation of the register machine M ends with reaching the *HALT*-instruction, the simulating P system Π uses the following rules:

$$c_1 l_h \to c_1 dd \text{ and } c_2 l'_h \to c_2 dd.$$

After the register machine has halted (with the first two 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

$$c_1 d \to c_1 \text{ and } c_2 d \to c_2.$$

Table 7
Rule combinations applicable in the second step, $|reg(r)| > 0$.

rules	generated objects	difference of objects
(1) and (5)	8	4
(1) and (7)	14	10
(1) and (8)	13	9
(1) and (10)	9	4
(2) and (5)	7	3
(2) and (8)	12	8
(2) and (10)	8	4
(9) and (5)	9	5
(9) and (7)	15	11
(9) and (8)	14	10
(9) and (10)	10	6

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.

The construction has been chosen in such a way that it works for all these derivation modes. Yet for the derivation mode $max_{\Delta objects}$ it is essential that the right-hand side contains at least three objects to enforce the application of all rules except the deletion rules $c_1 d \to c_1$ and $c_2 d \to c_2$. Otherwise the given construction of rules would not work correctly because of the missing condition for the multisets to be applied of being non-extendable. On the other hand, for the other derivation modes, i.e., $max_{\Delta objects}max$ and $max_{GENobjects}max$ as well as $max_{GENobjects}$, the construction would still work correctly if on the right-hand side of *all* rules, obviously again except the deletion rules $c_1 d \to c_1$ and $c_2 d \to c_2$, one dummy object d less is used.

These observations complete the proof. □

Corollary 5. *Any recursively enumerable set of (vectors of) natural numbers can be generated by a simple purely catalytic P system with only two catalysts working in one of the derivation modes $max_{\Delta objects}max$, $max_{GENobjects}max$, $max_{\Delta objects}$, or $max_{GENobjects}$.*

The result from Theorem 4 can be generalized to the case of $k \geq 2$ decrementable registers, following the ideas already elaborated in [19] as well as in [23]:

Theorem 6. *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. We extend the proof of Theorem 4 using some technical ideas from [19] and [23].

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

$$\Pi = (V, \{c_k \mid 1 \leq k \leq n\}, T, w_0, \mathcal{R})$$

simulating M. Without loss of generality, we may assume that the register machine M, as stated in Proposition 2 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.

As now we have an arbitrary number $n \geq 2$ of catalysts and only two of them shall do the work during the simulation of a *SUB*-instruction on register r in a similar way as elaborated in the proof of Theorem 4, we now use the corresponding catalyst c_r – which has to be left free for decrementing or for zero-checking in the first step of the simulation – and its "coupled" catalyst $c_{r\oplus_n 1}$ throughout all the simulation steps. Here $r\oplus_n 1$ for $r < n$ simply is $r + 1$, whereas for $r = n$ we define $n\oplus_n 1 = 1$. Moreover, the notation $[1..n]$ is used for the set (interval) of natural numbers $\{1, \ldots, n\}$.

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

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

For every $p \in B$ we define $Reg(p)$ to be the register affected by the instruction labeled by p; in addition, we define $Reg(l_h) = 1$.

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

$$l_0 l'_0 D'_{n,1} \prod_{i\in[1..n]} c_i.$$

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:

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

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 $\mathcal{R}$ 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:

  $$c_{Reg(p)} p \to c_{Reg(p)} q q' a_r d D'_{n,Reg(q)} \text{ or } c_{Reg(p)} p \to c_{Reg(p)} s s' a_r d D'_{n,Reg(s)},$$

  $$c_{Reg(p)\oplus_n 1} p' \to c_2 d^4.$$

  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, resulting in different sets of objects depending on the value of $|reg(r)|$, thereby neglecting the objects d_i, $1 \le i \le n$, and d; we emphasize that again the simulation is *deterministic*.

