Journal of Membrane Computing (2023) 5:170–181
https://doi.org/10.1007/s41965-023-00127-8

RESEARCH PAPER

On maximal parallel application of rules in rewriting P systems

Claudio Zandron[1]

Received: 11 May 2023 / Accepted: 3 August 2023 / Published online: 24 August 2023

Abstract
In rewriting P systems, that is P systems using structured strings instead of atomic symbols, rules can be applied in parallel on all strings, but a single rule at a time can be applied on each string. Nonetheless, parallel application of rules also on each string has been considered in various works. This leads to possible application of rules with conflicting target indications on the same string, and different strategies have been considered to face this problem; relations among different classes of languages generated in this way have been investigated in the literature. We continue the investigation on this subject, by highlighting some relations among different classes of maximally parallel rewriting P systems by means of direct simulations. The advantages of such simulations are highlighted, by showing how theoretical results concerning one such type of systems can immediately be adapted to the corresponding simulating systems.

Keywords Membrane computing · Parallel rewriting · Lindenmayer system · Matrix grammars · Chomsky grammars

1 Introduction

Membrane systems (or P systems) have been defined in [43] as a new computing model of a biochemical inspiration. In the basic model, several membranes are embedded, in a hierarchical way, in an external membrane, that is called skin. The membranes define regions containing atomic objects which, in the basic variant, are represented using symbols from a finite alphabet. At each computation step, objects are processed in a maximally parallel way by means of rewriting rules; then, evolved objects are eventually communicated to inner or outer regions, according to some target indications associated with each rewriting rule [45].

Various research works related to P systems have appeared, concerning their computing power [20, 43] or the computing power of some variants with limited features, like in [3, 54]. Such systems have also been considered to study computational complexity aspects for a variant where membranes can be created by division [44], resembling the process of cell mitosis. These studies concern both time complexity aspects [30–34, 41, 49, 51] as well as space complexity [1, 2, 4, 47, 48, 50, 55, 58]. A recent survey on this subject can be found in [53].

Crossing of membrane systems with other kind of systems have also been proposed (see, e.g., population P systems [17] or water based computing [25]), as well as application of Membrane systems to computational problems (we refer the reader to [12, 15, 18, 38, 40, 61] for some examples and details), or to more general problems (some recent examples can be found in [14, 21, 26, 27, 35, 46, 57, 60]).

The use of membrane systems to model different processes has also been considered in many papers appeared in the literature. For example, an epidemiological model developed in the context of the fight against SARS-CoV-2 has been presented in [5], while [23] models the vertical migration of zooplankton in shallow and deep water, just to cite some very recently appeared works.

In this work, we consider a variant of Membrane systems named Rewriting P systems [22, 36, 37, 43, 59] (or RP systems, for short): in such a variant, objects are described by finite strings over a finite alphabet. The rules allowing a string to evolve are context-free rewriting rules.

This variant has been investigated under different aspects. For instance, in [13] systems of this type where communication is controlled according to the contents of the strings are considered. More recently, the use of splicing rules in RP systems has been the subject of other works like [42], while systems using sets of pictures made

✉ Claudio Zandron
claudio.zandron@unimib.it

1 Università degli Studi di Milano-Bicocca Dipartimento di Informatica, Sistemistica e Comunicazione, Viale Sarca 336, 20126 Milan, Italy

of symbols in a two dimensional lattice (array languages) have been considered in [16, 56]. In [39] authors consider the use the use of such systems applied to chain code picture generation.

In particular, we concentrate on the computational properties of *maximally parallel* RP systems, that is rewriting P systems where, as it happens for standard P systems, all strings evolve in parallel and all regions evolve in parallel but, moreover, at each computation step all rules that can be applied simultaneously to the same string must be applied. Other kinds of parallelism can also be considered: we recall some of them in the last section (to highlight some research topics on this subject). In this kind of systems, when considering parallel application of rules also on single strings, a problem arises: in fact, different rules applied on a single string in a single computing step can have different target membranes. In this case, it is not immediately clear in which region the resulting string must be communicated, and some strategies must then be considered.

Three main such strategies, considered in the literature, are the following:

1. In Krishna–Rama Parallel Rewriting P systems (KRPRP systems) [29], when a conflict occurs the string is moved by choosing the region indicated by the target occurred *exactly once* among the applied rules (if any). In case more than one target has occurred once, then a nondeterministically choice occurs. If no target have occurred exactly once, then the string is blocked.
2. In P systems with deadlock [6, 7, 9], when different rules having different target indications are applied on a single string at the same time, a *deadlock state* is obtained: the string is blocked, and it will not be processed anymore by any other rules.
3. In P systems without target conflicts [8], the situation is avoided a priori: only set of rules with the same target indication can be applied at the same time on the same string.

Some partial results concerning the computing power of such systems have already been presented in the literature [6–9, 29], by comparing such systems with classic systems from the formal languages area of research.

In this work, we present various new results regarding the relationships between parallel rewriting P systems achieved through direct simulations. These simulations involve systems that employ different strategies to resolve conflicts. Specifically, we demonstrate:

- How to simulate a parallel RP system with Deadlock using a KRPRP system
- How to simulate a parallel RP system without target conflicts using a parallel RP system with Deadlock

Such direct simulations between systems having different features offer several advantages. Firstly, an advantage of employing direct simulations between systems lies in their potential practical implementations. For instance, if we were to implement these systems using a biological medium, certain operations that are theoretically feasible might become practically restricted, at least in an easy manner. By leveraging the simulation, one can design a computational system based on a specific theoretical model and then translate it into another model for practical implementation, while taking into account the limitations imposed by the chosen implementation medium.

Another advantage concerns theoretical results pertaining to properties of the simulated system can be directly applied to the simulating system, leveraging the relationships between the two systems identified through the direct simulation. To illustrate this aspect, in particular, we provide results concerning the relationships between parallel rewriting P systems with deadlock and other systems such as Lindenmayer systems, Matrix grammars, and classical Chomsky grammars. We then report the corresponding results that can be obtained exploiting the direct simulation provided before using KRPRP systems. For instance, we prove that parallel rewriting P systems with deadlock using a single membrane are capable of generating all context-free languages, including some non-context-free languages. By utilizing the outcome of the direct simulation, we can readily establish that the same languages can be generated by KRPRP systems using two membranes of depth one. Considering another example, we also recall a result from [11], stating that various classes of languages generated by different Lindenmayer systems can be generated by parallel rewriting P systems with deadlock using 4 membranes in a structure of depth three. An immediate consequence of the provided simulations is that these results (and, similarly, all results related to such systems) can be immediately translated to state that the same classes of languages can be generated by KRPRP systems using eight membranes arranged in a structure of depth four.

The rest of the paper is organized as follows. In Sect. 2 we recall main definitions used in the rest of the paper. In Sect. 3 we describe relations among classes of parallel rewriting P systems using different ways of solving target conflicts, by means of direct simulations between systems. In Sect. 4 we provide some results considering specific languages to highlight some relations among parallel RP systems, Lindenmayer systems, Matrix grammars, and Chomsky grammars. Finally, in Sect. 5 we draw some conclusions and describe some open problems related to this work.

