

P Systems with Deadlock^{*}

Daniela Besozzi^a, Claudio Ferretti^b, Giancarlo Mauri^b,
Claudio Zandron^{b,*}

^a*Università degli Studi dell'Insubria
Dipartimento di Scienze Chimiche, Fisiche e Matematiche
Via Valleggio 11, 22100 Como, Italy*

^b*Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi 8, 20136 Milano, Italy*

Abstract

Rewriting P systems with parallel application of evolution rules, as defined in Besozzi et al. (2002A), are considered here. Different kinds of parallelism methods are defined for string rewriting. The notion of deadlock is then introduced to describe situations where rules with mixed target indications are simultaneously applied to a common string. The generative power of parallel P systems with deadlock is analyzed, with respect to Lindenmayer systems, and some relations among different types of parallel P systems with or without deadlock, allowing to rewrite all occurrences of a single symbol, or all the symbols applying either any of the rules or only those belonging to a specific set (table) of rules are studied. Some open problems are also formulated.

Key words: Membrane Computing, Lindenmayer System, Deadlock

1 Introduction

The P systems were introduced in Păun (1998/2000) as a class of distributed parallel computing devices of a biochemical type. The basic model consists

^{*} Work partially supported by contribution of EU commission under The Fifth Framework Programme, project "MolCoNet" IST-2001-32008.

^{*} Corresponding author. Tel. +39.02.64487875, fax: +39.02.64487839.

Email addresses: daniela.besozzi@uninsubria.it (Daniela Besozzi), ferretti@disco.unimib.it (Claudio Ferretti), mauri@disco.unimib.it (Giancarlo Mauri), zandron@disco.unimib.it (Claudio Zandron).

of a membrane structure composed by several cell-membranes, hierarchically embedded in a main membrane called the skin membrane. The membranes delimit regions and can contain objects, which evolve according to given evolution rules associated with the regions. Such rules are applied in the following way: in one step all regions are processed simultaneously by using the rules in a nondeterministic and maximally parallel manner, and at each step all the objects which can evolve should evolve. All the evolved objects are then communicated to the prescribed regions, which are always specified by a target indication associated with each rule.

A computation device is obtained: we start from an initial configuration and we let the system evolve. A computation halts when no further rule can be applied. The objects expelled through the skin membrane (or collected inside a specified output membrane) are the result of the computation.

A survey and an up-to-date bibliography can be found at the web address <http://psystems.disco.unimib.it>.

In this paper we consider rewriting P systems (see, e.g., Păun (1998/2000), Zandron (2001), Ferretti et al. (2002)) and our aim is to extend the application of evolution rules from sequential rewriting to the parallel one (see also Krishna (2001), Krishna et al. (2000), (2001)). This fact is also biologically motivated, as a cellular substance could be processed by many chemical reactions (each on a different site of it) at the same time.

Using parallel rewriting means that at each step of a computation a string has to be processed, if possible, by more than one rule in parallel, according to the parallel rewriting method. So in parallel rewriting P systems we have a three-level parallelism, involving membranes, objects and rules as well.

On the other hand, if the rules we apply on the same string have mixed target indications, then we have consistency problems for the communication of the resulting string, as there are contradictory indications about the region where the string should be at the next step.

This problem has been previously faced and solved with different strategies, for example (see Krishna (2001)) by counting the number of target indications of types *here*, *in*, *out* appearing after the parallel application of rules, and then communicating the string to the region corresponding to the maximal number of indications. Another possibility is to choose as target region the one which corresponds to the indication (if existing) that appears exactly once after the parallel application of rules. If communication problems are to be avoided, then parallel rewriting P systems without target conflicts can be considered as well (see Besozzi et al. (2002B)). In such a case, only the rules which match on the target indication can be simultaneously applied to a common string.

In Besozzi et al. (2002A) we introduced a different approach for facing the problem of communication consistency: when rules with mixed target indications are applied at the same time on a common string, then a *deadlock state* occurs inside the system (see, e.g., Tannenbaum (2001) for a definition

of deadlock and the ways of dealing with it in the field of Concurrent Systems). When a situation of deadlock arises for a string, then that string is not sent to outer or inner regions but it remains inside the current membrane, though it will not be processed anymore by any other rule. Hence the deadlock state for that string causes its further processing and communication to be stopped.

