Journal of Membrane Computing (2020) 2:14–25
https://doi.org/10.1007/s41965-019-00028-9

REGULAR PAPER

How derivation modes and halting conditions may influence the computational power of P systems

Rudolf Freund[1]

Received: 2 August 2019 / Accepted: 11 November 2019 / Published online: 9 December 2019

Abstract
In the area of P systems, besides the standard maximally parallel derivation mode, many other derivation modes have been investigated, too. In this overview paper, many variants of hierarchical P systems using different derivation modes are considered and the effects of using different derivation modes, especially the maximally parallel derivation modes and the maximally parallel set derivation modes, on the generative and accepting power are illustrated. Moreover, an overview on some control mechanisms used for P systems is given. Furthermore, besides the standard total halting, we also consider different halting conditions such as unconditional halting and partial halting and explain how the use of different halting conditions may considerably change the computing power of P systems.

Keywords Derivation modes · Halting conditions · P systems

1 Introduction

The basic model of *P systems* as introduced in [21] can be considered as a distributed multiset rewriting system, where all objects—if possible—evolve in parallel in the membrane regions and may be communicated through the membranes. But also P systems operating on more complex objects (e.g., strings, arrays) are often considered, too, for instance, see [10].

Besides the maximally parallel derivation mode, many other derivation modes have been investigated during the last two decades. Hence, in this paper, the definitions of the standard derivation modes used for P systems are recalled. Various interpretations of derivation modes known from the P systems area are illustrated and well-known results are presented in a different manner.

Moreover, we consider not only the standard total halting, but also other halting conditions such as unconditional halting, see [7], and partial halting, see [14]. We explain and give some examples how the use of different halting modes may considerably change the computing power of P systems.

Overviews on the field of P systems can be found in the monograph [22] and the Handbook of Membrane Computing [23]; for actual news and results we refer to the P systems webpage [26] as well as to the Bulletin of the International Membrane Computing Society. The reader is assumed to be familiar with the basic definitions and notations of P systems as well as of the commonly used derivation modes and halting conditions.

The rest of the paper is organized as follows: In the next section, basic notions from formal language theory needed in this paper are recalled. In Sect. 3, the definition of the basic model of P systems is given and explained, including the standard derivation modes used in many papers on P systems, the basic types of rules, as well as the main halting conditions found in the literature and considered in more detail in Sect. 7. Some well-known results are summarized in a compact form in Sect. 4; special focus is put on results for catalytic P systems regarding the number of rules needed for simulating (the instructions of) register machines. In Sect. 5, important results for P systems with control mechanisms are recalled, including the variant of P systems with target selection, which is one of the very few models known

A short version of this paper especially focusing on the influence of choosing different parallel derivation modes was presented at CMC 20, the 20th anniversary edition of the meeting of the membrane systems community, in Curtea de Argeş, Romania, from August 5 to 9, 2019.

✉ Rudolf Freund
rudi@emcc.at

[1] TU Wien, Institut für Logic and Computation, Favoritenstraße 9-11, 1040 Wien, Austria

from the literature of P systems which takes advantage of using a non-trivial membrane structure. An own section then is devoted to a special derivation mode called *minimal parallelism* and its variants. Examples and results for the halting conditions different from the standard variant of total halting are considered in Sect. 7. A short summary concludes the paper.

2 Prerequisites

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

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

The family of regular, context-free, and recursively enumerable string languages is denoted by REG, CF, and RE, respectively. For example, $PsREG = PsCF$, which is the reason why in the area of multiset rewriting CF plays no role at all, and in the area of membrane computing we usually get characterizations of $PsREG$ and $PsRE$.

An *extended Lindenmayer system* (an *E0L system* for short) is a construct $G = (V, T, P, w)$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, $w \in V^*$ is the axiom, and P is a finite set of non-cooperative rules over V of the form $a \to u$. In a derivation step, each symbol present in the current sentential form is rewritten using one rule arbitrarily chosen from P. The language generated by G, denoted by $L(G)$, consists of all the strings over T which can be generated in this way by starting from the initial string w. An E0L system with $T = V$ is called a *0L system*.

For more details of formal language theory, the reader is referred to the monographs and handbooks in this area as [9] and [24].

2.1 Register machines

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

- $l_1 : (ADD(j), l_2, l_3)$, with $l_1 \in B\backslash\{l_h\}, l_2, l_3 \in B, 1 \le j \le m$. Increases the value of register j by one, followed by a non-deterministic jump to instruction l_2 or l_3. This instruction is usually called *increment*.
- $l_1 : (SUB(j), l_2, l_3)$, with $l_1 \in B\backslash\{l_h\}, l_2, l_3 \in B, 1 \le j \le m$. If the value of register j is zero then jump to instruction l_3; otherwise, the value of register j is decreased by one, followed by a jump to instruction l_2. The two cases of this instruction are usually called *zero-test* and *decrement*, respectively.
- $l_h : HALT$. Stops the execution of the register machine.

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

M is called deterministic if in all ADD-instructions $p : (\mathtt{ADD}(r), q, s)$, it holds that $q = s$; in this case we write $p : (\mathtt{ADD}(r), q)$.

Register machines provide a computationally complete model for computations with natural numbers:

In the generating case, we start with empty registers, use the last two registers for the necessary computations and take as results the vectors of natural numbers $(x_1, \ldots, x_d)$ obtained as contents of the first d registers 1 to d in all possible halting computations. Without loss of generality, we may assume that at the beginning of a computation, all registers are empty and that during any computation of M, only the registers $d + 1$ and $d + 2$ can be decremented.

In the accepting case, we start with the natural numbers $x_1, \ldots, x_d$ in the first d registers (and with 0 in the registers $d + 1$ and $d + 2$) and use the two additional registers $d + 1$ and $d + 2$ for the necessary computations; in this case, all registers may be decremented; moreover, the register

machine can be assumed to be deterministic, i.e., we only have ADD-instructions of the form $l_1 : (\mathtt{ADD}(j), l_2)$, with $l_1 \in B \backslash \{l_h\}$, $l_2 \in B$, $1 \leq j \leq m$. The vector $(x_1, \ldots, x_d)$ is accepted if and only if M halts with the natural numbers $x_1, \ldots, x_d$ having been given as input in the first d registers.

For these and other useful results on the computational power of register machines, we refer to [20].

3 A general model for hierarchical P systems

We now recall the main definitions of the general model for hierarchical P systems and the basic derivation modes as defined, for example, in [18]. Moreover, we define the halting conditions discussed in this paper.