step	$\|reg(r)\|$	rule for c_r	rule for $c_{r\oplus_n 1}$	resulting objects
first	>0	$c_r a_r \to c_r \hat{a}_r d^2$	$c_{r\oplus_n 1} p' \to c_{r\oplus_n 1} \bar{p} d^{10} D'_{n,Reg(p)}$	$p, \bar{p}, \hat{a}_r$
	$=0$	$c_r p \to c_r d^2$	$c_{r\oplus_n 1} p' \to c_{r\oplus_n 1} \bar{p} d^{10} D'_{n,Reg(p)}$	$\bar{p}$
second	>0	$c_r \bar{p} \to c_r \hat{p} d^3$	$c_{r\oplus_n 1} p \to c_{r\oplus_n 1} d^9 D'_{n,Reg(p)}$	$\hat{a}_r, \hat{p}$
	$=0$	$c_r d \to c_r$ $(*)$	$c_{r\oplus_n 1} \bar{p} \to c_{r\oplus_n 1} s s' d^6 D'_{n,Reg(s)}$	s, s'
third	>0	$c_r \hat{p} \to c_1 q q' d^2 D'_{n,Reg(q)}$	$c_{r\oplus_n 1} \hat{a}_r \to c_{r\oplus_n 1} d^4$	q, q'

All arguments explained in the proof of Theorem 4 can be taken over to the general case, because the additional symbols generated by $D'_{n,Reg(l)}$, l being the label of a register machine instruction, always introduce exactly $n-2$ objects, i.e., in the case of $n=2$ we regain the same table as in the proof of Theorem 4. Yet it is essential for the correctness of the proof that $D'_{n,Reg(l)}$ is always generated from a program symbol and never to such combination of objects can be generated twice in any step of the derivation.
Again 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

$$c_1 l_h \to c_1 dd \text{ and}$$
$$c_2 l'_h \to c_2 dd.$$

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

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

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.

Again we mention that in the derivation modes $max_{\Delta objects}max$ and $max_{GENobjects}max$ as well as $max_{GENobjects}$, the construction would still work correctly if on the right-hand side of *all* rules except the deletion rules $c_1 d \to c_1$ and $c_2 d \to c_2$, one dummy object d less is used. These observations complete the proof. □

In sum, we then obtain the following result:

Corollary 7. *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.*

4. Simple purely catalytic P systems working in the derivation modes $min_{\Delta objects}max$ or $min_{GENobjects}max$

In this section we show that also the new derivation modes $min_{\Delta objects}max$ and $min_{GENobjects}max$ allow for simulating register machines by purely catalytic P systems with one catalyst less than in the original proofs given in [19].

Theorem 8. *For any register machine with two decrementable registers we can construct a simple purely catalytic P system with only two catalysts, working in one of the derivation modes $min_{\Delta objects}max$ or $min_{GENobjects}max$ which can simulate any computation of the register machine.*

Proof. We follow the proof elaborated for Theorem 4, yet now have to choose the inverse order relation for the lengths of the right-hand sides of the rules, as now the rules with the smallest right-hand sides have priority.

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

$$\Pi = (V, \{c_1, c_2\}, T, c_1 c_2 l_0 l'_0, \mathcal{R})$$

simulating M. Without loss of generality, we may assume that the register machine M, as stated in Proposition 2 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. Finally, we assume the two decrementable registers to be the first ones.

The following proof is elaborated for both the derivation modes $min_{\Delta objects}max$ and $min_{GENobjects}max$.

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 both catalysts are kept busy to guarantee that the simulation is executed in a correct way. As now we are only using non-extendable multisets of rules, a catalyst can only stay idle if the underlying configuration does not contain any object which can evolve together with the catalyst. On the other hand, 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 both derivation modes under consideration, as every rule in a purely catalytic P system has exactly two objects on its left-hand side.

Whenever possible, the dummy objects generated during the simulation steps have to be deleted immediately, as erasing rules have highest priority:

$$c_1 d \to c_1, \quad c_2 d \to c_2.$$

In order to keep both catalysts synchronized, the number of dummy objects throughout the computation has to be even. In fact, the simulation of the next instruction can only start when all dummy objects have been erased by these two rules $c_1 d \to c_1$ and $c_2 d \to c_2$.

The main basis for choosing the right number of the objects d on the right-hand side of the rules again is based on the importance and role of the rules

$$c_1 a_1 \to c_1 \hat{a}_1 d^6 \text{ and}$$
$$c_2 a_2 \to c_2 \hat{a}_2 d^6,$$

which should only be applicable in the first step of the simulation of a decrement instruction on register 1 and register 2, respectively. In addition to the previous proof construction, we now also have to guarantee that the number of dummy objects generated in every step of the derivation is even – which means that they are eliminated in a synchronized way immediately afterwards.