In this paper we do not consider any biological counterpart of deadlock, we only propose a theoretical analysis of parallel rewriting and of its consequences. We continue the analysis of P systems which use different parallel rewriting methods and whose configurations may present deadlock states or not. In particular, we compare parallel P systems to (1) Lindenmayer systems, (2) parallel P systems to the aim of determining whether or not the possibility of having deadlock states modifies the generative power, (3) parallel P systems which use different parallel rewriting methods, in order to find differences with respect to the class of languages generated.

In Sections 2, 3 we introduce some parallel rewriting methods and the definition of parallel P systems with deadlock. In Section 4 we give an analysis of deadlock in P systems and in Section 5 we show the obtained results.

Several open problems remain to be studied, corresponding to some further types of parallel P systems (with and without deadlock, or which use other parallel rewriting methods) not yet analyzed.

2 Parallel Rewriting Methods

We denote by V^* the free monoid generated by the alphabet V . The empty string is denoted by λ , $V^+ = V^* \setminus \{\lambda\}$ is the set of non-empty strings over V , $|w|$ represents the length of a string $w \in V^*$ and $\#_a(w)$ is the number of occurrences of a symbol $a \in V$ which appear in the string w . We will refer to Dassow et al. (1989) and Rozenberg et al. (1997) for other Formal Language Theory notions.

In this section we present some kind of parallel rewriting methods for context free rules. We assume the condition that no more than one rule will be allowed to rewrite a symbol at the same time, as in interactionless Lindenmayer systems (Rozenberg et al. (1980), (1997)). The following methods of parallel rewriting will be used:

- (S) With a **symbol parallelism** rewriting step, for each symbol that can be the subject of a rewriting rule, all of its occurrences are substituted according to the same rule. That is, given some distinct symbols $a_1, \dots, a_n \in V$ and a string $w = x_1 a'_1 x_2 a'_2 x_3 a'_3 \dots x_m a'_m x_{m+1}$, with $a'_i \in \{a_1, \dots, a_n\}$, $i = 1, \dots, m$, $m \geq n$, and $x_j \in (V \setminus \{a_1, \dots, a_n\})^*$, $j = 1, \dots, m + 1$, and given only one context free rule for each symbol $r_1 : a_1 \rightarrow \alpha_1, \dots, r_n : a_n \rightarrow \alpha_n$

- (nondeterministically chosen between all rules which can be applied to each symbol), in one step we obtain the string $w' = x_1\alpha'_1x_2\alpha'_2x_3\alpha'_3\dots x_m\alpha'_mx_{m+1}$, where $\alpha'_i \in \{\alpha_1, \dots, \alpha_n\}$, $i = 1, \dots, m$, and $\alpha'_i = \alpha_k$ in w' if and only if $a'_i = a_k$ in w for some $k = 1, \dots, n$.
- (M) With a **maximal parallelism** rewriting step, all occurrences of all symbols which can be the subject of a rewriting rule are simultaneously rewritten by rules which are nondeterministically chosen in the set of all applicable rewriting rules. That is, if the string $w = x_1a_1x_2a_2x_3a_3\dots x_na_nx_{n+1}$, with $a_1, \dots, a_n \in V$ (not necessarily distinct) and $x_i \in (V \setminus \{a_1, \dots, a_n\})^* \forall i = 1, \dots, n+1$, is such that there are no rules defined over symbols in the strings x_1, \dots, x_{n+1} , and there are some rules $r_1 : a_1 \rightarrow \alpha_1, \dots, r_m : a_m \rightarrow \alpha_m$, not necessarily distinct, then we obtain in one maximal parallel rewriting step the string $w' = x_1\alpha_1x_2\alpha_2x_3\alpha_3\dots x_m\alpha_mx_{m+1}$.
- (T) As in *TOL*, *ETOL* systems, we can consider the set of rewriting rules divided into subsets of rules, that is **tables** of rules. In this case, if we have a string w and some tables $t_1 : [r_1^1 : a_1 \rightarrow \alpha_1, \dots, r_k^1 : a_k \rightarrow \alpha_{k_1}], \dots, t_l : [r_1^l : a_1 \rightarrow \alpha_1, \dots, r_k^l : a_k \rightarrow \alpha_{k_l}]$, then in one step only the rules from a table (which is nondeterministically chosen) can be applied, and these rules must be applied in one step on all occurrences of all symbols in w , not necessarily following the order the rules appear in the table. If some rules in the chosen table are defined over symbols not in w , or if the number of rules in the table exceeds the length of the string, then we skip those (not defined or exceeding) rules without forbidding the application of the entire table.