A *(hierarchical) P system (with rules of type X)* working in the derivation mode δ is a construct

$\Pi = (V, T, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, f, \Longrightarrow_{\Pi,\delta})$ where

- V is the alphabet of *objects*;
- $T \subseteq V$ is the alphabet of *terminal objects*;
- μ is the hierarchical membrane structure (a rooted tree of membranes) with the membranes uniquely labeled by the numbers from 1 to m;
- $w_i \in V^*, 1 \leq i \leq m$, is the *initial multiset* in membrane i;
- $R_i, 1 \leq i \leq m$, is a finite set of *rules of type X* assigned to membrane i;
- f is the label of the membrane from which the result of a computation has to be taken from (in the generative case) or into which the initial multiset has to be given in addition to w_f (in the accepting case);
- $\Longrightarrow_{\Pi,\delta}$ is the derivation relation under the derivation mode δ.

The symbol X in "rules of type X" may stand for "evolution", "communication", "membrane evolution", etc. In this paper, we will mainly consider non-cooperative as well as catalytic and purely catalytic rules, see Sect. 3.2.

A configuration is a list of the contents of each membrane region; a sequence of configurations $C_1, \ldots, C_k$ is called a *computation* in the derivation mode δ if $C_i \Longrightarrow_{\Pi,\delta} C_{i+1}$ for $1 \leq i < k$. The derivation relation $\Longrightarrow_{\Pi,\delta}$ is defined by the set of rules in Π and the given derivation mode which determines the multiset of rules to be applied to the multisets contained in each membrane region.

The *language generated by* Π is the set of all terminal multisets which can be obtained in the output membrane f starting from the initial configuration $C_1 = (w_1, \ldots, w_m)$ using the derivation mode δ in a halting computation, i.e.,

$$L_{\mathrm{gen},\delta}(\Pi) = \left\{ (C(f))_{T^\circ} \mid C_1 \overset{*}{\Longrightarrow}_{\Pi,\delta} C \wedge \neg\exists C' : C \Longrightarrow_{\Pi,\delta} C' \right\},$$

where $(C(f))_{T^\circ}$ stands for the terminal part of the multiset contained in the output membrane f of the configuration C; the configuration C is halting, i.e., no further configuration C' can be derived from it.

The family of languages of multisets generated by P systems of type X with at most n membranes in the derivation mode δ is denoted by $Ps_{\mathrm{gen},\delta}OP_n(X)$.

We may also consider P systems as accepting mechanisms: in membrane f, we add the input multiset w_0 to w_f in the initial configuration $C_1 = (w_1, \ldots, w_m)$ thus obtaining $C_1[w_0] = (w_1, \ldots, w_f w_0, \ldots, w_m)$; the input multiset w_0 is accepted if there exists a halting computation in the derivation mode δ starting from $C_1[w_0]$, i.e.,

$$L_{\mathrm{acc},\delta}(\Pi) = \left\{ w_0 \in T^\circ \mid \exists C : \left(C_1[w_0] \overset{*}{\Longrightarrow}_{\Pi,\delta} C \wedge \neg\exists C' : C \Longrightarrow_{\Pi,\delta} C' \right) \right\}.$$

Then, the family of languages of multisets accepted by P systems of type X with at most n membranes in the derivation mode δ is denoted by $Ps_{\mathrm{acc},\delta}OP_n(X)$.

We finally mention that P systems can also be used to compute functions and relations, with using f both as input and output membrane or even using two different membranes for the input and the output. Yet, in this paper, we will mainly focus on the generating case.

3.1 Derivation modes

The set of all multisets of rules applicable in a P system to a given configuration C is denoted by $Appl(\Pi, C)$ and can be restricted by imposing specific conditions, thus yielding the following basic derivation modes (for example, see [18] for formal definitions):

- asynchronous mode (abbreviated *asyn*): at least one rule is applied;
- sequential mode (*sequ*): only one rule is applied;
- maximally parallel mode (*max*): a non-extendable multiset of rules is applied;
- maximally parallel mode with maximal number of rules (max_{rules}): a non-extendable multiset of rules of maximal possible cardinality is applied;
- maximally parallel mode with maximal number of objects ($max_{objects}$): a non-extendable multiset of rules affecting as many objects as possible is applied.

In [6], the *set* variants of these derivation modes are considered, i.e., each rule can be applied at most once. Thus, starting from the set of all sets of applicable rules, we obtain the set modes *sasyn*, *smax*, $smax_{rules}$, and $smax_{objects}$ (the sequential mode is already a set mode by definition):

- asynchronous set mode (abbreviated *sasyn*): at least one rule is applied, but each rule at most once;
- maximally parallel set mode (*smax*): a non-extendable set of rules is applied;
- maximally parallel set mode with maximal number of rules ($smax_{rules}$): a non-extendable set of rules of maximal possible cardinality is applied;
- maximally parallel set mode with maximal number of objects ($smax_{objects}$): a non-extendable set of rules affecting as many objects as possible is applied.

Let us denote the set of all multisets (possibly only sets) of rules applicable in a P system Π to a given configuration C in the derivation mode δ by $Appl(\Pi, C, \delta)$. We immediately observe that $Appl(\Pi, C, asyn) = Appl(\Pi, C)$.

To collect the set and multiset derivation modes, we use the following notations:

$D_S = \{sequ, sasyn, smax, smax_{rules}, smax_{objects}\}$ and
$D_M = \{asyn, max, max_{rules}, max_{objects}\}$.

3.2 Standard rule variants

Non-cooperative rules have the form $a \to w$, where a is a symbol and w is a multiset, catalytic rules have the form $ca \to cw$, where the symbol c is called the *catalyst*, and cooperative rules have no restrictions on the form of the left-hand side. These types of rules will be denoted by *ncoo* (*non-cooperative*), *pcat* (*purely catalytic*), and *coo* (*cooperative*); if both non-cooperative and catalytic rules are allowed, we write *cat* (*catalytic*).

If the P system has more than one membrane, each symbol on the right-hand side may have assigned a target where the symbol has to be sent after the application of the rule; the targets take into account the tree structure of the membranes:

here	the symbol stays in the membrane where the rule is applied;
out	the symbol is sent to the outer membrane, i.e., the membrane enclosing the membrane where the rule is applied;
in	the symbol is sent to an inner membrane, i.e., a membrane enclosed by the membrane where the rule is applied;
in_j	the symbol is sent to the inner membrane labeled by j.

3.3 Halting conditions

Besides the standard total halting with no (multi)set of rules being applicable any more to the current configuration, some more variants of halting conditions have been considered in the literature:

total halting (H)	the common halting strategy where the computation stops with no (multi)set of rules being applicable any more
unconditional halting (u)	the result of a computation can be taken from every configuration derived from the initial one (possibly only taking terminal results)
partial halting (h)	the set of rules R is partitioned into disjoint subsets R_1 to R_h, and a computation stops if there is no multiset of rules applicable to the current configuration which contains a rule from every set R_j, $1 \le j \le h$
halting with states (s)	the configuration with which a derivation may stop must fulfill a recursive condition (which corresponds with a *final state*)

The variant of unconditional halting was introduced in [7]. Partial halting, for example, was investigated in [3, 4, 14], using the membranes for partitioning the rules. Formal definitions for the halting conditions H, h, s can be found in [18].