2 Maximally parallel rewriting P systems

In this section, we recall main definitions of P systems and of maximally parallel rewriting P systems.

A *membrane structure* μ is obtained by embedding, in a hierarchical way, various membranes which are surrounded by a unique membrane, that is called *skin*. A membrane structure can be described in various ways, for instance by a string of matching parentheses, where each pair identifies a specific membrane. another way of describing such a structure is by means of a tree graph; the height of this tree is called the *depth* of the corresponding membrane structure.

The membranes in a membrane structure identify regions, that is the space delimited by a membrane and by the membranes that are immediately inside it. If a membrane does not contain other membranes inside, then it is called *elementary*.

By using symbols or strings over an alphabet V, we identify *objects*; in standard P systems, one usually considers *multisets* of objects. In parallel rewriting P systems, the subject of this work, only strings are considered: in each region $i = 0, 1, \dots, n$ of μ we consider multisets of finite support over V. In other words, there is a map $M_i : V^* \to \mathbf{N}$ where $M_i = \{(x_1, M_i(x_1)), \dots, (x_p, M_i(x_p))\}$, for some $x_k \in V^+$ such that $M_i(x_k) > 0\ \forall k = 1, \dots, p$.

Evolution rules of the systems are context free rewriting rules of the following form: $a \to \alpha(tar)$, where $a \in V, \alpha \in V^*$, and $tar \in \{here, out, in\}$ denotes the target membrane. After the application of an evolution rule, the object can be sent to another membrane, according to the target indication: when $tar = here$, the object stays in the same region; when $tar = out$, the object is sent out from the region where the rule was applied; when $tar = in$, the object is sent to one (nondeterministically chosen) of the membranes that are immediately inside the actual region.

We can now formally define a parallel rewriting P system.

Definition 1 A *parallel rewriting P system* of degree $n + 1$ is a construct

$$\Pi = (V, T, \mu, M_0, \dots, M_n, (R_0, \rho_0), \dots, (R_n, \rho_n)),$$

where:

1. V denotes the alphabet of the system;
2. $T \subseteq V$ denotes the terminal alphabet;
3. μ denotes a membrane structure, having $n + 1$ membranes injectively labeled by means of numbers in $\{0, 1, \dots, n\}$;
4. $M_0, \dots, M_n$ are multisets over the alphabet V; they represent the strings initially present in the corresponding regions $0, 1, \dots, n$;
5. $R_0, \dots, R_n$ denotes finite sets of *evolution rules* associated with each region of μ; the rules are of the form $a \to \alpha(tar)$, where $a \in V, \alpha \in V^*, tar \in \{here, out, in\}$;
6. $\rho_0, \dots, \rho_n$ are partial order relations over $R_0, \dots, R_n$.

A configuration of the system at a specific time step is defined by the multisets of objects associated with each region together with the membrane structure. The initial configuration is the $(n + 2)$-tuple $C_0 = (\mu, M_0, \dots, M_n)$.

A transition from a configuration $C_t = (\mu, M_0^t, \dots, M_n^t)$ to a configuration $C_{t+1} = (\mu, M_0^{t+1}, \dots, M_n^{t+1})$ is obtained by applying the rules present in the regions, following the prescriptions of the specific parallel RP system.

A *computation* is a sequence of transitions. A computation *halts* when, in the current configuration, no rule can be further applied; in case at least one rule can be applied forever, then we have a *non-halting* computation.

In this work, we will consider *extended* RP systems: the *output* is the set of strings containing only symbols of the terminal alphabet T sent out of the system during a halting computation. If a string sent out from the system contains non-terminal symbols, then it does not contribute to the output. If a computation never halts, then it produces no output.

2.1 Krishna–Rama parallel P systems

In a Krishna–Rama parallel P system [28, 29], at each computation step all rules in the system are applied in a nondeterministic and maximally parallel manner on all strings, in all the regions. Then, the obtained string is sent to the region that has occurred *exactly once* among all targets defined in the applied rules. In case different targets have occurred exactly once, one of them is chosen in a non deterministic way, and the string is sent to the corresponding region. If, on the contrary, no target has occurred exactly once, then the string is blocked and the computation *freezes*. Formal definitions concerning Krishna–Rama parallel P system can be found in [29]. In the same work, it is shown that extended parallel P systems of this type are computationally complete.

In the following, we will consider a slightly modified version of such systems, using generic target indications of the form $\{here, out, in\}$. In other words, instead of using targets $\{in_j\}$ (allowing to send a string to a specific inner membrane with label j), the generic target indication $\{in\}$ sends the string to an inner membrane *non-deterministically* chosen among all membranes immediately inside the membrane where the rule has been applied. It is easy to prove that the system obtained is equivalent to the original one.

In the following, we will denote the families of languages that are generated by (extended) Krishna–Rama systems (using maximal parallelism) by $EKRP^k_n(M)$, where n is the degree of the system and k is the depth of the membrane structure. If the number of membranes is not bounded a-priori, then we will replace the subscript n by $*$.

2.2 Parallel rewriting P systems with deadlock

Also in a maximally parallel P system with deadlock [6], at each step all rules are applied in a nondeterministic and maximally parallel manner on all strings, in all the regions.

The difference between these systems and KRPRP systems lies in the action taken when rules with conflicting target indications are applied. When rules with conflicting targets are applied at the same time on a string, the system enters in a *deadlock state*. The string remains inside the membrane where conflicting rules have been applied, and it will not be processed anymore by any other rule. In the following we will consider deadlock state applied to single strings; this means that, when a string is deadlocked, the remaining strings (if any) can continue their computation within the system, and eventually can produce some outputs. In case all strings are deadlocked during the computation, then the system halts producing no output. It is also possible to consider systems that halt as soon as a single string is deadlocked: we refer the reader to [6] for details.

In P systems with deadlock, any sequence of transitions producing no deadlock or producing only some local deadlock configurations forms a *computation*. The non-deadlocked strings (of terminal symbols) that exit the system are the output of the computation.

By $EParRP^k_n(M, D)$ we will denote the family of languages generated by extended rewriting P systems of degree n and membrane structure of depth k, where M denotes the use of maximal parallel application of rules, and D denotes the possibility of having deadlocks. Again, when the number of membranes is not limited, we will substitute the subscript n by $*$.

2.3 Parallel rewriting P systems without target conflicts

In a parallel P system without target conflicts [8, 10], only rules with the same target indication can be applied on a string at the same computation step, avoiding in this way the possibility of conflicting targets.