3 Parallel Rewriting P Systems with Deadlock

A membrane structure μ is a construct consisting of several membranes hierarchically embedded in a unique membrane, called the *skin membrane*. We identify a membrane structure with a string of correctly matching square parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. We can also associate a tree with the structure, in a natural way; the height of the tree is the *depth* of the structure itself.

Each membrane identifies a region, delimited by it and the membranes (if any) immediately inside it. A membrane is said to be *elementary* if it does not have any other membrane inside. If we place multisets of objects in the region from a specified finite set V , we get a super-cell. A super-cell system (or P system) is a super-cell provided with evolution rules for its objects.

We will work with string-objects, so with every region $i = 0, 1, \dots, n$ of μ we associate a multiset of finite support over V , that is a map $M_i : V^* \rightarrow \mathbb{N}$ where $M_i = \{(x_1, M_i(x_1)), \dots, (x_p, M_i(x_p))\}$, for some $x_k \in V^+, p \in \mathbb{N}$ and $0 < M_i(x_k) < \infty \forall k = 1, \dots, p$.

A *parallel rewriting P system* of degree $n + 1$ is defined by the construct

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

where:

- (1) V is the alphabet of the system;
- (2) $T \subseteq V$ is the terminal alphabet;
- (3) μ is a membrane structure with $n + 1$ membranes, which are injectively labeled by numbers in the set $\{0, 1, \dots, n\}$;
- (4) M_0, \dots, M_n are multisets over V , representing the strings initially present in the regions $0, 1, \dots, n$ of the system;
- (5) R_0, \dots, R_n are finite sets of *evolution rules* of the form $a \rightarrow \alpha(\text{tar})$, with $a \in V, \alpha \in V^*, \text{tar} \in \{\text{here}, \text{out}, \text{in}\}$, associated with the regions of μ .

The application of evolution rules is performed as follows: in one step all regions are processed simultaneously by using the rules in a nondeterministic and parallel manner. This means that in each region the objects to evolve and the rules to be applied to them are nondeterministically chosen, but all objects which can evolve should evolve. Moreover, at each step of a computation a string has to be processed according to the chosen parallelism method.

The strings resulting after the parallel application of the rules must be communicated to the prescribed region, which is always specified by the target indication associated with each rule. When we apply two or more rules to the same string, we have to check that their target indications match before communicating the resulting string to the right region. To this aim, for every region $i = 0, \dots, n$ of the membrane structure we divide the set R_i of evolution rules into mutually disjoint subsets of rules which have the same target indications, that is $R_i = \mathcal{H}_i \cup \mathcal{O}_i \cup \mathcal{I}_i$, where $\mathcal{H}_i = \{r \in R_i \mid \text{tar}(r) = \text{here}\}$, $\mathcal{O}_i = \{r \in R_i \mid \text{tar}(r) = \text{out}\}$ and $\mathcal{I}_i = \{r \in R_i \mid \text{tar}(r) = \text{in}\}$. Observe that for every elementary membrane the set \mathcal{I}_i will always be empty, and that for any other region any subset of rules could be empty as well.

Consider now some rules r_1, \dots, r_m , for some $m \geq 2$, which can be applied to a common string w at the same time. If it holds that (1) $r_1, \dots, r_m \in \mathcal{H}_i$ or (2) $r_1, \dots, r_m \in \mathcal{I}_i$ or (3) $r_1, \dots, r_m \in \mathcal{O}_i$, that is we apply in parallel only rules which match on the target membrane, then the resulting string w' (obtained after the parallel application of r_1, \dots, r_m) (1) remains inside the current region i , (2) is communicated to a (nondeterministically chosen) inner region, (3) is communicated to the outer region. In particular, if the string exits the system, it can never come back, and it may contribute to the output of the system.