For $\beta \in \{H, h, u, s\}$, we add the halting condition β in the description of the generated or accepted language, i.e., we then write $L_{\gamma,\delta,\beta}(\Pi)$, $\gamma \in \{gen, acc\}$. The same extension is made for the corresponding families of languages of multisets, i.e., for $n \ge 1$, we write $Y_{\gamma,\delta,\beta}OP_n(X)$. By default, β is understood to be the total halting H and then usually omitted in all these notations.

3.4 Flattening

As many variants of P systems can be *flattened* to only one membrane, see [13], we often may assume the simplest membrane structure of only one membrane which in effect reduces the P system to a multiset processing mechanism, and, observing that $f = 1$, in what follows we then will use the reduced notation

$\Pi = \left(V, T, w, R, \Longrightarrow_{\Pi,\delta}\right)$.

In case we use catalysts, we write

$\Pi = \left(V, C, T, w, R, \Longrightarrow_{\Pi,\delta}\right)$

with $C \subseteq (V \backslash T)$ denoting the set of catalysts.

For a one-membrane system, the definitions for the *language generated by* Π and the *language accepted by* Π using the derivation mode δ and the halting condition β can be written in an easier way; for example, with v_{T° denoting the terminal multiset in the multiset v, we have

$$L_{\text{gen,max}}(\Pi) = \left\{ v_{T^\circ} \mid w \overset{*}{\Longrightarrow}_{\Pi,\delta} v \wedge \neg\exists z : v \Longrightarrow_{\Pi,\delta} z \right\} \text{ and}$$

$$L_{\text{acc,max}}(\Pi) = \left\{ w_0 \in T^\circ \mid \exists v : \left(w w_0 \overset{*}{\Longrightarrow}_{\Pi,\delta} v \wedge \neg\exists z : v \Longrightarrow_{\Pi,\delta} z \right) \right\}.$$

The family of languages of multisets generated by one-membrane P systems of type X in the derivation mode δ and with the halting condition β is denoted by $Ps_{\text{gen},\delta,\beta}OP(X)$.

The family of languages of multisets accepted by one-membrane P systems of type X in the derivation mode δ and with the halting condition β is denoted by $Ps_{\text{acc},\delta,\beta}OP(X)$.

In the following, we will mainly focus on the generative case, and when writing $Ps_{\delta,\beta}OP(X)$ we by default will mean $Ps_{\text{gen},\delta,\beta}OP(X)$.

4 Some well-known results

In this section, we recall some well-known results, which usually are not stated in the compact form given here.

4.1 Non-cooperative rules

Using only non-cooperative rules leaves us on the level of semi-linear sets, as for the derivation with context-free rules (and non-cooperative rules correspond to those), the resulting derivation tree does not depend on an interpretation of a sequential or a parallel derivation of any kind. Moreover, context-free (string or multiset) languages are closed under projections, hence, taking (even only terminal) results out from a specific output membrane does not make any difference. Therefore, we may state the following result:

Theorem 1 *For any $Y \in \{N, Ps\}$ and any $n \geq 1$ as well as any derivation mode $\delta \in D_S \cup D_M$,*

$$Y_{\text{gen},\delta}OP_n(ncoo) = YREG.$$

Although P systems working in the maximally parallel derivation mode are a parallel mechanism, we cannot go beyond *PsREG*, see Theorem 1.

For example, the rule $a \to aa$ used in parallel very much reminds us of a 0*L* system, i.e., a Lindenmayer system of the simplest form, which, when starting from the axiom aa, yields the language $L_1 = \{a^{2^n} \mid n \geq 1\}$. In order to also get this language with P systems working in one of the maximally parallel derivation modes, we either need some control mechanism (see Sect. 5) or some other special halting condition (see Sect. 7).

4.2 The importance of using catalysts

If in a one-membrane system we only have one catalyst c and only catalytic rules assigned to c, then this corresponds to a sequential use of non-cooperative rules, which together with Theorem 1 yields the following result:

Theorem 2 *For any $Y \in \{N, Ps\}$ and any derivation mode $\delta \in D_S \cup D_M$,*

$$Y_{\text{gen},\delta}OP\left(pcat_1\right) = Y_{\text{gen,sequ}}\, OP\left(pcat_1\right) = Y_{\text{gen,sequ}}\, OP(ncoo) = YREG.$$

Even without additional control mechanisms, only two (three) catalysts are sufficient to obtain computational completeness for (purely) catalytic P systems using the derivation mode *max*, see [12]. In a more general way, the following results were already proved there:

Theorem 3 *For any $d \geq 1$ and any $k \geq d + 2$,*

$$Ps_{\text{acc,max}}OP\left(pcat_{k+1}\right) = Ps_{\text{acc,max}}OP\left(cat_k\right) = N^dRE.$$

The complexity of the construction, for all these derivation modes, has been considerably reduced since the original paper from 2005, for example, see [1, 5, 25], and [6].

Although not yet stated in [12], we mention that these results are also valid when replacing the derivation mode *max* by any other maximally parallel (set) derivation mode, i.e., for any δ in

$$\{max, max_{rules}, max_{objects}, smax, smax_{rules}, smax_{objects}\}.$$

The following theorem states the best results known so far with respect to the number of catalysts and the number of rules for catalytic P systems; the proof follows the one given in [6] for the set maximally parallel derivation modes.

Theorem 4 *For any register machine $M = (d, B, L_0, l_h, R)$, with $m \leq d$ being the number of decrementable registers, we can construct a catalytic P system*

$$\Pi = \left(V, C, T, w, R_1, \Longrightarrow_{\Pi,\delta}\right)$$

which works with any of the maximally parallel (set) derivation modes, i.e., with any δ in

$$\{max, max_{rules}, max_{objects}, smax, smax_{rules}, smax_{objects}\},$$

and simulates the computations of M such that

$$|R_1| \leq \texttt{ADD}^1(R) + 2 \times \texttt{ADD}^2(R) + 5 \times \texttt{SUB}(R) + 5 \times m + 1,$$

where $\texttt{ADD}^1(R)$ denotes the number of deterministic ADD*-instructions in R, $\texttt{ADD}^2(R)$ denotes the number of non-deterministic* ADD*-instructions in R, and $\texttt{SUB}(R)$ denotes the number of* SUB*-instructions in R.*

Proof We simulate a register machine $M = (d, B, l_0, l_h, R)$ by a catalytic P system Π, with $m \leq d$ being the number of decrementable registers.

