Catalytic P Systems with Weak Priority of Catalytic Over Non-catalytic Rules

Artiom Alhazov[1], Rudolf Freund[2], and Sergiu Ivanov[3]

[1] Vladimir Andrunachievici Institute of Mathematics and Computer Science
Academiei 5, Chișinău, MD-2028, Moldova
artiom@math.md

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

[3] IBISC, Univ Évry, Paris-Saclay University
23, boulevard de France 91034 Évry, France
sergiu.ivanov@ibisc.univ-evry.fr

Summary. 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 particularly 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. In this paper we show that one catalyst is sufficient for obtaining even computational completeness if catalytic rules have weak priority over the non-catalytic rules.

1 Introduction

Membrane systems were introduced in [8] as a multiset-rewriting model of computing inspired by the structure and the functioning of the living cell. During two decades now membrane computing has attracted the interest of many researchers, and its development is documented in two textbooks, see [9] and [10]. For actual information see the P systems webpage [12] 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 [8] 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 $ca \to cv$, 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 $a \to v$.

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 [3] it has already been shown that two 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, without catalysts only regular (semi-linear) sets can be generated when using 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 [5], 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. On the other hand, for catalytic P systems with only one catalyst a lower bound has been established in [6]: P systems with one catalyst can simulate partially blind register machines, i.e., they can generate more than just semi-linear sets.

In this paper we now return to the idea of using a priority relation on the rules, but take only a very weak form of such a priority relation: we only require 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, computational completeness can be established for catalytic P systems with only one catalyst, which is the main result of our paper.

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. For further notions and results in formal language theory we refer to textbooks like [2] and [11].

2.1 Register Machines

Register machines are well-known universal devices for computing (or generating or accepting) sets of vectors of natural numbers.

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

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

where

- m *is the number of registers,*
- P *is the set of instructions bijectively labeled by elements of* B*,*
- $l_0 \in B$ *is the initial label, and*
- $l_h \in B$ *is the final label.*

The instructions of M *can be of the following forms:*

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

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 a 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.

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

For useful results on the computational power of register machines, we refer to [7]; 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 an accepting 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.

2.2 Partially Blind Register Machines

We now consider one-way nondeterministic machines which have registers allowed to hold positive or negative integers and which accept by final state with all registers being zero. Such machines are called *blind* if their actions depend on state and input alone and not on the register configuration. They are called *partially blind* if they block when any register is negative (i.e., only non-negative register contents is allowed) but do not know whether or not any of the registers contains zero.

Definition 2. *A* partially blind register machine *is a construct*

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

where

- m *is the number of registers,*
- P *is the set of instructions bijectively labeled by elements of* B*,*
- $l_0 \in B$ *is the initial label, and*
- $l_h \in B$ *is the final label.*

The instructions of M *can be of the following forms:*

- $p : (ADD(r), q, s)$*, 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*.*
- $p : (SUB(r), q)$*, with* $p \in B \setminus \{l_h\}$*,* $q \in B$*,* $1 \leq r \leq m$*.*
  If the value of register r *is not zero then decrease the value of register* r *by one and jump to instruction* l_2*, otherwise abort the computation.*
- $l_h : HALT$*.*
  Stop the execution of the register machine.

Again, a *configuration* of a partially blind 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.

A computation works as for a register machine, yet with the restriction that a computation is aborted if one tries to decrement a register which is zero. Moreover, computing, accepting or generating now also requires all registers (except output registers) to be empty at the end of the computation.

Example 1. In [6] it was shown that the vector set

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

(which is not semi-linear) can be generated by a P system with only one catalyst and 19 rules.

2.3 Catalytic P Systems

As in [6], the following definition cites Definition 4.1 in Chapter 4 of [10].

Definition 3. *An* extended catalytic P system *of degree* $m \geq 1$ *is a construct*

$$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0)$$

where

- O *is the alphabet of objects;*

- $C \subseteq O$ *is the alphabet of catalysts;*
- μ *is a membrane structure of degree* m *with membranes labeled in a one-to-one manner with the natural numbers* $1, \ldots, m$*;*
- $w_1, \ldots, w_m \in O^*$ *are the multisets of objects initially present in the* m *regions of* μ*;*
- R_i, $1 \leq i \leq m$, *are finite sets of evolution rules over* O *associated with the regions* $1, 2, \ldots, m$ *of* μ*; these evolution rules are of the forms* $ca \to cv$ *or* $a \to v$*, where* c *is a catalyst,* a *is an object from* $O \setminus C$*, and* v *is a string from* $((O \setminus C) \times \{here, out, in\})^*$*;*
- $i_0 \in \{0, 1, \ldots, m\}$ *indicates the output region of* Π*.*

The membrane structure and the multisets in Π constitute a *configuration* of the P system; the *initial configuration* is given by the initial multisets $w_1, \ldots, w_m$. A transition between configurations is governed by the application of the evolution rules, which is done in the maximally parallel way, i.e., only applicable multisets of rules which cannot be extended by further rules are to be applied to the objects in all membrane regions.