As usually done in corresponding proofs, 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 $r > 2$ are never changed again, as they represent output registers.

$$\begin{aligned} V &= \{a_r \mid 1 \le r \le m\} \cup \{\hat{a}_r \mid 1 \le r \le 2\} \\ &\cup \{p, p' \mid p \in B_{ADD} \cup \{l_h\}\} \\ &\cup \{p, p', \bar{p}, \hat{p} \mid p \in B_{SUB(r)}, 1 \le r \le n\} \\ &\cup \{c_1, c_2, d\}, \\ T &= \{a_r \mid 3 \le r \le m\}. \end{aligned}$$

The set $\mathcal{R}$ 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:

  $$c_1 p \to c_1 q q' a_r d^2 \text{ or } c_1 p \to c_1 s s' a_r d^2,$$
  $$c_2 p' \to c_2 d^4.$$

  The smaller numbers of dummy objects d used in these rules above guarantee that they have priority over the rules $c_1 a_1 \to c_1 \hat{a}_1 d^6$ and $c_2 a_2 \to c_2 \hat{a}_2 d^6$, respectively.

- $p : (SUB(r), q, s)$, with $p \in B_{SUB}$, $q, s \in B$, $1 \le r \le 2$.

  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 $c_r a_r \to c_r \hat{a}_r d^6$ 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, $r \in \{1, 2\}$, i.e., we use the following rules, resulting in different sets of objects depending on the value of $|reg(r)|$, thereby neglecting the objects d; we emphasize that the simulation is *deterministic*.

step	$\|reg(r)\|$	rule for c_{3-r}	rule for c_r	resulting objects
first	>0	$c_{3-r}p' \to c_{3-r}\bar{p}d^0$	$c_r a_r \to c_r \hat{a}_r d^6$	$p, \bar{p}, \hat{a}_r$
	$=0$	$c_{3-r}p' \to c_{3-r}\bar{p}d^0$	$c_r p \to c_r d^8$	$\bar{p}$
second	>0	$c_{3-r}p \to c_{3-r}d^2$	$c_r\bar{p} \to c_r\hat{p}d^4$	$\hat{a}_r, \hat{p}$
	$=0$	$c_{3-r}\bar{p} \to c_{3-r}ss'd^2$		s, s'
third	>0	$c_{3-r}\hat{a}_r \to c_{3-r}d^4$	$c_r\hat{p} \to c_1 qq'd^2$	q, q'

As in the zero-test case no register object a_r is present and all dummy objects have been erased before the second step of the simulation starts, the catalyst c_r stays idle in this second step of the zero-test case.

- $l_h : HALT$.

$$c_1 l_h \to c_1 d^2 \text{ and}$$
$$c_2 l_h' \to c_2$$

Due to the normal form of the register machine simulated by the P system, the two final labels l_h and l_h' can only appear at the end of a computation, when no other objects than the catalyst c and the objects representing the contents of the output registers are present any more in the configuration, i.e., these two rules are the last ones to be applied in a halting computation before in the last step the last two objects d are erased. This observation concludes the proof. □

Corollary 9. *Any recursively enumerable set of (vectors of) natural numbers can be generated by a simple purely catalytic P system with only two catalysts working in one of the derivation modes $min_{\Delta objects}max$ or $min_{GENobjects}max$.*

As for the maximal derivation modes, the result from Theorem 8 can be generalized to the case of $n \geq 2$ decrementable registers, following the proof ideas already elaborated in the proof of Theorem 6:

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

Proof. We follow the proof elaborated for Theorem 6, yet as in Theorem 8, have to choose the inverse order relation for the lengths of the right-hand sides of the rules, as now the rules with the smallest right-hand sides have priority.

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 two catalysts

$$\Pi = (V, \{c_1, \ldots, c_n\}, T, c_1 \ldots c_n l_0 l_0', \mathcal{R})$$

simulating M. Without loss of generality, we may assume that the register machine M, as stated in Proposition 2 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. Finally, we assume the n decrementable registers to be the first ones.

The following proof is elaborated for both the derivation modes $min_{\Delta objects}max$ and $min_{GENobjects}max$.