For all d registers, n_i copies of the symbol o_i are used to represent the value n_i in register i, $1 \leq i \leq d$. For each of the m decrementable registers, we take a catalyst c_i and two specific symbols d_i, e_i, $1 \leq i \leq m$, for simulating SUB-instructions on these registers. For every $l \in B$, we use p_l, and also its variants $\bar{p}_l, \hat{p}_l, \tilde{p}_l$ for $l \in B_{\texttt{SUB}}$, where $B_{\texttt{SUB}}$ denotes the set of labels of SUB-instructions; w_0 stands for the additional input present at the beginning, for example, for the given input in case of accepting systems.

$$\begin{aligned}
\Pi =& \left(V, C, T, w, R_1, \Longrightarrow_{\Pi,\delta}\right),\\
w =& c_1 \dots c_m d_1 \dots d_m p_1 w_0,\\
V =& C \cup D \cup E \cup T \cup \{\#\} \cup \{p_l \mid l \in B\}\\
&\cup \{\bar{p}_l, \hat{p}_l, \tilde{p}_l \mid l \in B_{\texttt{SUB}}\},\\
C =& \{c_i \mid 1 \leq i \leq m\},\\
D =& \{d_i \mid 1 \leq i \leq m\},\\
E =& \{e_i \mid 1 \leq i \leq m\},\\
T =& \{o_i \mid 1 \leq i \leq d\},\\
R_1 =& \{p_j \to o_r p_k D_m, p_j \to o_r p_l D_m\\
&\quad \mid j : (\texttt{ADD}(r), k, l) \in R\}\\
&\cup \{p_j \to \hat{p}_j e_r D_{m,r}, p_j \to \bar{p}_j D_{m,r},\\
&\quad \hat{p}_j \to \tilde{p}_j D'_{m,r}, \bar{p}_j \to p_k D_m, \tilde{p}_j \to p_k D_m\\
&\quad \mid j : (\texttt{SUB}(r), k, l) \in R\}\\
&\cup \{c_r o_r \to c_r d_r, c_r d_r \to c_r, c_{r\oplus_m 1} e_r \to c_{r\oplus_m 1}\\
&\quad \mid 1 \leq r \leq m\},\\
&\cup \{d_r \to \#, c_r e_r \to c_r \# \mid 1 \leq r \leq m\}\\
&\cup \{\# \to \#\}.
\end{aligned}$$

We define $r\oplus_m 1 := r + 1$ for $r < m$ and $m\oplus_m 1 := 1$.

Usually, every catalyst $c_i, i \in \{1, \dots, m\}$, is kept busy with the symbol d_i using the rule $c_i d_i \to c_i$, as otherwise the symbols d_i would have to be trapped by the rule $d_i \to \#$, and the trap rule $\# \to \#$ then enforces an infinite non-halting computation. Only during the simulation of SUB-instructions on register r, the corresponding catalyst c_r is left free for decrementing or for zero-checking in the second step of the simulation, and in the decrement case both c_r and its "coupled" catalyst $c_{r\oplus_m 1}$ are needed to be free for specific actions in the third step of the simulation.

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

$$\begin{aligned}
D_m &= \prod_{i \in \{1,\dots,m\}} d_i,\\
D_{m,r} &= \prod_{i \in \{1,\dots,m\} \setminus \{r\}} d_i,\\
D'_{m,r} &= \prod_{i \in \{1,\dots,m\} \setminus \{r, r\oplus_m 1\}} d_i.
\end{aligned}$$

The HALT-instruction labeled l_h is simply simulated by not introducing the corresponding state symbol p_{l_h}, i.e., replacing it by λ, in all rules defined in R_1.

Each ADD-instruction $j : (\texttt{ADD}(r), k, l)$, for $r \in \{1, \dots, d\}$, can easily be simulated by the rules $p_j \to o_r p_k D_m$ and $p_j \to o_r p_l D_m$; in parallel, the rules $c_i d_i \to c_i, 1 \leq i \leq m$, have to be carried out, as otherwise the symbols d_i would have to be trapped by the rules $d_i \to \#$.

Each SUB-instruction $j : (\texttt{SUB}(r), k, l)$, is simulated as shown in the table listed below (the rules in brackets [and] are those to be carried out in case of a wrong choice):

Simulation of the SUB-instruction $j : (\texttt{SUB}(r), k, l)$ if	
Register r is not empty	Register r is empty
$p_j \to \hat{p}_j e_r D_{m,r}$	$p_j \to \bar{p}_j D_{m,r}$
$c_r o_r \to c_r d_r$ $[c_r e_r \to c_r \#]$	c_r should stay idle
$\hat{p}_j \to \tilde{p}_j D'_{m,r}$	$\bar{p}_j \to p_k D_m$
$c_r d_r \to c_r$ $[d_r \to \#]$	$[d_r \to \#]$
$\tilde{p}_j \to p_k D_m$	
$c_{r\oplus_m 1} e_r \to c_{r\oplus_m 1}$	

In the first step of the simulation of each instruction (ADD-instruction, SUB-instruction, and even HALT-instruction) due to the introduction of D_m in the previous step (we also start with that in the initial configuration), every catalyst c_r is kept busy by the corresponding symbol $d_r, 1 \leq r \leq m$. Hence, this also guarantees that the zero-check on register r works correctly enforcing $d_r \to \#$ to be applied, as in the case of a wrong choice two symbols d_r are present. □

Exactly the same construction as elaborated above can be used when allowing for $m + 2$ catalysts, with catalyst c_{m+1}

being used with the state symbols and catalyst c_{m+2} being used with the trap rules.

Yet for the purely catalytic case, only one additional catalyst c_{m+1} is needed to be used with all the non-cooperative rules, but in this case a slightly more complicated simulation of SUB-instructions is needed, see [25]), where for catalytic P systems

$$|R_1| \leq 2 \times \mathtt{ADD}^1(R) + 3 \times \mathtt{ADD}^2(R) + 6 \times \mathtt{SUB}(R) + 5 \times m + 1$$

and for purely for catalytic P systems

$$|R_1| \leq 2 \times \mathtt{ADD}^1(R) + 3 \times \mathtt{ADD}^2(R) + 6 \times \mathtt{SUB}(R) + 6 \times m + 1$$

is shown.

The simulation results established above hold true for register machines and their corresponding (purely) catalytic P systems for generating and accepting systems as well as even for systems computing functions or relations on natural numbers.