More formally, each set R_i of evolution rules of each region is divided into mutually disjoint subsets of rules, each subset having the same target indications: $R_i(here)$, $R_i(in)$, or $R_i(out)$. When we have to apply rules in parallel on a string in region i, we first non-deterministically select one target among $\{here, out, in\}$ that have at least one applicable rule on the string with the selected target (assume, for instance, it is the target in); then, we apply in a maximally parallel way all rules from the subset having the selected target indication that can be applied on the string (in our example, all rules from the set $R_i(in)$). Finally, the string is communicated according to the specific selected target.

By $EMPRP^k_n$, we denote the family of languages generated by extended RP systems without target conflicts, having degree at most n and membrane structure depth k, using the maximal parallelism method in the way just described. When the number of membranes is not limited, the subscript n is replaced by $*$.

3 Simulations of systems using different types of parallelism

It is known ([11] that parallel P systems without target conflicts are strictly included in parallel P systems with Deadlock and, moreover, that these last systems are included in KR parallel P systems. Nonetheless, no direct simulations have been proposed so far. In this section, we thus propose such direct simulations between systems using different types of parallelism.

First of all, we propose a direct simulation of parallel P systems without target conflicts by means of parallel P systems with Deadlock. In the simulation we propose, the membrane structure of the simulating systems requires two levels more than the simulated one.

Theorem 1 $EMPRP^k_* \subseteq EParRP^{k+2}_*(M, D)$

Proof Consider a parallel P system without target conflicts Π. The parallel P system with Deadlock Π' simulating Π can be built as follows. Consider a generic membrane i of Π, and add inside it three membranes labeled by Out_i, In_i, and $Here_i$. Inside each of these added membranes, add a further membrane labeled by Out'_i, In'_i, and $Here'_i$, respectively. A summary of modifications of the structure of membrane i just discussed is presented in Fig. 1.

All rules with target *Out* in membrane i are then moved in membrane Out'_i, those with target *In* are moved in membrane In'_i, and those with target *Here* are moved in membrane

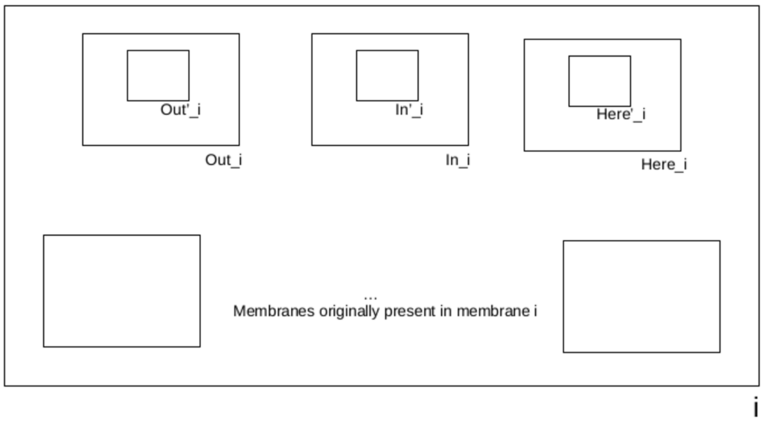

Fig. 1 Simulating a parallel P system without target conflicts by means of a parallel P system with deadlock

Table 1 Summary of modifications required

EMPRP	*EParRP*
Rules with target (*Out*)	Moved in added membrane Out'_i
Rules with target (*In*)	Moved in added membrane In'_i
Rules with target (*Here*)	Moved in added membrane $Here'_i$
	Add in membrane Out_i rules
	$T' \rightarrow O(in)$, $O \rightarrow O(out)$, and $T \rightarrow L(out)$
	Add in membrane In_i rules
	$T' \rightarrow I(in)$, $I \rightarrow I(out)$, and $T \rightarrow L(out)$
	Add in membrane $Here_i$ rules
	$T' \rightarrow H(in)$, $H \rightarrow H(out)$, and $T \rightarrow L(out)$
	Add in membrane i rules
	$T \rightarrow T'(in)$, $T' \rightarrow L(here)$, $O \rightarrow T(out)$
	$I \rightarrow T(in)$, and $H \rightarrow T(here)$
Each initial string w	Replaced by Tw

$Here'_i$, replacing in all of them the original target by the target Out. We also add rules $T' \rightarrow O(in)$, $O \rightarrow O(out)$, and $T \rightarrow L(out)$ in membrane Out_i, rules $T' \rightarrow I(in)$, $I \rightarrow I(out)$, and $T \rightarrow L(out)$ in membrane In_i, and rules $T' \rightarrow H(in)$, $H \rightarrow H(out)$, and $T \rightarrow L(out)$ in membrane $Here_i$. Finally we add to membrane i the rules $T \rightarrow T'(in)$, $T' \rightarrow L(here)$, $O \rightarrow T(out)$, $I \rightarrow T(in)$, and $H \rightarrow T(here)$.

A summary of required changes is presented in Table 1.

Each initial string w is then replaced by a string Tw (i.e. we add a symbol T to it). All symbols T, T', O, I, H, L are special symbols not initially present. The symbol L (lock symbol) is a symbol that, once introduced in the string, is never removed (thus the string produces no output).

Consider now a string Tw in membrane i. The idea is to simulate a parallel application of rules without the possibility to have conflicts concerning their target indications and, as a consequence, without deadlocks. In order to do this, we send the string to one inner added membrane, containing only rules having the same target indication. The movement of the string is then managed by means of the symbol added to the string: T corresponds to choosing the target, replaced then by O (out), I (in), or H (here) to simulate the correct movement of the string to a target membrane.

Initially, the only applicable rule in membrane i is the rule $T \rightarrow T'(in)$, which replaces T by T' and non-deterministically sends the string to an inner membrane. If such a membrane is a membrane $j \neq i$ originally already present inside membrane i, then the rule $T' \rightarrow L$ is applied, that introduces the loop symbol in the string (in fact, the string cannot reach such membranes without applying any original rules). If, on the contrary, the string reaches one of the added membrane Out_i, In_i, or $Here_i$, then the simulation of all possible applicable rules having the same target indication is started. Assume, for instance, that the chosen membrane is In_i. The symbol T' is then replaced by I, and the string is sent to membrane In'_i. Here, all applicable rules to w are applied in parallel (and all of them have now target Out), together with the rule $I \rightarrow I(out)$. The string is sent back to membrane In_i, where the only applicable rule is $I \rightarrow I(out)$ that sends the string back to membrane i. Having started from Tw we have now Iw', such that w' has been obtained by w having applied in a maximal parallel way all rules with a target indication In.

In membrane i we can apply now only the rule $I \rightarrow T(in)$, producing a string Tw' that is then sent to an inner membrane (in the discussed case; otherwise, the string is sent to the membrane immediately outside - in case of target Out - or it stays in the same membrane - in case of target $Here$). Notice that if the string reaches again an added membrane Out_i, In_i, or $Here_i$ when the symbol T is present (instead of T'), then T is replaced by the loop symbol L, and then sent back to membrane i.