Otherwise, if the set of rules $\{r_1, \dots, r_m\} = \mathcal{R}$ that we want to apply have mixed target indications, that is, for instance, $\mathcal{R} \cap \mathcal{H}_i \neq \emptyset \wedge \mathcal{R} \cap \mathcal{I}_i \neq \emptyset$, then we have consistency problems for the communication of the resulting string, as

there are contradictory indications about the region where the string should be at the next step.

This problem was faced and solved with different strategies (Krishna (2001)), though we will use here the different approach chosen in Besozzi et al. (2002A): we say that when rules with mixed target indications are applied at the same time on a common string we have a **deadlock state** inside the system. The string is not sent to outer or inner regions but it remains inside the current membrane, though it will not be processed anymore by any other rule; this choice does not mean that the indication *here* determines the target region, it means that further string processing and communications are stopped when a situation of deadlock arises for that string. In particular, we may choose to consider a deadlock state only for that string, or to define the deadlock state for the entire membrane where such string is placed. In other words, the membrane could be seen and used in two distinct ways:

(D_1) : other strings can enter the membrane and be processed by local rules, but they can never exit the region even if they are not in a deadlock state;

(D_2) : other strings can enter the membrane, be processed by local rules and even exit the region (if they are not in a deadlock state after the application of local rules).

In the first case, (D_1) , it happens that a consistency problem for target matching on a single string causes the system to lose an entire computing unit, as no strings are allowed to exit that membrane anymore. This interpretation differs, for example, from the dissolving action δ of membranes (see Păun (1998/2000)), in fact after dissolving a membrane we lose the membrane but we recover its objects in the outer region, while for deadlock membrane both the membrane and its objects are lost.

In the second case, (D_2) , the membrane would act like a filter for wrong or right strings, that is for strings with or without deadlock, stopping the wrong ones and letting the right ones proceed. A wrong string could be seen as an error taking place during the computation of the P system, and hence it must be stopped.

At a given time, the membrane structure together with all multisets of objects associated with the regions represent the *configuration* of the system at that time. The $(n + 2)$ -tuple $C_0 = (\mu, M_0, \dots, M_n)$ constitutes the initial configuration of the system. For two configurations $C_t = (\mu, M_0^t, \dots, M_n^t)$, $C_{t+1} = (\mu, M_0^{t+1}, \dots, M_n^{t+1})$ of the system we say that there is a *transition* from C_t to C_{t+1} if we can apply the rules present in the regions according to the above prescriptions.

We say that a generic configuration $C_t = (\mu, M_0^t, \dots, M_n^t)$ is *free* if there are no deadlock states inside the system at that time. Otherwise, we say that the system is in a *deadlock configuration*, and we denote by $\langle M_j^t \rangle$ all multisets which contain at least a deadlocked string.

A transition starting from a deadlock configuration will always reach another

deadlock configuration, we do not consider the chance of removing deadlock states. So if $C_t = (\mu, M_0^t, \dots, \langle M_j^t \rangle, \dots, M_n^t)$ is a deadlock configuration, then for all $t' \geq t$ it holds that $C_{t'} = (\mu, M_0^{t'}, \dots, \langle M_j^{t'} \rangle, \dots, M_n^{t'})$, where other multisets besides $\langle M_j^{t'} \rangle$ could have reached a deadlock state. We remark that the multiset $\langle M_j^{t'} \rangle$ still represents a deadlock state even though it is not necessarily equal to $\langle M_j^t \rangle$, because other strings may have entered membrane j .

A configuration where all multisets are in a deadlock state (that is, at least one string in each multiset is in a deadlock state) is said to be a *global deadlock* configuration, otherwise we talk about *local deadlock* configurations.

A sequence of transitions of free and (local) deadlock configurations forms a *computation*. We say that a computation is *halting* when there is no rule which can be further applied in the current configuration. If we interpret deadlock as in (D_1) , then a global deadlock configuration always causes the computation to halt. Observe that if the P system is processing a single string and if there are no rules which can increase the number of the strings, then in both cases (D_1) and (D_2) even a local deadlock configuration causes the computation to halt. A computation is said to be *non-halting* if there is at least one rule which can be applied forever.

In this paper we consider *extended* P systems. The *output* of an extended system is the set of strings over T (if any) sent out of the system during a computation which eventually halts. Anyway, a string which exits the system but contains symbols not in T does not contribute to the generated language. Non-halting computations provide no output.