The application of a rule $u \to v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to the targets *in* and *out*. We refer to [10] for further details and examples.

The P system continues with applying multisets of rules in the maximally parallel way until there remain no applicable rules in any region of Π. Then the system *halts*. We consider the number of objects from $O \setminus C$ contained in the output region i_0 at the moment when the system halts as the *result* of the underlying computation of Π. The system is called *extended* since the catalytic objects in C are not counted to the result of a computation. Yet as often done in the literature, in the following we will omit the term *extended* and just speak of *catalytic P systems*, especially as we will restrict ourselves to P systems with only one catalyst.

The set of results of all computations possible in Π is called the set of natural numbers *generated by* Π and it is denoted by $N(\Pi)$ if we only count the total number of objects in the output membrane; if we distinguish between the multiplicities of different objects, we obtain a set of vectors of natural numbers denoted by $Ps(\Pi)$.

Remark 1. As in this paper we only consider catalytic P systems with only one catalyst, without loss of generality, we can restrict ourselves to one-membrane catalytic P systems with the single catalyst in the skin membrane, by taking into account the well-known flattening process, e.g., see [4].

Remark 2. Finally, we make the convention that a one-membrane catalytic P system with the single catalyst in the skin membrane and with internal output in the skin membrane, not taking into account the single catalyst c for the results, throughout the rest of the paper will be described without specifying the trivial membrane structure or the output region (assumed to be the skin membrane), i.e., we will just write

$$\Pi = (O, \{c\}, w, R)$$

where O is the set of objects, c is the single catalyst, w is the initial input specifying the initial configuration, and R is the set of rules.

As already mentioned earlier, the following result was shown in [6], 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 have at least the computational power of partially blind register machines.*

3 Weak Priority of Catalytic Rules

In this paper we now study catalytic P systems with only one catalyst in which the catalytic rules have weak priority over the non-catalytic rules.

Example 2. To illustrate this weak priority of catalytic rules over the non-catalytic rules, consider the rules $ca \to cb$ and $a \to d$. If the current configuration contains $k > 0$ copies of a, then the catalytic rule $ca \to cb$ *must* be applied to one of the copies, while the rest of objects a may be taken up by the non-catalytic rule $a \to d$. In particular, if $k = 1$, only $ca \to cb$ may be applied.

We would like to highlight the fact that weak priority of catalytic rules is much weaker than the general weak priority, as the priority relation is only constrained by the types of rules.

Remark 3. The reverse weak priority, i.e., non-catalytic rules having priority over catalytic rules, is useless, since it is equivalent to removing all catalytic rules for which there are non-catalytic rules with the same symbol on the left-hand side of the rule. In that way we just end up with an even restricted variant of P systems with only one catalyst.

3.1 Computational Completeness with Weak Priority

In this section, we show that catalytic P systems with one catalyst only and with weak priority of catalytic rules are computationally complete.

Theorem 1. *Catalytic P systems with only one catalyst and with weak priority of catalytic rules over the non-cooperative rules are computationally complete.*

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 $\Pi = (O, \{c\}, w, R)$ simulating M. Without loss of generality, we may assume that, depending on its use as an accepting or generating or computing device, the register machine M, as stated in Proposition 1, Proposition 2, and Proposition 3, fulfills

the condition that on the output registers we never apply any *SUB*-instruction. The following proof is given for the most general case of a register machine computing any partial recursive relation on vectors of natural numbers with l components as input and vectors of natural numbers with k components as output using precisely $l+2+k$ registers, where without loss of generality, we may assume that at the end of a successful computation the first $l+2$ registers are empty, and, moreover, on the output registers, i.e., the last k registers, no *SUB*-instruction is ever used. In fact, the proof works for any number n 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 symbols except the catalyst c and the output symbols (representing the contents of the output registers) go through a cycle of length $2n$ where n is the number of decrementable registers of the simulated register machine. When the symbols are traversing the r-th section of the n sections of length 2, they "know" that they are to probably simulate a *SUB*-instruction on register r of the register machine M.