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 all n catalysts are kept busy to guarantee that the simulation is executed in a correct way. As now we are only using non-extendable multisets of rules, a catalyst can only stay idle if the underlying configuration does not contain any object which can evolve together with the catalyst. On the other hand, 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 both derivation modes under consideration, as every rule in a purely catalytic P system has exactly two objects on its left-hand side.

Whenever possible, the dummy objects generated during the simulation steps have to be deleted immediately, as erasing rules have highest priority:

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

In order to keep both catalysts synchronized, the number of dummy objects throughout the computation has to be even. In fact, the simulation of the next instruction can only start when all dummy objects have been erased by these n rules $c_k d \to c_k$, $1 \leq k \leq n$, which also means that at any time of the computation in Π the number of objects d must be a multiple of n.

The main basis for choosing the right number of the objects d on the right-hand side of the rules again is based on the importance and role of the rules

$$c_k a_k \to c_k \hat{a}_k d^{3*n},\ 1 \le k \le n,$$

which should only be applicable in the first step of the simulation of a decrement instruction on register k.

As usually done in corresponding proofs, 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.

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

The set $\mathcal{R}$ 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:

  $$c_{Reg(p)} p \to c_{Reg(p)} q q' a_r d^{1*n} D'_{n,Reg(q)} \text{ or } c_{Reg(p)} p \to c_{Reg(p)} s s' a_r d^{1*n} D'_{n,Reg(s)},$$

  $$c_{Reg(p)\oplus_n 1} p' \to c_2 d^{2*n}.$$

  All other catalysts c_i with $i \in [1..n] \setminus \{Reg(p), Reg(p) \oplus_n 1\}$ are forced to apply the rule $c_i d_i \to c_i d^{2*n}$.
  The smaller numbers of dummy objects d used in these rules above guarantee that they have priority over the rules $c_k a_k \to c_k \hat{a}_k d^{3*n}$.

- $p : (SUB(r), q, s)$, 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 $c_r a_r \to c_r \hat{a}_r d^{3*n}$ 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. The following table summarizes the rules to be used for the simulation of the SUB-instruction on register r, $r \in [1..n]$, i.e., we use the following rules, depending on the value of $|reg(r)|$; we emphasize that the simulation is *deterministic*.

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

  As in the zero-test case no register object a_r is present and all dummy objects have been erased before the second step of the simulation starts, the catalyst c_r stays idle in this second step of the zero-test case.

- $l_h : HALT$.

  $$c_1 l_h \to c_1 d^n \text{ and}$$
  $$c_2 l'_h \to c_2$$

Due to the normal form of the register machine simulated by the P system, the two final labels l_h and l'_h can only appear at the end of a computation, when no other objects than the catalyst c and the objects representing the contents of the output registers are present any more in the configuration, i.e., these two rules are the last ones to be applied in a halting computation before in the last step the last n objects d are erased. This observation concludes the proof. □

In sum, we then obtain the following result:

Corollary 11. *Purely catalytic P systems working in any of the derivation modes $min_{\Delta objects}max$ or $min_{GENobjects}max$ are computationally complete, i.e., they can compute any partial recursive relation on natural numbers.*

5. Conclusion

In this paper we have continued our investigation of the new derivation modes $max_{GENobjects}max$, $max_{\Delta objects}max$, $max_{GENobjects}$, and $max_{\Delta objects}$ as introduced in [10], now applied in simple purely catalytic P systems. In contrast to the proof technique used there for catalytic P systems, we here had to go back to the original construction as elaborated in [19], yet also using ideas related to energy-controlled P systems as described in [4] in order to be able to show that *two* catalysts are enough for generating all recursively enumerable sets of multisets.

Moreover, we could prove similar results for purely catalytic P systems working in the derivation modes $min_{GENobjects}max$ and $min_{\Delta objects}max$. In sum, we have even shown that purely catalytic P systems working in any of the six derivation modes mentioned above are computationally complete, i.e., they can compute any partial recursive relation on natural numbers.

The results obtained in this paper can also be extended to P systems dealing with strings, following the definitions and notions as, for example, used in [25], thus showing computational completeness for computing with strings.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors gratefully thank the referees for their useful comments.

Artiom Alhazov 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.

Rudolf Freund acknowledges the TU Wien Bibliothek for financial support through its Open Access Funding Programme.

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

References

[1] A. Alhazov, B. Aman, R. Freund, 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, in: Lecture Notes in Computer Science, vol. 8961, Springer, 2014, Revised Selected Papers.