We denote by $EParRP_n^k(\pi, \Delta)$ the family of languages generated by extended rewriting P systems of degree n and depth k , where $\pi \in \{(S), (M), (T)\}$ denotes the used parallelism method and $\Delta \in \{D, nD\}$ denotes systems with or without the possibility of having deadlock states, respectively. We use the notation $EParRP(\pi, \Delta)$ for systems whose depth or degree are not specified. If the depth is specified but the number of membranes is not limited, then the subscript n is replaced by $*$.

4 Analysis of Deadlock

Let us now consider a generic string $w \in V^+$ which, at a given time during the computation, is inside membrane i , for some $i = 0, \dots, n$. We want to analyze under which circumstances the membrane i can be considered a *safe region* (Fajstrup et al. (1998)) for the string w ; that is, given the pair (w, R_i) , we want to determine if there is no possibility for w to be in a deadlock state after the parallel application of some rules from R_i . Otherwise, we say that membrane i is an *unsafe region* for the string w .

To this aim, suppose that the set R_i contains m evolution rules which are defined over a set of distinct symbols $a_1, \dots, a_l \in V$, with $l \leq m$ and with the condition that, for all $j = 1, \dots, l$, there exists at least one rule in R_i which is defined over a_j . Moreover, suppose that $\#_{a_j}(w) \geq 0 \forall j = 1, \dots, l$ and that $|w| \geq 2$ (so we can have parallel rewriting).

Given the above assumptions, consider the following conditions:

(\star) : there exists at least one couple of symbols a_{j_1}, a_{j_2} such that $\#_{a_{j_1}}(w) > 0$, $\#_{a_{j_2}}(w) > 0$ (for some $j_1, j_2 \in \{1, \dots, l\}, j_1 \neq j_2$), and there exists at least one couple of rules $r_{j_1}, r_{j_2} \in R_i$, defined over a_{j_1}, a_{j_2} respectively, such that $\text{tar}(r_{j_1}) \neq \text{tar}(r_{j_2})$;

($\star\star$) : there exists at least one symbol a_j such that $\#_{a_j}(w) > 1$ (for some $j \in \{1, \dots, l\}$), and there exists at least one couple of rules $r_{j_1}, r_{j_2} \in R_i$, both defined over a_j , such that $\text{tar}(r_{j_1}) \neq \text{tar}(r_{j_2})$.

Depending on the parallel method we decide to use, the conditions (\star), ($\star\star$) define the only possibilities for the membrane i to be an unsafe region for w . Observe that one condition does not exclude the other, they both could hold at the same time in the same membrane.

In the following table, when a letter u (respectively s) is placed at the crossing between a row marked with (\star) or ($\star\star$) and a column marked with (S), (M) or (T), it means that membrane i is unsafe (resp. safe) for w under condition (\star) or ($\star\star$) and using the parallel method (S), (M) or (T).

For (T)-parallelism, the letter u' means that the region is unsafe if the rules in conditions (\star), ($\star\star$) belong to the same table. An interesting research topic related to this matter concerns the complexity of deciding whether or not a string is safe, in a given parallel method.

Table 1
Safe and unsafe regions

	(S)	(M)	(T)
(\star)	u	u	u'
($\star\star$)	s	u	u'

5 The Computational Power

In this section we analyze the computational power of parallel P systems with deadlock which use maximal, symbol or table parallelism.

These systems are compared to Lindenmayer systems, in particular we prove that the family of languages generated by $ETOL$ systems is strictly included in

the family of languages generated by maximal parallel P system with deadlock. Then we show that, when using symbol parallelism, the possibility of having deadlock states does not modify the generative power of the corresponding P system without deadlock, as it was shown to hold for maximal parallelism in Besozzi et al. (2002A).

Finally, we study the relations among P systems with and without deadlock which use maximal or table parallel rewriting methods.

5.1 Relations with Lindenmayer Systems

An *ETOL system* is a construct $G = (V, T, w, P_1, \dots, P_m)$, $m \geq 1$, where V is an alphabet, $T \subseteq V$, $w \in V^+$, and P_i , $1 \leq i \leq m$, are finite sets (*tables*) of context-free rules over V such that for each $A \in V$ there is at least one rule $A \rightarrow x$ in each set P_i (we say that these tables are *complete*). In a derivation step, all the symbols present in the current sentential form are rewritten using *one* table. The language generated by G is $L(G) = \{x \in T^* \mid w \xRightarrow{P_{j_1}} w_1 \xRightarrow{P_{j_2}} \dots \xRightarrow{P_{j_m}} w_m = x, m \geq 0, 1 \leq j_i \leq m, 1 \leq i \leq m\}$.