We conclude the proof by highlighting the fact that small differences are needed for elementary membranes and for the skin membrane. For elementary membranes, no rules with target *In* are present, hence we do not need to add the submembranes to manage this target, nor the rule $I \rightarrow T(in)$. For the skin membrane, we simply need one small change: the rule $O \rightarrow T(out)$ is replaced by $O \rightarrow \lambda(out)$, which removes the symbol used to manage the movement of the string, and sends the string in the environment. If the string contains only terminal symbol, then it is a valid output.

It is easy to see that the computation produces the same results as the original parallel P system without target conflicts Π.

In order to explain the result, we consider now a simple example illustrating the simulation.

Example 1 Let us consider a maximally parallel P system without target conflicts $\Pi = (V, T, \mu, M_0, \dots, M_n, (R_0, \rho_0), \dots, (R_n, \rho_n))$ and a string *ABCDEF* inside a generic membrane i of Π. Moreover, let us assume that the rules in membrane i are the following:

- $A \rightarrow u(in)$
- $B \rightarrow v(in)$
- $C \rightarrow w(out)$
- $D \rightarrow x(out)$
- $E \rightarrow y(here)$
- $F \rightarrow z(here)$

In a P system without target conflicts, rules are applied in a maximally parallel way, but avoiding conflicting target indications. Thus, one possible way to apply the rules in our example is by non-deterministically choosing to apply the first two rules, thus obtaining the string *uvCDEF*, that is then sent to a membrane immediately inside membrane i; another possibility is to apply the third rule and the fourth rules, thus obtaining the string *ABwxEF*, that is then sent to the membrane immediately outside membrane i; finally, a further possibility is to apply the fifth rule and the sixth rule, obtaining the string *ABCDyz*, that remains in membrane i.

As explained above, the parallel P system with Deadlock Π' simulating Π has, inside membrane i, three membranes labeled by Out_i, In_i, and $Here_i$. Inside each of these added membranes, there is a further membrane labeled by Out'_i, In'_i, and $Here'_i$, respectively. The rules with target *in* ($A \rightarrow u(in)$, $B \rightarrow v(in)$) are moved in membrane In'_i, those with target *out* ($C \rightarrow w(out)$, $D \rightarrow x(out)$) are moved in membrane Out'_i, and those with target *here* are moved in membrane $Here'_i$, replacing in all of them the original target by the target *Out*.

We also add rules $T' \rightarrow O(in)$, $O \rightarrow O(out)$, and $T \rightarrow L(out)$ in membrane Out_i, rules $T' \rightarrow I(in)$, $I \rightarrow I(out)$, and $T \rightarrow L(out)$ in membrane In_i, and rules $T' \rightarrow H(in)$, $H \rightarrow H(out)$, and $T \rightarrow L(out)$ in membrane $Here_i$. Finally we add to membrane i the rules $T \rightarrow T'(in)$, $T' \rightarrow L(here)$, $O \rightarrow T(out)$, $I \rightarrow T(in)$, and $H \rightarrow T(here)$. The original string *ABCDEF* is replaced by the string *TABCDEF*.

The only applicable rule in membrane i is now the rule $T \rightarrow T'(in)$, which produces the string $T'ABCDEF$, and then send it in an inner membrane, non-deterministically chosen among all membranes inside membrane i.

Let's assume the string reaches membrane Out_i. The symbol T' is thus replaced by O, and the string *OABCDEF* is sent to membrane Out'_i. Here, all applicable rules (that is, $C \rightarrow w(out)$, and $D \rightarrow x(out)$) are applied in parallel, together with the rule $O \rightarrow O(out)$. The obtained string *OABwxEF* is sent back to membrane Out_i, where the only applicable rule is $O \rightarrow O(out)$ that sends the string back to membrane i. Here we finally apply the rule $O \rightarrow T(out)$, thus obtaining *TABwxEF* which is sent to the membrane immediately outside membrane i. We started from *TABCDEF* and we correctly simulated the parallel application of the two rules having target (out), and then sent the obtained string to the membrane outside membrane i.

We now propose a direct simulation of maximal parallel P systems with Deadlock by means of Krishna-Rama maximal parallel P systems. In this case, the membrane structure of the simulating systems requires just one level more than the simulated one.

Theorem 2 $EParRP^k_n(M, D) \subseteq EKRP^{k+1}_{2n}(M)$

Proof We recall that the difference between Krishna-Rama maximal parallel P systems and maximal parallel P systems with Deadlock is related to the management of conflicting target indications in the application of the rules in parallel. In fact, while in maximal parallel P systems with Deadlock a string is locked every time rules with different target indications are applied, in KR parallel P systems if a target indication is specified in exactly one rule among those applied, then it is used to denote the used target. We thus proceed with the simulation of parallel P systems with deadlock by using KR P systems in which we check the use of conflicting rules, even for the case with a single rule using a specific target indication.

Thus, consider a parallel P system with deadlock Π. The KR parallel P systems simulating Π can be built as follows. Consider a generic membrane i of Π, and add inside it a membrane labeled by $Check_i$.

Every rule $A \rightarrow x(in)$, $B \rightarrow y(out)$, and $C \rightarrow z(here)$ in membrane i is moved to membrane $Check_i$, and replaced by $A \rightarrow xII(out)$, $B \rightarrow yOO(out)$ and $C \rightarrow zHH(out)$, respectively. In membrane $Check_i$ we also add the rules $T \rightarrow L(here)$ and $T' \rightarrow \lambda(out)$.

Table 2 Summary of modifications required

Rules $A \to x(in)$ in membr. i	Replaced by $A \to xII(out)$ in $Check_i$
Rules $B \to y(out)$ in membr. i	Replaced by $B \to yOO(out)$ in $Check_i$
Rules $C \to z(here)$ in membr. i	Replaced by $C \to zHH(out)$ in $Check_i$
	Add $T \to L(here), T' \to \lambda(out)$ to $Check_i$
	Add rules $T \to T'(in), I \to T(in), H \to T(here)$
	$O \to T(out)$, and $T' \to L(here)$ to membrane i
Each initial string w in i	Replaced by TTw

In membrane i we add the rules $T \to T'(in)$, $I \to T(in)$, $H \to T(here)$, $O \to T(out)$, and $T' \to L(here)$. Each initial string w is replaced by a string TTw (i.e. we add two symbols T to it).

All symbols I, O, H, T, T', L are special symbols not initially present.

A summary of required changes is presented in Table 2.

Consider a string TTw in membrane i. We apply the rule $T \to T'(in)$: if the string reaches a membrane j originally present in membrane i, then the rule $T' \to L(here)$ is applied, which introduces the lock symbol in the string. If, on the contrary, the string reaches membrane $Check_i$, then we apply modified rules originally present in membrane i. Every such rule introduces now two symbols, corresponding to the target indication. These two symbols are used to avoid the possibility that a single rule with a specific target indication can be applied on the string. In fact, consider an application of two conflicting rules, such that one of them is applied exactly once (e.g., assume that a rule with target (in) is applied exactly once and other rules with target (out) are applied at the same time). While in a parallel system with deadlock this would lead to a conflict, in the case of Krishna-Rama systems this would not lead to a conflict: the target of the rule applied exactly once will be chosen as the target of the string.