[2] A. Alhazov, B. Aman, R. Freund, Gh. Păun, 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, Proceedings, Turku, Finland, August 5–8, 2014, in: Lecture Notes in Computer Science, vol. 8614, Springer, 2014.

[3] A. Alhazov, R. Freund, 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, in: Lecture Notes in Computer Science, vol. 8961, Springer, 2014, Revised Selected Papers.

[4] A. Alhazov, R. Freund, S. Ivanov, Variants of energy-controlled P systems, in: Proceedings of NIT 2016, 2016.

[5] A. Alhazov, R. Freund, S. Ivanov, Variants of P systems with activation and blocking of rules, Nat. Comput. 18 (3) (2019) 593–608.

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

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

[8] A. Alhazov, R. Freund, S. Ivanov, P systems with limited number of objects, J. Membr. Comput. 3 (2021) 1–9.

[9] A. Alhazov, R. Freund, S. Ivanov, When catalytic P systems with one catalyst can be computationally complete, J. Membr. Comput. 3 (3) (2021) 170–181.

[10] A. Alhazov, R. Freund, S. Ivanov, S. Verlan, Variants of simple P systems with one catalyst being computationally complete, in: Gy. Vaszil, C. Zandron, G. Zhang (Eds.), International Conference on Membrane Computing ICMC 2021, Proceedings, 2021.

[11] A. Alhazov, R. Freund, M. Oswald, S. Verlan, Partial halting and minimal parallelism based on arbitrary rule partitions, Fundam. Inform. 91 (1) (2009) 17–34.

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

[13] A. Alhazov, R. Freund, S. Verlan, 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, in: Lecture Notes in Computer Science, vol. 10105, Springer, 2017, Revised Selected Papers.

[14] J. Dassow, Gh. Păun, Regulated Rewriting in Formal Language Theory, Springer, 1989.

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

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

[17] R. Freund, 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, in: books@ocg.at, vol. 321, Österreichische Computer Gesellschaft, 2016.
[18] R. Freund, How derivation modes and halting conditions may influence the computational power of P systems, J. Membr. Comput. 2 (1) (2020) 14–25.
[19] R. Freund, L. Kari, M. Oswald, P. Sosík, Computationally universal P systems without priorities: two catalysts are sufficient, Theor. Comput. Sci. 330 (2) (2005) 251–266.
[20] R. Freund, A. Leporati, G. Mauri, A.E. Porreca, S. Verlan, C. Zandron, Flattening in (tissue) P systems, in: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, in: Lecture Notes in Computer Science, vol. 8340, Springer, 2014, pp. 173–188.
[21] R. Freund, M. Oswald, Partial halting in P systems, Int. J. Found. Comput. Sci. 18 (6) (2007) 1215–1225.
[22] R. Freund, M. Oswald, 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, in: books@ocg.at, vol. 294, Österreichische Computer Gesellschaft, 2013.
[23] R. Freund, M. Oswald, Gh. Păun, Catalytic and purely catalytic P systems and P automata: control mechanisms for obtaining computational completeness, Fundam. Inform. 136 (1–2) (2015) 59–84.
[24] R. Freund, Gh. Păun, 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, in: EPTCS, vol. 128, 2013.
[25] R. Freund, P. Sosík, 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, in: Lecture Notes in Computer Science, vol. 9504, Springer, 2015, Revised Selected Papers.
[26] R. Freund, S. Verlan, A formal framework for static (tissue) P systems, in: G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing, in: Lecture Notes in Computer Science, vol. 4860, Springer, 2007, pp. 271–284.
[27] K. Krithivasan, Gh. Păun, A. Ramanujan, On controlled P systems, Fundam. Inform. 131 (3–4) (2014) 451–464.
[28] M.L. Minsky, Computation. Finite and Infinite Machines, Prentice Hall, Englewood Cliffs, NJ, 1967.
[29] Gh. Păun, Computing with membranes, J. Comput. Syst. Sci. 61 (1) (2000) 108–143.
[30] Gh. Păun, Membrane Computing: An Introduction, Springer, 2002.
[31] Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), The Oxford Handbook of Membrane Computing, Oxford University Press, 2010.
[32] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, 1997.
[33] The P Systems Website, http://ppage.psystems.eu/.