It is known that $CF \subset ETOL \subset CS$, where CF, CS denote the families of context free and context sensitive languages, respectively (see Rozenberg et al. (1997),(1980) for more details about Lindenmayer systems).

In Besozzi et al. (2002A) it was shown that $ETOL \subseteq EParRP_4^4((M), D)$. Here we prove that the inclusion is proper:

Theorem 1 $ETOL \subset EParRP_4^4((M), D)$

Proof. We show that the language $L = \{(ab^n)^m \mid m \geq n \geq 1\}$, which does not belong to $ETOL$, can be generated by a system of type $EParRP_4^4((M), D)$.

Consider the system $\Pi = (V, T, [{}_0[{}_1[{}_2[{}_3]_2]_1]_0, \emptyset, \emptyset, \emptyset, M_3, R_0, \dots, R_3)$ where $V = \{A, B, B', C, a, b\}$, $T = \{a, b\}$, $M_3 = \{AB\}$ and with the following sets of rules:

$$\begin{aligned} R_0 &= \{B' \rightarrow \lambda(out), C \rightarrow \lambda(out)\}; \\ R_1 &= \{A \rightarrow \lambda(in), A \rightarrow \lambda(out)\}; \\ R_2 &= \{A \rightarrow \lambda(out), B' \rightarrow bB'(out), C \rightarrow A(out), C \rightarrow C(out)\}; \\ R_3 &= \{A \rightarrow AC(here), A \rightarrow AC(out), B \rightarrow aB'B(here), B \rightarrow aB'(out)\}. \end{aligned}$$

The computation starts in membrane 3 where the string AB can only be rewritten by means of the rules $A \rightarrow AC(here), B \rightarrow aB'B(here)$ or $A \rightarrow AC(out), B \rightarrow aB'(out)$, otherwise a deadlock state occurs. Using the rules with target indication *here* for $k - 1$ steps, for some $k \geq 1$, we obtain the

string $AC^{k-1}(aB')^{k-1}B$; if at step k the rules with target *out* are used, then the string $AC^k(aB')^k$ is sent to membrane 2. Observe that in membrane 3 we create strings where the number of occurrences of the symbol C and of the couple of symbols aB' are equal. In particular, we point out that the k occurrences of C will be used to check that the number of symbols b introduced in membrane 2 will not be greater than such fixed value k .

Let us see how the string $AC^k(aB')^k$ ($k \geq 1$) is processed inside membrane 2. In one step we can simultaneously apply all the rules from R_2 : the symbol A is deleted, one copy of symbol b is introduced and the k occurrences of C can be nondeterministically transformed into a number $n_{A,1}$ of symbols A and a number $n_{C,1}$ of C . We obtain a string $x(abB')^k$ where x contains only A 's and C 's and $\#_A(x) = n_{A,1}$, $\#_C(x) = n_{C,1}$ (observe that $n_{A,1} + n_{C,1} = k$). The order of appearance of symbols A and C in the string is not relevant for our purpose. For simplicity, in the following we assume strings where all symbols A will appear before all symbols C .

The string $A^{n_{A,1}}C^{n_{C,1}}(abB')^k$ is sent to membrane 1 where, in order to avoid deadlock states, we have to apply the same rule $A \rightarrow \lambda(in)$ or $A \rightarrow \lambda(out)$ for all the occurrences of A . Observe that the computation halts (without any output) if $n_{A,1} = 0$, so in membrane 2 we have to use the rule $C \rightarrow A(out)$ at least for one copy of C to let the computation proceed.

Assume that $n_{A,1} \neq 0$ and we use the rule $A \rightarrow \lambda(out)$: the string $C^{n_{C,1}}(abB')^k$ is sent to the skin membrane, where all occurrences of C and B' are erased. In this way we generate the string $(ab)^k$.