Many computational completeness results for variants of P systems are obtained by simulating register machines, which in fact means that a sequential machine has to be simulated by a parallel mechanism. Exactly, this feature of breaking down the parallelism to sequentiality is the main importance of using catalysts: when using a maximally parallel (set) derivation mode δ, for decrementing the number of a symbol o_r to carry out the decrement case of a SUB-instruction of the register machine, we cannot use the non-cooperative rule $o_r \to \lambda$; instead, we have to use the catalytic rule $co_r \to c$.

What happens in the case of two catalysts in purely catalytic P systems (and one catalyst in the case of catalytic P systems), is one of the most intriguing open problems in the area of P systems since long time, e.g., see [17], where it is shown that catalytic P systems with one catalyst can simulate partially blind register machines and partially blind counter automata.

With respect to the importance of using catalytic rules, the set derivation modes offer new opportunities, i.e., using specific control mechanisms they are not needed any more, as eliminating only one symbol o_r to carry out the decrement case of a SUB-instruction of a register machine now *can* be done by a non-cooperative rule $o_r \to \lambda$, because due to the set restriction, this rule is not applied more than once.

5 Control mechanisms

To reduce the number of catalysts needed for obtaining computational completeness, specific control mechanisms can be used. Some of these control mechanisms are considered in this section. For example, label selection or control languages allow for using only one catalyst (two catalysts) in (purely) catalytic P systems for getting computational completeness, for instance, see [6, 11, 15, 16]. With target selection and maximally parallel set derivation modes, catalysts can even be avoided completely, only non-cooperative rules are needed.

For all the control mechanisms described in this section, as a special example, we will show how the $0L$ language $L_1 = \{a^{2^n} \mid n \geq 1\}$ can be generated using the maximally parallel derivation mode.

5.1 P systems with label selection

For all the variants of P systems of type X, we may consider labeling all the rules in the sets $R_1, \ldots, R_m$ in a one-to-one manner by labels from a set H and taking a set W containing subsets of H. In any derivation step of a *P system with label selection* Π, we first select a set of labels $U \in W$ and then, in the given derivation mode, we apply a non-empty multiset R of rules such that all the labels of these rules from R are in U.

Example 1 Consider the one-membrane P system

$$\Pi = (V = \{A, a\}, T = \{a\}, w = AA, R = \{r_1 : A \to AA, r_2 : A \to a\}, W = \{\{r_1\}, \{r_2\}\}, \Longrightarrow_{\Pi,\max}).$$

with the labeled rules $r_1 : A \to AA$ and $r_2 : A \to a$; only one of these can be used according to the sets of labels in W. Using r_1 in $n-1$ derivation steps and finally using r_2 yields a^{2^n}, for any $n \geq 1$, i.e., we get $N_{\mathrm{gen},\max}(\Pi) = L_1$, where $L_1 = \{a^{2^n} \mid n \geq 1\}$.

The families of sets $Y_{\gamma,\delta}(\Pi), Y \in \{N, Ps\}, \gamma \in \{gen, acc\}$, and $\delta \in D_M \cup D_S$ computed by P systems with label selection with at most m membranes and rules of type X are denoted by $Y_{\gamma,\delta}OP_m(X, ls)$.

Theorem 5 *$Y_{\gamma,\delta}OP(cat_1, ls) = Y_{\gamma,\delta}OP(pcat_2, ls) = YRE$ for any $Y \in \{N, Ps\}, \gamma \in \{gen, acc\}$, and any maximally parallel (set) derivation mode δ,*

$$\delta \in \{max, max_{rules}, max_{objects}, smax, smax_{rules}, smax_{objects}\}.$$

The proof given in [16] for the maximally parallel mode *max* can be taken over for the other maximally parallel (set) derivation modes word by word; the only difference again is that in set derivation modes, in non-successful computations where more than one trap symbol # has been generated, the trap rule # → # is only applied once.

5.2 Controlled P systems and time-varying P systems

Another method to control the application of the labeled rules is to use control languages (see [19] and [2]).

In a *controlled P system* Π, in addition we use a set H of labels for the rules in Π, and a string language L over 2^H (each subset of H represents an element of the alphabet for L) from a family FL. Every successful computation in Π has to follow a control word $U_1 \dots U_n \in L$: in derivation step i, only rules with labels in U_i are allowed to be applied (in the underlying derivation mode, for example, *max* or *smax*), and after the n-th derivation step, the computation halts; we may relax this end condition, i.e., we may stop after the i-th derivation for any $i \leq n$, and then we speak of *weakly controlled P systems*. If $L = \left(U_1 \dots U_p\right)^*$, Π is called a *(weakly) time-varying P system*: in the computation step $pn + i, n \geq 0$, rules from the set U_i have to be applied; p is called the *period*.

Example 2 Consider the one-membrane P system

$$\Pi = (V = \{A, a\}, T = \{a\}, w = AA, R = \{r_1 : A \to AA, r_2 : A \to a\}, L = \{r_1\}^*\{r_2\}, \Longrightarrow_{\Pi,\max})$$

with the labeled rules $r_1 : A \to AA$ and $r_2 : A \to a$. Using the control word ${r_1}^{n-1}r_2$ means using r_1 in $n-1$ derivation steps and finally using r_2, thus yielding a^{2^n}, for any $n \geq 1$, i.e., as in Example 1, we get $N_{\text{gen,max}}(\Pi) = L_1$.

As now we do not have to distinguish between non-terminal and terminal symbols due to the use of control words, the same result can be obtained by the much simpler system

$$\Pi' = (V = \{a\}, T = \{a\}, w = aa, R = \{r_1 : a \to aa\}, L = \{r_1\}^*, \Longrightarrow_{\Pi',\max})$$

again yielding $N_{\text{gen,max}}(\Pi') = L_1$.

The family of sets $Y_{\gamma,\delta}(\Pi)$, $Y \in \{N, Ps\}$, computed by (weakly) controlled P systems and (weakly) time-varying P systems with period p, with at most m membranes and rules of type X as well as control languages in FL is denoted by $Y_{\gamma,\delta}OP_m(X, C(FL))$ $(Y_{\gamma,\delta}OP_m(X, wC(FL)))$ and $Y_{\gamma,\delta}OP_m\left(X, TV_p\right)$ $(Y_{\gamma,\delta}OP_m\left(X, wTV_p\right))$, respectively, for $\gamma \in \{gen, acc\}$ and $\delta \in D_M \cup D_S$.

Theorem 6 $Y_{\gamma,\delta}OP\left(cat_1, \alpha TV_6\right) = Y_{\gamma,\delta}OP\left(pcat_2, \alpha TV_6\right) = YRE$, *for any* $\alpha \in \{\lambda, w\}$, $Y \in \{N, Ps\}$, $\gamma \in \{gen, acc\}$, *and*

$$\delta \in \left\{max, max_{rules}, max_{objects}, smax, smax_{rules}, smax_{objects}\right\}.$$

The proof given in [16] for the maximally parallel mode *max* again can be taken over for the other maximally parallel (set) derivation modes word by word, e.g., see [6].