Adding two copies of the same target allows to avoid this problem. In fact, the string is now sent back to membrane i, and it contains two copies of the symbols I, O, and H for each applied rule with target In, Out, and $Here$, respectively. At the same time, all symbols T' in the string are deleted.

If the string contains conflicting symbols (that is, at least two different symbols among I, H, and O are present), then the computation is blocked in membrane i. In fact, at least two rules among $I \to T(in)$, $H \to T(here)$, and $O \to T(out)$, can be applied and, moreover, each rule is applied at least two times, since every target symbol is introduced in couples, as explained above.

If, on the contrary, only rules with a unique target indication were applied, then only copies of one symbol among I, O, and H are present when the string reaches membrane i. If the symbols in the string are I, then we apply the rule $I \to T(in)$; the string is sent to an inner membrane. If it is again membrane $Check_i$, then T is replaced by L and the string is locked. If, instead the string reaches one of the membrane originally inside i, then the computation can continue, exploiting the symbols T just introduced.

If the string contains only symbols O, then they are replaced by T and the string is sent outside. Finally, if the string contains only symbols H, then they are replaced by T and the string remains in i, where it is ready to simulate another computation step.

As for the previous proof, we point out that for the skin membrane we simply need one small change: the rule $O \to T(out)$ is replaced by $O \to \lambda(out)$, which removes the symbol used to manage the movement of the string, and sends the string in the environment. If the string contains only terminal symbols, then it is a valid output.

It is easy to see that the computation produces the same results as the original parallel P system without target conflicts Π.

As for the previous case, we consider now a simple example illustrating the simulation.

Example 2 Let us consider a maximally parallel P system with deadlock $\Pi = (V, T, \mu, M_0, \dots, M_n, (R_0, \rho_0), \dots, (R_n, \rho_n))$ and a string $ABCEF$ inside a generic membrane i of Π.

Moreover, let us assume that the rules in membrane i are the following:

- $A \to u(in)$
- $B \to v(in)$
- $C \to w(out)$
- $E \to y(here)$
- $F \to z(here)$

We recall that in a KR parallel P system rules are applied in a maximally parallel way and, if a target indication is specified in exactly one rule among those applied, then it is used to select the used target.

In the example above, while in P systems with deadlock we would have a conflict, that would result in a locked string, in KR systems the (unique) rule with the target *out* would select the target where the string is sent, allowing the computation to proceed.

In order to correctly simulate the parallel system with deadlock, we thus need to produce a KR P systems in which we check the use of conflicting rules, even for the case with a single rule using a specific target indication.

As explained above, the KR parallel P system simulating Π is built adding inside each membrane i a membrane labeled by $Check_i$. Then, all rules in membrane i are moved to membrane $Check_i$, with some added symbols:

- $A \to uII(out)$
- $B \to vII(out)$
- $C \to wOO(out)$
- $E \to yHH(out)$
- $F \to zHH(out)$

In membrane $Check_i$ we have the rules $T \to L(here)$ and $T' \to \lambda(out)$, while in membrane i we have the rules $T \to T'(in)$, $I \to T(in)$, $H \to T(here)$, $O \to T(out)$, and $T' \to L(here)$. The original string $ABCEF$ is replaced by the string $TABCEF$.

The only applicable rule in membrane i is now the rule $T \to T'(in)$, which produces the string $T'ABCEF$, and then send it in an inner membrane, non-deterministically chosen among all membranes inside membrane i.

If it reaches one membrane originally present in membrane i, then the rule $T' \to L(here)$ is applied, thus producing the string $LABCEF$; since this string contains the lock symbol L it will not produce any output. Let's assume, on the contrary, that the string reaches membrane $Check_i$.Here, we apply all rules in parallel, thus obtaining the string $uIIvIIwOOyHHzHH$, that is then sent back to membrane i. In this membrane, each symbol I, O, and H is replaced by T, thus obtaining the string $uTTvTTwTTyTTzTT$; since the rules replacing those symbols have conflicting target indications, the string is then locked, thus producing no output; as one can notice, this happens even if the original rule with target *out* is unique, since we introduced two symbols O in the string, to avoid this problem.

Of course, considering a combination of these two simulations is possible: starting from a Parallel Rewriting P System without Target Conflicts, we first obtain an equivalent system using Deadlock, and then we simulate this one by means of a Krishna-Rama parallel system. A direct consequence of the previous two theorems is thus the following:

Corollary 1 $EMPRP_*^k \subseteq EKRP_*^{k+3}(M)$

Regarding simulations in the opposite direction, the following points should be noted. Firstly, we recall that parallel P systems without target conflicts are strictly included by parallel P systems with Deadlock. Therefore, it is not possible to simulate all systems in the opposite direction in this case.

In the case of P systems with Deadlock and KR parallel P systems, it remains uncertain whether a simulation in the opposite direction is always feasible. This uncertainty arises from the fact that we do not have conclusive information on whether the inclusion between these two classes is strict or not, as mentioned earlier in this section. Establishing a direct simulation of KR parallel P systems using P systems with Deadlock would also solve the open problem concerning their relationship.

4 Further results: Lindenmayer systems, matrix grammars and Chomsky grammars

As we saw, it is known that *EKRP* systems are Turing equivalent. On the contrary, the exact computing power of maximal parallel P systems with Deadlock and of maximal parallel P systems without conflict target indications is not known. In this section, we present some (partial) results in this respect.

The goal of this section is twofold. On the one hand, we prove some specific results concerning comparison of RP systems (in particular, using a limited amount of membranes) with other classic computing models. On the other hand, we show how to exploit the results obtained in the previous section to extend the results obtained for one type of parallel RP systems to other types of parallel RP systems.

We start by recalling from [7] the definition of Lindenmayer systems. An *ET0L system* is a tuple $G = (V, T, w, P_1, \dots, P_m)$, $m \geq 1$, where V is a finite alphabet of symbols, $T \subseteq V$, $w \in V^+$, and P_i, $1 \leq i \leq m$, are finite sets (usually called *tables*) of context-free rules defined over the alphabet V; for each $A \in V$ there is at least one rule $A \to x$ in each set P_i (tables are *complete*).

In each computation step, all the symbols in the current sentential form are rewritten by means of *one* table. The language generated by G is denoted by

$$L(G) = \{x \in T^* \mid w \Longrightarrow_{P_{j_1}} w_1 \Longrightarrow_{P_{j_2}} \cdots \Longrightarrow_{P_{j_m}} w_m = x, m \geq 0, 1 \leq j_i \leq n, 1 \leq i \leq m\} .$$

The following results concerning such systems are known: $CF \subset ET0L \subset CS$; CF and CS denote, respectively, the families of context free languages and the families of context sensitive languages. Details on L systems can be found in [52].