If $n_{A,1} \neq 0$ and we use the rule $A \rightarrow \lambda(in)$, then the string $C^{n_{C,1}}(abB')^k$ returns to membrane 2 where the symbols A are deleted, one copy of symbol b is introduced, and the $n_{C,1}$ occurrences of C are nondeterministically transformed into a number $n_{A,2}$ of symbols A and a number $n_{C,2}$ of C , such that $n_{A,2} + n_{C,2} = n_{C,1} < k$. The string $A^{n_{A,2}}C^{n_{C,2}}(ab^2B')^k$ returns to membrane 1 and the process can be iterated.

Consider now the string $A^{n_{A,i}}C^{n_{C,i}}(ab^iB')^k$ that, after the i -th iteration, is inside membrane 1. If we choose to apply the rule $A \rightarrow \lambda(out)$, then we generate the string $(ab^i)^k$. If we choose to apply the rule $A \rightarrow \lambda(in)$, the iteration goes on. What we want to show is that the number i of iterations will always be less or equal to the value k , so the system never generates any string of the form $(ab^n)^m$ with $n > m$. From the previous analysis, we see that at the i -th iteration it holds $n_{A,i} + n_{C,i} = n_{C,i-1} \leq k$ for any value of $i \geq 1$ (with the initial condition $n_{C,0} = k$), and that the string $(ab^i)^k$ can be generated. So we have to prove that after k iterations, that is if $i > k$, the computation halts and no output is produced.

Observe that, in any case, the iteration process eventually stops because at each step we decrease by one the finite value k . Moreover, we can say that the halting condition for the iteration process is given by the values $n_{A,i} = 1$ and

$n_{C,i} = 0$, which corresponds to having the string $A(ab^i B')^k$ inside membrane 1. In fact, if now we try to repeat the iteration by using the rule $A \rightarrow \lambda(in)$, then $(ab^i B')^k$ enters membrane 2 and one more b is introduced, the string $(ab^{i+1} B')^k$ would return to membrane 1 where no rules can be applied anymore.

So, the maximum number of iterations which permits the generation of a string is the value $i = k$, which is achieved when $n_{A,i} = 1$ and $n_{C,i} = n_{C,i-1} - 1$ for all $i \geq 1$, that is when in membrane 2 only one copy of C is always changed into one copy of A , and all the other occurrences of C always remain the same. Of course this process can be iterated up to k times (the time needed to reach the halting condition decreasing k by 1 at each iteration), that is until there are no more occurrences of C in the current string. This is also the only case when we can generate the string $(ab^k)^k$. Anyway, as we have already shown, no strings of the form $(ab^i)^k$ for $i > k$ can be produced. In fact, if now we apply the rule $A \rightarrow \lambda(in)$ on the string $A(ab^k B')^k$, then $(ab^k B')^k$ would enter membrane 2 and one more b is introduced, the string $(ab^{k+1} B')^k$ returns to membrane 1 but here no rules can be applied anymore.

Hence we can generate each and every string of the language $L = \{(ab^n)^m \mid m \geq n \geq 1\}$, and it follows that $ETOL \subset EParRP_4^A((M), D)$. \square

5.2 Deadlock vs. Non Deadlock

In Besozzi et al. (2002A) we proved that parallel rewriting P systems with deadlock which use the maximal parallelism method are equivalent to the same kind of systems without deadlock. Those results about P systems with and without deadlock also hold when using the symbol parallelism method:

Theorem 2 (i) $EParRP_n^k((S), nD) \subseteq EParRP_n^k((S), D)$;
(ii) $EParRP_n^k((S), D) \subseteq EParRP_{7n}^{k+2}((S), nD)$.

The proof of the previous theorem is similar to the proof of Theorem 3 in Besozzi et al. (2002A), so we skip it here. We show, instead, a result concerning the particular case of systems of depth two:

Theorem 3 $EParRP_n^2((S), D) \subseteq EParRP_{4n+3}^2((S), nD)$.

Proof. Consider a system $\Pi = (V, T, \mu, M_0, \dots, M_{n-1}, R_0, \dots, R_{n-1})$, such that $L(\Pi) \in EParRP_n^2((S), D)$, where the skin membrane is labeled with 0 and the inner membranes with $i = 1, \dots, n - 1$.