5.3 Target selection

In P systems with target selection, all objects on the right-hand side of a rule must have the same target, and in each derivation step, for each region a (multi)set of rules—non-empty if possible—having the same target is chosen. In [6], it was shown that for P systems with target selection (abbreviated *ts*) in the derivation mode *smax* ***no*** catalyst is needed any more, and with $smax_{rules}$, we even obtain a deterministic simulation (indicated by the abbreviation detacc) of deterministic register machines:

Theorem 7 *For any* $Y \in \{N, Ps\}$,

$$Y_{\text{gen ,smax}}\, OP(ncoo, ts) = YRE.$$

Theorem 8 *For any* $Y \in \{N, Ps\}$,

$$Y_{\text{detacc,smax}_{rules}}\, OP(ncoo, ts) = YRE.$$

In contrast to all the other variants of P systems, P systems with target selection really take advantage of the membrane structure, no flattening is used or even reasonable. In that sense, this variant of P systems reflects the spirit of membrane systems with a non-trivial membrane structure in the best way.

Example 3 Consider the two-membrane P system

$$\Pi = (V = \{a\}, T = \{a\}, \mu = [\,[\,]_2\,]_1, w_1 = aa, w_2 = \lambda, R_1 = \{a \to aa, a \to (a, in)\}, R_2 = \emptyset, f = 2, \Longrightarrow_{\Pi,\max})$$

with the rule $a \to aa$ having target *here* and the rule $a \to (a, in)$ having target *in*; only one of these two rules can be used in one derivation step according to the condition of target selection. Using $a \to aa$ in $n-1$ derivation steps in the skin membrane and finally using $a \to (a, in)$ yields a^{2^n} in the elementary membrane $[\,]_2$, for any $n \geq 1$, i.e., we again get $N_{\text{gen,max}}(\Pi) = L_1$.

6 The strangeness of minimal parallelism

There is another derivation mode known from literature, which has two possible basic definitions, but these two variants unfortunately do not yield the same results.

Following the definition given in [18], for the minimally parallel derivation mode (*min*), we need an additional feature for the set of rules R used in the overall P system, i.e., we consider a partitioning θ of R into disjoint subsets R_1 to R_h. Usually, this partitioning of R may coincide with a specific assignment of the rules to the membranes. We

observe that this partitioning θ may, but need not be the same as the partitioning η used for partial halting.

There are now several possible interpretations of this minimally parallel derivation mode which in an informal way can be described as applying multisets such that from every set $R_j, 1 \leq j \leq h$, at least one rule—if possible—has to be used (e.g., see [8]). Yet this *if possible* allows for two possible interpretations:

Minimal parallelism as a restriction of asyn As defined in [18], we start with a multiset R' of rules from $Appl(\Pi, C, asyn)$ and only take it if it cannot be extended to a multiset R' of rules from $Appl(\Pi, C, asyn)$ by some rule from a set R_j from which so far no rule is in R'.

Minimal parallelism as an extension of smax We start with a set R' of rules from $Appl(\Pi, C, smax_\theta)$, where the notion $smax_\theta$ indicates that we are using *smax* with respect to the partitioning of R into the subsets R_1 to R_h, and then possibly extend it to a multiset R'' of rules from $Appl(\Pi, C, asyn)$ which contains R'. This definition finally was used in [23] without using the notion *smax*, because at the moment when this handbook was written the notion of maximally parallel set derivation modes had not been invented yet. Moreover, the use of the notion *smax* so far was restricted to the discrete topology, where every rule formed its own set R_j; whereas for $smax_\theta$, the condition is fulfilled if *one* of the rules in the R_j is used if possible.

Example 4 Consider the one-membrane P system working in the *min*-mode

$$\Pi = \left(V = \{a, b\}, T = \{b\}, w = aa, R = R_1 \cup R_2, \Longrightarrow_{\Pi, min}\right)$$

with $R_1 = \{a \to bb\}$ and $R_2 = \{a \to bbb\}$ being the partitions of $R = R_1 \cup R_2$.

Starting from *smax*, we get only one set of rules, i.e., $R' = \{a \to bb, a \to bbb\}$, whose application yields the result b^5.

In the case of starting with *asyn*, we may use one of the two rules twice, thus also getting the results b^4 and b^6.

Hence, when two rules are competing for the same objects, the results obtained with the two different definitions may be different, where the set of results obtained when using the first definition will always include the results obtained by the second definition.

The condition that the sets $R_j, 1 \leq j \leq h$, have to be disjoint may be alleviated, for example, see [4].

6.1 The derivation mode min_1

A special variant of the minimally parallel derivation mode, with the sets $R_j, 1 \leq j \leq h$, not being required to be disjoint, is the mode min_1, which in fact means that we stay with $smax_\theta$. Now let θ_k denote a partioning θ with k sets of rules. As an interesting result, we then get the interpretation of a purely catalytic P system using *max* as a P system using min_1 with the partitioning $R_j, 1 \leq j \leq k$, where R_j is the set of non-cooperative rules $a \to u$ representing the corresponding catalytic rules $c_j a \to c_j u$. Using such a partitioning θ_k in k sets of rules corresponding to the sets of rules associated with the k catalysts, we obtain the following result:

Theorem 9 *For any $d \geq 1$ and any $k \geq d + 3$,*

$$Ps_{\mathrm{acc},\min_1} OP\left(ncoo, \theta_k\right) = Ps_{\mathrm{gen},\min_1} OP\left(ncoo, \theta_3\right) = N^d RE.$$

6.2 Minimal parallelism with all applicable sets

There is an even stranger variant for minimal parallelism already defined in [18]:

To a configuration C, we can only apply a multiset of rules which contains at least one rule from each $R_j, 1 \leq j \leq h$, that contains a rule applicable to C, i.e., we take all possible multisets R' from $Appl(\Pi, C, asyn)$ which also fulfill the condition that $R' \cap R_j \neq \emptyset$ provided $Appl(\Pi, C, asyn) \cap R_j \neq \emptyset$, for all $1 \leq j \leq h$.

This derivation mode is abbreviated $all_{aset}min$ in [18] and used under the notion *amin* in [4].

Example 5 Consider the one-membrane P system from Example 4, now working in the *amin*-mode,

$$\Pi = \left(V = \{a, b\}, T = \{b\}, w = aa, R = R_1 \cup R_2, \Longrightarrow_{\Pi, amin}\right)$$

with $R_1 = \{a \to bb\}$ and $R_2 = \{a \to bbb\}$.