It is known [11] that $ET0L$ systems are properly included in the class of languages generated by maximal parallel P systems with Deadlock using at most four membranes. By considering the relations among classes of languages generated by Lindenmayer systems, we can state the following:

Theorem 3 $ED0L \subset E0L \subset ET0L \subset EParRP^3_4(M,D)$

$EDT0L \subset ET0L \subset EParRP^3_4(M,D)$

By exploiting this result and the result obtained in Theorem 2 we have:

Theorem 4 $ED0L \subset E0L \subset ET0L \subset EParRP^3_4(M,D) \subseteq EKRP^4_8(M)$ $EDT0L \subset ET0L \subset EParRP^3_4(M,D) \subseteq EKRP^4_8(M)$

Proof The result in Theorem 3 is obtained by using a system having four membranes organized so that the skin contains a single membrane, containing a single membrane, and so on. The corresponding membrane structure is of depth 3. Such systems can be simulated by Krishna-Rama Parallel P systems having twice the number of membranes of the original systems, and requiring a membrane structure having one added level of depth.

Considering context-free grammars in the Chomsky hierarchy, it is easy to see that they can be generated by maximal parallel P systems with Deadlock using a single membrane:

Theorem 5 $CF \subseteq EParRP^0_1(M,D)$.

Proof Consider a CF grammar G, and the axiom S. We can generate the same language generated by G by means of a maximal parallel P system with Deadlock Π (with a single membrane) as follows. Add the axiom S' to Π. For every rule $A \to x$ in G add the rule $A \to x(here)$ in Π. Moreover, add the rules $S' \to S'(here)$ and $S' \to \lambda(out)$.

Context free rules are applied (in parallel, but this is not an issue since no interactions among different non terminal symbols are possible) on the string in Π. At the same time, the rule $S' \to S'(here)$ is also applied. After some time, the rule $S' \to \lambda(out)$ is applied. If the string still contains nonterminal symbols, then also other rules with target *Here* are also applied, thus producing a deadlock. On the contrary, S' is removed, and the terminal string is sent out, thus producing a valid output.

It is also possible to show that also some non-context free languages can be generated by maximal parallel P systems with Deadlock using a single membrane:

Theorem 6 $L = \{a^n b^n c^n | n \geq 0\} \in EParRP^0_1(M,D)$

Proof Consider the system $\Pi = (V, T, [_0]_0, M_0 = \{S\}, R_0)$. We add to R_0 the following rules: $S \to ABC, A \to A', B \to B', C \to C'$, $A' \to Aa, B' \to Bb, C' \to Cc$ all having target *Here*, plus the rules $A \to \lambda(out), B \to \lambda(out), C \to \lambda(out)$.

Starting from the axiom S, we apply the rule $S \to ABC$. Now, consider a generic string of the form $Aa^kBb^kCc^k$ for some $k \geq 0$. If we apply in parallel rules $A \to A', B \to B', C \to C'$, then we obtain $A'a^kB'b^kC'c^k$, and we can only apply now the rules $A' \to Aa, B' \to Bb, C' \to Cc$ thus obtaining $Aa^{k+1}Bb^{k+1}Cc^{k+1}$.

At some point (or even immediately, for the string ABC) we can apply the rules $A \to \lambda(out), B \to \lambda(out), C \to \lambda(out)$, producing the string $a^h b^h c^h$ (for some $h \geq 0$) which is then sent out, thus producing a valid string.

In case of parallel application of rules having on the left side the symbols A, B, C but with different target indications on the right (*Here* or *Out*), the string produces a deadlock, and no output is obtained.

As a consequence, context free languages are strictly included in the class of languages generated by maximally parallel rewriting P systems with deadlock using a single membrane. Considering the results obtained in the previous section, such systems can be simulated by Krishna-Rama parallel systems by simply adding a membrane inside the unique membrane used to obtain the results just described. Thus, we can immediately state the following:

Theorem 7 $CF \subset EParRP^0_1(M,D)\, CF \subset EKRP^1_2(M)$

Other classes of formal languages that can be considered are those generated by Matrix grammars. A matrix grammar is a grammar $G = (N, T, S, M, C)$, where N, T denote finite disjoint alphabets of symbols, $S \in N$, M denotes a finite set of sequences of context-free rules over $N \cup T$ (nonterminal and terminal alphabets) of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, where $A_i \in N, x_i \in (N \cup T)^*$, while C is a set of rules in M.

The symbol S is called the axiom, and the elements of the set M are the matrices. For $w, z \in (N \cup T)^*$, we have $w \Rightarrow z$ if and only if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w''_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}, A_i$ does not appear in w_i, and the rule $A_i \to x_i$ appears in C (all the rules that are in a matrix must be applied following the order; it is only possible to skip the rules in the set C, in case they cannot be applied to the string; these rules are said to be applied in the appearance checking mode.) If the set C is empty, then the matrix grammar is without appearance checking.

We denote by $L(G) = \{w \in T^* | S \Rightarrow^* w\}$ the language generated by G, where the symbol $\Rightarrow^*$ denotes the reflexive and transitive closure of the relation $\Rightarrow$. By MAT_{ac} we denote the family of languages generated in this way, and by MAT the family generated by matrix systems without appearance checking. We refer the reader to [19] for further details on matrix grammars.

Relations between such classes and the class of languages generated by maximal parallel P systems with Deadlock are not known, but the following partial result can be shown, proving that maximal parallel P systems with Deadlock can generate languages not in the class MAT.

Theorem 8 $L = \{a^{2^n} | n \geq 0\} \in EParRP^0_1(M, D)$ $L = \{a^{2^n} | n \geq 0\} \in EKRP^1_2(M)$

Proof It is known [24] that all the one-letter languages in MAT are regular.

Consider the system $\Pi = (V, T, [_0]_0, M_0 = \{A\}, R_0)$. We add to R_0 the following rules: $A \rightarrow AA(here), A \rightarrow a(out)$.

Starting from A, if we apply in parallel on all symbols the rule $A \rightarrow AA(here)$, then we duplicate them. After k steps, we have a string of the form A^{2^k}. If we then apply in parallel on all symbols the rule $A \rightarrow a(out)$, we produce a correct output string. If, at some point, we apply the first rule to some symbols and the second rule to the remaining symbols, then a deadlock is obtained and no output is produced.

It is easy to see that the language generated by such a system is then $L = \{a^{2^n} | n \geq 0\}$.

As for the previous Theorem, $L = \{a^{2^n} | n \geq 0\} \in EKRP^1_2(M)$ follows immediately from the result of the direct simulation obtained in the previous section.

A direct consequence of this result is that either the class MAT is strictly included in $EParRP^*_*(M, D)$ or the two classes are incomparable, but this is still an open problem.