The alphabet O of symbols includes register symbols $(a_r, 2i-1), (a_r, 2i)$ for every decrementable register r of the register machine and only the register symbol a_r for each of the k output registers r, $m-k+1 \leq r \leq m$, the state symbols $(p, 2i-1), (p, 2i)$, $1 \leq i \leq n$, for every *ADD*-instruction of the register machine as well as, for $p \in B_{SUB(r)}$ the state symbols $(p, 2i-1), (p, 2i)$ for $1 \leq i \leq r$ as well as $(p, 2j-1)^-$ and $(p, 2j)^0$ for $r+1 \leq j \leq n$ for every *SUB*-instruction $p : (SUB(r), q, s)$ of the register machine, i.e., $p \in B_{SUB(r)}$, where $B_{SUB(r)}$ denotes the set of labels of all *SUB*-instruction $p : (SUB(r), q, s)$ of decrementable registers r. Moreover, we use decrement witness symbols λ_r for every decrementable register r, as well as the catalyst c and the trap symbol $\#$. Observing that $n = m - k$, in total we get the following set of objects:

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

B_{ADD} denotes the set of labels of *ADD*-instructions.

The starting configuration of Π is

$$w = c(l_0, 1)\alpha_0,$$

where l_0 is the starting label of the machine and α_0 is the multiset encoding the initial values of the registers.

All register symbols a_r, $1 \leq r \leq n$, representing the contents of decrementable registers, are equipped with the rules evolving them throughout the whole cycle:

$$(a_r, i) \to (a_r, i+1), 1 \leq r \leq 2n-1; \qquad (a_r, 2n) \to (a_r, 1). \tag{1}$$

The construction also includes the trap rule $\# \to \#$: once the trap symbol $\#$ is introduced, it will always keep the system busy and prevent it from halting and thus from producing a result.

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

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

The (variants of the) symbol p cycles together with all the other symbols, always involving the catalyst:

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

At the end of the cycle, the register is incremented and the non-deterministic jump to q or s occurs: for r being a decrementable register, we take

$$c(p, 2n) \to c(q, 1)(a_r, 1), \qquad c(p, 2n) \to c(s, 1)(a_r, 1), \tag{3}$$

whereas for r being a register never to be decremented, we take

$$c(p, 2n) \to c(q, 1)a_r, \qquad c(p, 2n) \to c(s, 1)a_r \tag{4}$$

The output symbols need not undergo the cycle, in fact, they must not do that because otherwise the computation would never stop. When the computation of the register machine halts, only output symbols will be present, as we have assumed that at the end of a computation all decrementable registers will be empty, i.e., no cycling symbols will be present any more in the P system. Finally, we have to mention that if q or s is the final label l_h, then we take λ instead, which means that also the P system will halt, because, as already explained above, the only symbols left in the configuration will be output symbols, for which no rules exist.

The state symbol is not allowed to evolve without the catalyst:

$$(p, i) \to \#, \quad 1 \leq i \leq 2n. \tag{5}$$

Hence, in that way it is guaranteed that the catalyst cannot be used in another way, i.e., affecting a symbol (a_r, i) as explained below during the simulation of a *SUB*-instruction on register r.

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

The simulation of a *SUB* instruction is carried out in two steps of the cycle, i.e., in steps $2r-1$ and $2r$.

Before reaching simulation phase r, i.e., step $2r-1$, the state symbol goes through the cycle, necessarily involving the catalyst:

$$c(p,i) \to c(p,i+1) \quad > \quad (p,i) \to \#, \qquad 1 \leq i < 2r-1. \tag{6}$$

Although by the definition of the P systems with priority of catalytic rules, the catalytic rule has priority over the non-catalytic rule for (p,i), we indicate the general priority relation by the sign $<$ (or $>$ for the reverse relation) in order to make the situation even clearer.

In the first step of the simulation phase r, i.e., in step $2r-1$, the state symbol releases the catalyst to try to perform the decrement and to produce a witness symbol if register r is not empty:

$$(p,2r-1) \to (p,2r), \qquad c(a_r,2r-1) \to c\lambda_r. \tag{7}$$

Note that due to the counters identifying the position of the register symbols in the cycle, it is guaranteed that the catalytic rule transforming $(a_r,2r-1)$ picks the correct register symbol. Furthermore, due to the priority of the catalytic rules, one of the the register symbols $(a_r,2r-1)$ *must* be transformed by the catalytic rule if present, instead of continuing along its cycle.

In the second step of simulation phase r, i.e., in step $2r$, the detection of the possible decrement happens. The outcome is stored in the state symbol:

$$\begin{array}{lcl} c\lambda_r \to c & > & \lambda_r \to \#, \\ (p,2r) \to (p,2r+1)^- & < & c(p,2r) \to c(p,2r+1)^0. \end{array} \tag{8}$$

If in the first step of the simulation phase the catalyst did manage to decrement the register, it produced λ_r. Thus, in the second step, the catalyst must erase λ_r, because otherwise this symbol will trap the computation (and because catalytic rules have priority). This means that the catalyst is not available to produce $(p,2r+1)^0$, and the rule $(p,2r) \to (p,2r+1)^-$ must be applied due to the maximally parallel mode. If, on the other hand, the decrement did *not* succeed in the previous step, both rules $(p,2r) \to (p,2r+1)^-$ and $c(p,2r) \to c(p,2r+1)^0$ can be applied, but due to the priority of the catalytic rules, the second rule must be preferred, thus producing $(p,2r+1)^0$. Therefore, the superscript of the state symbol correctly reflects the outcome of the decrement: it is $-$ if the decrement succeeded, and 0 if it did not.