As both the rule from R_1 and the rule from R_2 are applicable, the only (multi)set of rules applicable to the configuration aa is the same as that one when starting from $smax$, i.e., $R' = \{a \to bb, a \to bbb\}$, whose application yields the result b^5.

Yet if we take $w = a$ instead, then still both the rule from R_1 and the rule from R_2 are applicable, but there are not enough resources of symbols a for applying both rules, hence, no derivation step is possible in this case with the derivation mode $amin$. On the other hand, with the first two variants of the minimally parallel derivation mode, in both cases we may either apply $a \to bb$ or $a \to bbb$, thus getting bb and bbb, respectively.

Again, we observe that the results with different definitions of the minimally parallel derivation mode may be different when two rules are competing for the same object(s).

7 Halting conditions

As already mentioned, P systems working in the maximally parallel derivation mode at first sight look like (E)0L systems. Only the total halting condition completely destroys this similarity which looks so obvious at first sight. Yet, this connection between P systems working in the maximally parallel derivation mode and (E)0L systems can be shown when using unconditional halting, see [7].

Besides unconditional halting, in this section we will also discuss some results for partial halting and halting with states. In each case, as in Sect. 5, we will show how to obtain the special multiset language $L_1 = \{a^{2^n} \mid n \geq 1\}$.

7.1 Unconditional halting

Example 6 Consider the one-membrane P system

$$\Pi = (V = \{a\}, T = \{a\}, w = aa, R = \{a \to aa\}, \Longrightarrow_{\Pi,\max,u})$$

with the single rule $a \to aa$; with every application of this rule the number of symbols a is doubled, i.e., after $n-1$ derivation steps, $n \geq 1$, we get a^{2^n}, i.e., we obtain $N_{\text{gen,max,u}}(\Pi) = L_1$.

According to the results shown in [7], the following results hold true, if we use extended systems (indicated by the additional symbol E) and only take results from the output membrane which are terminal:

Theorem 10 *For any $Y \in \{N, Ps\}$ and any $m \geq 1$,*

$$Y_{\text{gen},\delta,u}EOP_m(ncoo) = YE0L,$$

for any maximally parallel derivation mode δ,

$$\delta \in \{max, max_{rules}, max_{objects}\}.$$

If we do not use extended systems, i.e., $V = T$, we immediately obtain the following:

Corollary 1 *For any $Y \in \{N, Ps\}$,*

$$Y_{\text{gen},\delta,u}OP_1(ncoo) = Y0L,$$

for any maximally parallel derivation mode δ,

$$\delta \in \{max, max_{rules}, max_{objects}\}.$$

These results now show the—somehow expected—correspondence between the two parallel mechanisms *P systems* and *Lindenmayer systems*.

We finally mention that with unconditional halting, considering acceptance would not make any sense, because according to the standard definition of accepting P systems, in any case they would accept every input.

7.2 Partial halting

Partial halting allows us to stop a derivation as soon as some specific symbols are not present any more:

Example 7 Consider the one-membrane P system

$$\Pi = (V = \{a, s\}, T = \{a\}, w = as, R_1 \cup R_2, \Longrightarrow_{\Pi,\max,h}),$$

where $R_1 = \{a \to aa\}$ and $R_2 = \{s \to s, s \to \lambda\}$ are the two partitions of the rule set $R = \{a \to aa, s \to s, s \to \lambda\}$.

As long as one of the rules from R_2 can be applied to the symbol s, the symbols a are doubled as usual by the rule $a \to aa$ from R_1. Using $s \to s$ in $n-1$ derivation steps, $n \geq 1$, and finally applying $s \to \lambda$, we get a^{2^n}; hence, $N_{\text{gen,max,h}}(\Pi) = L_1$.

Some interesting results for the partial halting may be looked up in [3, 4, 14].

7.3 Halting with states

In general, speaking of states reminds us of mechanisms like register machines; there a computation halts when the halt instruction $l_h : HALT$ is applied. In simulations of register machines by P systems, the computation often is made halting by applying the final rule $l_h \to \lambda$, provided no trap rules are still applicable. When l_h disappears this means that no

instruction label appears any more in the configuration of the simulating P system; such a condition checking for the absence (or presence) of specific symbols in a given configuration is computable, i.e., decidable, and therefore is a condition we can use for halting with states (which ironically in this case means the absence of state symbols).

Example 8 Consider the one-membrane P system

$$\Pi = (V = \{a, s\}, T = \{a\}, w = as, R = \{a \to aa, s \to s, s \to \lambda\}, \Longrightarrow_{\Pi,\max,s}),$$

which uses the same ingredients as the one considered in Example 7, but instead of partial halting now uses the condition that a computation halts if no symbol s is present any more, which gives the same computations as for the P system in Example 7, with the only difference that the computations halt because of s having been deleted. Thus, we obtain $N_{\text{gen,max,s}}(\Pi) = L_1$.

8 Conclusion

In this paper, the effects of using different derivation modes on the computing power of many variants of hierarchical P systems have been illustrated. Especially, some differences between the maximally parallel derivation modes and the maximally parallel set derivation modes have been exhibited. We have also given an overview on some control mechanisms used for P systems. Moreover, we have discussed the effect of using different halting conditions such as unconditional and partial halting.

Many more relations between derivation modes and halting conditions as well could have been discussed, but this would have gone much beyond such a normal article.

Acknowledgements Open access funding provided by TU Wien (TUW). Many of the ideas for this paper came up in the inspiring atmosphere of the Brainstorming Week on Membrane Computing in Sevilla 2019 and even in some previous years, and they are based on many discussions with Artiom Alhazov, Sergiu Ivanov, and Sergey Verlan, but also other colleagues from the P community, especially with Gheorghe Păun. Moreover, the helpful comments of the referees are gratefully acknowledged.