5 Conclusions

In this work, we shortly recall definitions and results concerning maximally parallel rewriting P systems, and we illustrate relations among systems using different strategies to solve conflict problems, by means of direct simulations. In particular, we show how to simulate systems without target conflicts by means of systems with deadlock, and how to simulate systems with deadlock by means of Krishna-Rama parallel rewriting systems. Partial results concerning relations with standard Chomsky grammars, and with Lindenmayer systems and Matrix grammars have also been presented.

We conclude the work with some open problems. First, while it is known that Krishna-Rama systems are Turing universal, complete characterizations for systems without target conflict or with Deadlock have not yet been defined.

As already pointed out in some previous works, another interesting research topic is to consider other types of parallelism, and to consider relations among parallel systems described in this work but using, instead of maximal parallelism, different parallel semantics. Some possibilities have been discussed in [6]: we recall them shortly.

- In *unique* parallel semantic, the occurrences of exactly one symbol are substituted using exactly one rule; the rule to be used is nondeterministically chosen among the set of all applicable rules.
- In *symbol* parallel semantic, we substitute all occurrences of one symbol according to exactly one rule, but different symbols can be modified in parallel in the same computing step.
- In *table* parallel semantic, we divide the set of rewriting rules into tables (like in Lindenmayer systems); at each step, the rules from a single table (non-deterministically chosen) is applied to the string.
- In *limited* parallel semantic, we define a constant k and, at each step, we apply *exactly* k rules in parallel. Subvariants that can be considered are the application of *at most* k rules or *at least* k rules.

Author contributions All the work has been done by the unique author of this paper.

Funding Open access funding provided by Università degli Studi di Milano - Bicocca within the CRUI-CARE Agreement. Work partially supported by Università degli Studi di Milano-Bicocca, Fondo diAteneo Quota Dipartimentale (2019-ATE-0454).

Data availability Not applicable.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Alhazov, A., Leporati, A., Manzoni, L., Mauri, G., & Zandron, C. (2022). Alternative space definitions for P systems with active membranes. *Journal of Membrane Computing, 4*(3), 251–260.
2. Alhazov, A., Leporati, A., Manzoni, L., Mauri, G., & Zandron, C. (2021). Evaluating space measures in P systems. *Journal of Membrane Computing, 3*(2), 87–96.
3. Alhazov, A., Freund, R., & Ivanov, S. (2021). When catalytic P systems with one catalyst can be computationally complete. *Journal of Membrane Computing, 3*(3), 170–181.
4. Alhazov, A., Leporati, A., Mauri, G., Porreca, A. E., & Zandron, C. (2014). Space complexity equivalence of P systems with active membranes and Turing machines. *Theoretical Computer Science, 529*, 69–81.
5. Baquero, F., Campos, M., Llorens, C., & Sempere, J. M. (2021). P systems in the time of COVID-19. *Journal of Membrane Computing, 3*, 246–257.
6. Besozzi, D., Ferretti, C., Mauri, G., & Zandron, C. (2003). Parallel Rewriting P Systems with Deadlock, Proc. of 8th International Workshop on DNA Based Computers (M. Hagiya, A. Ohuchi, eds.), Springer-Verlag, LNCS 2568, 302–314.
7. Besozzi, D., Ferretti, C., Mauri, G., & Zandron, C. (2003). P Systems with Deadlock. *Bio Systems, 70*(2), 95–105.
8. Besozzi, D., Mauri, G., & Zandron, C. (2003). Parallel Rewriting P Systems without Target Conflicts. Proc. of Membrane Computing International Workshop WMC-CdeA2002 (Păun, G., Rozenberg, G., Salomaa, A., Zandron, C., (eds) Springer-Verlag, LNCS 2597, 119–133.
9. Besozzi, D., Mauri, G., & Zandron, C. (2003). Deadlock Decidability in Partial Parallel P Systems, Proc. of DNA9, LNCS 2943, Springer-Verlag, pp 55–60.
10. Besozzi, D., Mauri, G., Vaszil, G., & Zandron, C. (2004) Collapsing Hierarchies of Parallel Rewriting P Systems without Target Conflicts. In Membrane Computing, International Workshop, WMC 2003, Tarragona, July 2003, Selected Papers (C. Martin-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 2933, Springer, Berlin, 55–69.
11. Besozzi, D., Mauri, G., & Zandron, C. (2004). Hierarchies of parallel rewriting P systems - A survey. *New Generation Computing, 22*, 331–347.
12. Bhuvaneswari, K., Kalyani, T., & Lalitha, D. (2019). An Intelligent Solution for a Sustainable Environment: Iso-Array Rewriting P Systems and Triangular Array Token Petri Net. *Ekoloji, 107*, 767–777.
13. Bottoni, P., Labella, A., & Martín-Vide, C. (2002). *Păun* (pp. 325–353). Gh.: Rewriting P systems with conditional communication, In Formal and Natural Computing.
14. Buiu, C., & Florea, A. (2019). Membrane computing models and robot controller design, current results and challenges. *Journal of Membrane Computing, 1*, 262–269.
15. Buño, K., & Adorna, H. (2023). Solving 3-SAT in distributed P systems with string objects. *Theoretical Computer Science, 964*, 113976.
16. Ceterchi, R., Mutyam, M., Păun, Gh., & Subramanian, K. G. (2003). Array-rewriting P systems. *Natural Computing, 2*, 229–249.
17. Ciencialová, L., Csuhaj-Varjú, E., Cienciala, L., & Sosik, P. (2019). P colonies. *Journal of Membrane Computing, 1*, 178–197.
18. Cooper, J., & Nicolescu, R. (2022). Neighbourhood message passing computation on a lattice with cP systems. *Journal of Membrane Computing, 4*, 120–152.
19. Dassow, J., & Păun, Gh. (1989). *Regulated Rewriting in Formal Language Theory*. Springer-Verlag.
20. Dassow, J., & Păun, Gh. (1999). On the power of membrane computing. *Journal of Universal Computer Science, 5*(2), 33–49.
21. Dupaya, A. G. S., Galano, A. C. A. P., Cabarle, F. G. C., Tristan De La Cruz, R., Ballesteros, K. J., & Lazo, P. P. L. (2022). A web-based visual simulator for spiking neural P systems. *Journal of Membrane Computing, 4*, 21–40.
22. Ferretti, C., Mauri, G., Păun, Gh., & Zandron, C. (2003). On three variants of rewriting P systems. *Theoretical Computer Science, 301*(1–3), 201–215.
23. Garcia-Quismondo, M., Hintz, W. D., Schuler, M. S., & Relyea, R. A. (2021). Modeling diel vertical migration with membrane computing. *Journal of Membrane Computing, 3*, 35–50.
24. Hauschildt, D., & Jantzen, M. (1994). Petri nets algorithms in the theory of matrix grammars. *Acta Informatica, 31*, 719–728.
25. Henderson, A., Nicolescu, R., Dinneen, M. J., Chan, T. N., Happe, H., & Hinze, T. (2021). Turing completeness of water computing. *Journal of Membrane Computing, 3*, 182–193.
26. Hinze, T. (2020). Coping with dynamical reaction system topologies using deterministic P modules: a case study of photosynthesis. *Journal of Membrane Computing, 2*, 281–289.
27. Huang, Y., Wang, T., Wang, J., & Peng, H. (2021). Reliability evaluation of distribution network based on fuzzy spiking neural P system with self-synapse. *Journal of Membrane Computing, 3*, 51–62.
28. Krishna, S. N., & Rama, R. (2001). A Note on Parallel Rewriting in P Systems. *Bulletin of the EATCS, 73*, 147–151.
29. Krishna, S. N., & Rama, R. (2000). On the Power of P Systems Based on Sequential/Parallel Rewriting. *International Journal of Computer Mathematics, 77*(1–2), 1–14.
30. Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., & Zandron, C. (2020). A Turing machine simulation by P systems without charges. *Journal of Membrane Computing, 2*(2), 71–79.
31. Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., & Zandron, C. (2019). Characterizing PSPACE with shallow non-confluent P systems. *Journal of Membrane Computing, 1*, 75–84.
32. Leporati, A., Manzoni, L., Mauri, G., Porreca, A. E., & Zandron, C. (2017). Characterising the complexity of tissue P systems with fission rules. *Journal of Computer and System Sciences, 90*, 115–128. P systems with input in binary form.
33. Leporati, A., Zandron, C., & Gutiérrez-Naranjo, M. A. (2006). P systems with input in binary form. *International Journal of Foundations of Computer Science, 17*(1), 127–146.
34. Liu, X., Suo, J., Leung, S. C., Liu, J., & Zeng, X. (2015). The power of time-free tissue P systems: Attacking NP-complete problems. *Neurocomputing, 159*, 151–156.
35. Liu, Y., & Zhao, Y. (2022). Weighted spiking neural P systems with polarizations and anti-spikes. *Journal of Membrane Computing, 4*, 269–283.
36. Madhu, M. (2003). Probabilistic rewriting P systems. *International Journal of Foundations of Computer Science, 14*(1), 157–166.
37. Madhu, M. (2004) Descriptional complexity of rewriting P systems. Journal of Automata, Languages and Combinatorics, 9(2-3), 311–316.
38. Mayne, R., Phillips, N., & Adamatzky, A. (2019). Towards experimental P-systems using multivesicular liposomes. *Journal of Membrane Computing, 1*, 20–28.
39. Nagar, A. K., Ramanujan, A., & Subramanian, K. G. (2018). Control words of string rewriting P systems. *International*