After the simulation of the decrement, the state symbols evolve to the end of the cycle and produce the corresponding next state symbols:

$$\begin{array}{lll} (p,i)^- \to (p,i+1)^-, \quad r+2 \leq i \leq n, & & (p,n+1)^- \to (q,1), \\ (p,i)^0 \to (p,i+1)^0, \quad r+2 \leq i \leq n, & & (p,n+1)^0 \to (s,1). \end{array} \tag{9}$$

If the register r is the last decrementable one, i.e., $r=n$, then equations 8 and 9 together read as follows:

$$\begin{array}{lcl} c\lambda_n \to c & > & \lambda_n \to \#, \\ (p,n+1) \to (q,1) & < & c(p,n+1) \to c(s,1). \end{array} \tag{10}$$

Finally, we again mention that if q or s is the final label l_h, then we take λ instead, which means that not only the register machine but also the P system halts, because, as already explained above, the only symbols left in the configuration will be output symbols, for which no rules exist. □

We would also like to emphasize that the simulation is what may be called toxic/trap-deterministic: the only non-deterministic choice happens between a rule producing a trap symbol # and another one which does not introduce #. This means that the appearance of the trap symbol may immediately abort the computation, which is the concept used for toxic P systems as introduced in [1]. Using the trap symbol # as such a toxic object, the only successful computations are simulating register machines in a quasi-deterministic way with a look-ahead of one, i.e., considering all possible configurations computable from a given one, there is at most one successful continuation of the computation.

For future research it remains a challenging question whether the length of the cycle now being $2n$ can still be reduced.

4 Conclusion

In this paper we revisited 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. In this paper, we come closer to showing computational completeness: we give a construction that simulates an arbitrary register machine with a very weak ingredient—the weak priority of catalytic rules over non-catalytic rules.

On the other hand, we still conjecture that P systems with one catalyst and no additional control mechanisms cannot reach computational completeness. Finding an answer to the question of characterizing the computational power of P systems with one catalyst therefore still remains one of the biggest challenges in the theory of P systems, although the result established in our paper has made the gap between the computational power of P systems with one catalyst and computational completeness smaller again.

The result obtained in this paper can also be extended to P systems dealing with strings, following the definitions and notions used in [6], thus showing computational completeness for computing with strings.

Acknowledgements

The ideas, concepts, and results described in this paper have mainly been developed in the inspiring atmosphere of the 18^{th} Brainstorming Week on Membrane Computing during the first week of February 2020 in Sevilla.

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. Artiom Alhazov and Rudolf Freund. P systems with toxic objects. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík, and Claudio Zandron, editors, *Membrane Computing – 15th International Conference, CMC 2014, Prague, Czech Republic, August 20–22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 99–125. Springer, 2014.
2. Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*. Springer, 1989.
3. Rudolf Freund, Lila Kari, Marion Oswald, and Petr Sosík. Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330(2):251–266, 2005.
4. Rudolf Freund, Alberto Leporati, Giancarlo Mauri, Antonio E. Porreca, Sergey Verlan, and Claudio Zandron. Flattening in (tissue) P systems. In Artiom Alhazov, Svetlana Cojocaru, Marian Gheorghe, Yurii Rogozhin, Grzegorz Rozenberg, and Arto Salomaa, editors, *Membrane Computing*, volume 8340 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 2014.
5. Rudolf Freund, Marion Oswald, and Gheorghe Păun. Catalytic and purely catalytic P systems and P automata: Control mechanisms for obtaining computational completeness. *Fundam. Inform.*, 136(1–2):59–84, 2015.
6. Rudolf Freund and Petr Sosík. On the power of catalytic P systems with one catalyst. In Grzegorz Rozenberg, Arto Salomaa, José M. Sempere, and Claudio Zandron, editors, *Membrane Computing – 16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers*, volume 9504 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2015.
7. Marvin L. Minsky. *Computation. Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
8. Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
9. Gheorghe Păun. *Membrane Computing: An Introduction*. Springer, 2002.
10. Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors. *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
11. Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*. Springer, 1997.
12. The P Systems Website. `http://ppage.psystems.eu/`.