References

1. Alhazov, A., & Freund, R. (2015). Variants of small universal P systems with catalysts. *Fundamenta Informaticae*, *138*(1–2), 227–250. https://doi.org/10.3233/FI-2015-1209.
2. Alhazov, A., Freund, R., Heikenwälder, H., Oswald, M., Rogozhin, Yu., & Verlan, S. (2013). Sequential P systems with regular control. In: E. Csuhaj-Varjú, M. Gheorghe, G. Rozenberg, A. Salomaa, Gy. Vaszil (Eds.), *Membrane Computing—13th International Conference, CMC 2012*, Budapest, Hungary, August 28–31, 2012, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 7762 (pp. 112–127). Berlin: Springer. https://doi.org/10.1007/978-3-642-36751-9_9.
3. Alhazov, A., Freund, R., Oswald, M., & Verlan, S. (2007). Partial halting in P systems using membrane rules with permitting contexts. In: J. Durand-Lose, M. Margenstern (Eds.), *Machines, Computations, and Universality* (pp. 110–121). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-74593-8_10.
4. Alhazov, A., Freund, R., Oswald, M., & Verlan, S. (2009). Partial halting and minimal parallelism based on arbitrary rule partitions. *Fundamenta Informaticae*, *91*(1), 17–34. https://doi.org/10.3233/FI-2009-0031.
5. Alhazov, A., Freund, R., & Sosík, P. (2015). Small P systems with catalysts or anti-matter simulating generalized register machines and generalized counter automata. *The Computer Science Journal of Moldova*, *23*(3), 304–328. http://www.math.md/publications/csjm/issues/v23-n3/11980/.
6. Alhazov, A., Freund, R., & Verlan, S. (2017). P systems working in maximal variants of the set derivation mode. In: A. Leporati, G. Rozenberg, A. Salomaa, C. Zandron (Eds.), *Membrane Computing—17th International Conference, CMC 2016*, Milan, Italy, July 25–29, 2016, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 10105 (pp. 83–102). Berlin: Springer. https://doi.org/10.1007/978-3-319-54072-6_6.
7. Beyreder, M., & Freund, R. (2009). Membrane systems using noncooperative rules with unconditional halting. In: D.W. Corne, P. Frisco, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), *Membrane Computing* (pp. 129–136). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-540-95885-7_10.
8. Ciobanu, G., Pan, L., Păun, Gh, & Pérez-Jiménez, M. (2007). P systems with minimal parallelism. *Theoretical Computer Science*, *378*(1), 117–130. https://doi.org/10.1016/j.tcs.2007.03.044.
9. Dassow, J., & Păun, Gh. (1989). Regulated rewriting in formal language theory. Berlin: Springer. https://www.springer.com/de/book/9783642749346.
10. Freund, R. (2005). P systems working in the sequential mode on arrays and strings. *International Journal of Foundations of Computer Science*, *16*(4), 663–682. https://doi.org/10.1142/S0129054105003224.
11. Freund, R. (2013). Purely catalytic P systems: Two catalysts can be sufficient for computational completeness. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin (Eds.), *CMC14 Proceedings—the 14th international conference on membrane computing*, Chişinău, August 20–23, 2013 (pp. 153–166). Institute of Mathematics and Computer Science, Academy of Sciences of Moldova. http://www.math.md/cmc14/CMC14_Proceedings.pdf.
12. Freund, R., Kari, L., Oswald, M., & Sosík, P. (2005). Computationally universal P systems without priorities: Two catalysts are sufficient. *Theoretical Computer Science*, *330*(2), 251–266. https://doi.org/10.1016/j.tcs.2004.06.029.
13. Freund, R., Leporati, A., Mauri, G., Porreca, A.E., Verlan, S., & Zandron, C. (2014). Flattening in (tissue) P systems. In: A. Alhazov, S. Cojocaru, M. Gheorghe, Yu. Rogozhin, G. Rozenberg, A. Salomaa (Eds.), *Membrane Computing, Lecture Notes*

in Computer Science, vol. 8340 (pp. 173–188). Berlin: Springer. https://doi.org/10.1007/978-3-642-54239-8_13.
14. Freund, R., & Oswald, M. (2007). Partial halting in P systems. *International Journal of Foundations of Computer Science*, *18*(6), 1215–1225. https://doi.org/10.1142/S0129054107005261.
15. Freund, R., & Oswald, M. (2013). Catalytic and purely catalytic P automata: control mechanisms for obtaining computational completeness. In: S. Bensch, F. Drewes, R. Freund, F. Otto (Eds.), *Fifth Workshop on Non-Classical Models for Automata and Applications*—NCMA 2013, Umeå, Sweden, August 13–August 14, 2013, Proceedings, *http://books@ocg.at*, vol. 294 (pp. 133–150). Wien: Österreichische Computer Gesellschaft.
16. Freund, R., & Păun, Gh. (2013). How to obtain computational completeness in P systems with one catalyst. In: T. Neary, M. Cook (Eds.), *Proceedings Machines, Computations and Universality 2013, MCU 2013,* Zürich, Switzerland, September 9–11, 2013, *EPTCS*, vol. 128 (pp. 47–61). https://doi.org/10.4204/EPTCS.128.13
17. Freund, R., & Sosík, P. (2015). On the power of catalytic P systems with one catalyst. In: G. Rozenberg, A. Salomaa, J.M. Sempere, C. Zandron (Eds.), *Membrane Computing—16th International Conference, CMC 2015, Valencia, Spain, August 17–21, 2015, Revised Selected Papers, Lecture Notes in Computer Science*, vol. 9504 (pp. 137–152). Berlin: Springer. https://doi.org/10.1007/978-3-319-28475-0_10.
18. Freund, R., & Verlan, S. (2007). A formal framework for static (tissue) P systems. In: G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.), *Membrane Computing, Lecture Notes in Computer Science*, vol. 4860 (pp. 271–284). Berlin: Springer. https://doi.org/10.1007/978-3-540-77312-2_17.
19. Krithivasan, K., Păun, Gh, & Ramanujan, A. (2014). On controlled P systems. *Fundamenta Informaticae*, *131*(3–4), 451–464. https://doi.org/10.3233/FI-2014-1025.
20. Minsky, M. L. (1967). *Computation, finite and infinite machines*. Englewood Cliffs, NJ: Prentice Hall.
21. Păun, Gh. (2000). Computing with membranes. *Journal of Computer and System Sciences*, *61*(1), 108–143. https://doi.org/10.1006/jcss.1999.1693.
22. Păun, Gh. (2002). *Membrane computing: An introduction*. Berlin: Springer. https://doi.org/10.1007/978-3-642-56196-2.
23. Păun, Gh, Rozenberg, G., & Salomaa, A. (Eds.). (2010). *The Oxford Handbook of Membrane Computing*. Oxford: Oxford University Press.
24. Rozenberg, G., Salomaa, A. (Eds.), (1997). Handbook of Formal Languages. Berlin: Springer. https://doi.org/10.1007/978-3-642-59136-5.
25. Sosík, P., & Langer, M. (2016). Small (purely) catalytic P systems simulating register machines. *Theoretical Computer Science*, *623*, 65–74. https://doi.org/10.1016/j.tcs.2015.09.020.
26. The P Systems Website. http://ppage.psystems.eu/.

Rudolf Freund works at the Institute for Logic and Computation, TU Wien, Austria. His research interests include Theory of Computation, Computing in Mathematics, Natural Science, Engineering and Medicine, and Artificial Intelligence. Currently he especially works in the area of Membrane Computing.