Journal of Advances in Engineering Sciences and Applied Mathematics, 10, 230–235.
40. Nash, A., & Kalvala, S. (2019). A P system model of swarming and aggregation in a Myxobacterial colony. *Journal of Membrane Computing, 1*, 103–111.
41. Orellana-Martín, D., Valencia-Cabrera, L., Riscos-Núñez, A., & Pérez-Jiménez, M. (2019). Minimal cooperation as a way to achieve the efficiency in cell-like membrane systems. *Journal of Membrane Computing, 1*, 85–92.
42. Pan, L., Song, B., Subramanian, K. G., Rewriting P systems with flat-splicing rules. In: Membrane Computing: 17th International Conference, CMC. (2016). Milan, Italy, July 25–29, 2016, 17. *Springer International Publishing, 2017*, 340–351.
43. Păun, Gh. (2000). Computing with Membranes. *Journal of Computer and System Sciences, 61*(1), 108–143.
44. Păun, Gh. (2001). P systems with active membranes: attacking NP-complete problems. *Journal of Automata, Languages and Combinatorics, 6*(1), 75–90.
45. Păun, Gh. (2002). *Membrane Computing*. An Introduction. Springer-Verlag.
46. Plesa, M. I., Gheorghe, M., Ipate, F., & Zhang, G. (2022). A key agreement protocol based on spiking neural P systems with anti-spikes. *Journal of Membrane Computing, 4*, 341–351.
47. Porreca, A. E., Leporati, A., Mauri, G., & Zandron, C. (2011). P systems with active membranes: trading time for space. *Natural Computing, 10*(1), 167–182.
48. Porreca, A. E., Leporati, A., Mauri, G., & Zandron, C. (2011). P systems with active membranes working in polynomial space. *International Journal of Foundations of Computer Science, 22*(1), 65–73.
49. Porreca, A. E., Leporati, A., Mauri, G., & Zandron, C. (2010). P systems with elementary active membranes: Beyond NP and coNP. *Lecture Notes in Computer Science, 6501*, 338–347.
50. Porreca, A.E., Leporati, A., Mauri, G., & Zandron, C. (2013). Sublinear-space P systems with active membranes. In: International Conference on Membrane Computing (CMC13), LNCS 7762, Springer, 342–357.
51. Porreca, A. E., Mauri, G., & Zandron, C. (2010). Non-confluence in divisionless P systems with active membranes. *Theoretical Computer Science, 411*(6), 878–887.
52. Rozenberg, G., & Salomaa, A. (1980). *The Mathematical Theory of L Systems*. Academic Press.
53. Sosík, P. (2019). P systems attacking hard problems beyond NP: a survey. *Journal of Membrane Computing, 1*, 198–208.
54. Sosík, P., & Freund, R. (2003). P systems without priorities are computationally universal. *Lecture Notes in Computer Science, 2597*, 400–409.
55. Sosík, P., & Rodríguez-Patón, A. (2007). Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences, 73*(1), 137–152.
56. Subramanian, K. G., Isawasan, P., Venkat, I., & Pan, L. (2014). Parallel array-rewriting P systems. *The Romanian Journal of Information Science and Technology, 17*(1), 103–116.
57. Yu, W., Xiao, X., Wu, J., Chen, F., Zheng, L., & Zhang, H. (2023). Application of fuzzy spiking neural dP systems in energy coordinated control of multi-microgrid. *Journal of Membrane Computing, 5*, 69–80.
58. Zandron, C. (2020). Bounding the space in P systems with active membranes. *Journal of Membrane Computing, 2*, 137–145.
59. Zandron, C., Ferretti, C., & Mauri, G. (2001). Two Normal Forms for Rewriting P Systems, in M. Margenstern, Y. Rogzhin (Eds.), Machines, Computations and Universality, 3rd Internat. Conf., MCU 2001, Lecture Notes in Computer Science, Vol. 2055, Springer, 153–164.
60. Zhang, H., Liu, X., & Shao, Y. (2022). Chinese dialect tone's recognition using gated spiking neural P systems. *Journal of Membrane Computing, 4*, 284–292.
61. Zhang, G., & Pérez-Jiménez, M. J. (2017). Gheorghe, M., Real-life applications with membrane computing, vol. 25. Springer International Publishing

Claudio Zandron got the Ph.D., in computer science from the University of Milan in 2002. Since 2006 he is an associate professor at the Department of Informatics, Systems and Communication of the University of Milano-Bicocca, Italy. His research interests concern the areas of formal languages, molecular computing models, DNA computing, membrane computing, and computational complexity.