We show how to construct a system $\Pi' = (V', T, \mu', M'_0, \dots, M'_m, R'_0, \dots, R'_m)$, such that $L(\Pi') \in EParRP_m^2((S), nD)$, which generates the same language as Π . Here the alphabet is $V \cup \bar{V} \cup \{X_i, X_{i,\text{here}}, X_{i,\text{out}} \mid i = 0, \dots, n - 1\} \cup \{X_{0,\text{in}}, X'_{0,\text{out}}, \dagger\}$, where $\bar{V} = \{\bar{A} \mid A \in V\}$ and $V, \bar{V}, \{X_i, X_{i,\text{here}}, X_{i,\text{out}} \mid i =$

$0, \dots, n-1$ }, $\{X_{0,\text{in}}, X'_{0,\text{out}}, \dagger\}$ are mutually disjoint.

In the membrane structure μ' , the skin membrane m'_0 and six new membranes $m_{0(\text{tar})}, m_{0(\text{tar}),\text{check}}$, for $\text{tar} \in \{\text{here}, \text{in}, \text{out}\}$ are used to simulate the skin membrane m_0 in μ , while any other membrane m_i in μ is simulated by means of four new membranes $m_{i(\text{tar})}, m_{i(\text{tar}),\text{check}}$, for $\text{tar} \in \{\text{here}, \text{out}\}$.

It follows that we need $7 + 4(n-1)$ membranes in Π' to correctly simulate each and every computation of Π , hence $m = 4n + 3$. Moreover, the inner membranes in Π' are all placed at the same hierarchical level, so the depth of μ' is unchanged with respect to μ .

Every string $w \in M_i$, for all $i = 0, \dots, n-1$, is transformed into the new string $X_i w$ and placed inside the skin membrane m'_0 . All other multisets in Π' are empty.

The system Π' contains the following sets of rules:

$$\begin{aligned}
R'_0 &= \{X_i \rightarrow X_i(\text{in}) \mid i = 0, \dots, n-1\} \cup \\
&\quad \{X_{i,\text{tar}} \rightarrow X_{i,\text{tar}}(\text{in}) \mid i = 0, \dots, n-1, \text{tar} \in \{\text{here}, \text{out}\}\} \cup \\
&\quad \{X_{0,\text{in}} \rightarrow X_{0,\text{in}}(\text{in}), X'_{0,\text{out}} \rightarrow \lambda(\text{out})\}; \\
R'_{0(\text{in})} &= \{A \rightarrow \bar{y}X_{0,\text{in}}(\text{out}) \mid A \rightarrow y(\text{in}) \in R_0\} \cup \\
&\quad \{X_0 \rightarrow \lambda(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 1, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{tar}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, \text{tar} \in \{\text{here}, \text{out}\}\}; \\
R'_{0(\text{in}),\text{check}} &= \{A \rightarrow \dagger(\text{out}) \mid A \rightarrow y(\text{tar}) \in R_0 \text{ such that } \text{tar} \neq \text{in}\} \cup \\
&\quad \{\bar{B} \rightarrow B(\text{out}) \mid \forall \bar{B} \in \bar{V}\} \cup \\
&\quad \{X_{0,\text{in}} \rightarrow X_j(\text{out}) \mid j = 1, \dots, n-1\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{tar}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, \text{tar} \in \{\text{here}, \text{out}\}\}; \\
R'_{0(\text{out})} &= \{A \rightarrow \bar{y}X_{0,\text{out}}(\text{out}) \mid A \rightarrow y(\text{out}) \in R_0\} \cup \\
&\quad \{X_0 \rightarrow \lambda(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 1, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{tar}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, \text{tar} \in \{\text{here}, \text{out}\}\}; \\
R'_{0(\text{out}),\text{check}} &= \{A \rightarrow \dagger(\text{out}) \mid A \rightarrow y(\text{tar}) \in R_0 \text{ such that } \text{tar} \neq \text{out}\} \cup \\
&\quad \{\bar{B} \rightarrow B(\text{out}) \mid \forall \bar{B} \in \bar{V}\} \cup \\
&\quad \{X_{0,\text{out}} \rightarrow X'_{0,\text{out}}(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{here}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{out}} \rightarrow \dagger(\text{out}) \mid j = 1, \dots, n-1\},
\end{aligned}$$

for all $i = 0, \dots, n-1$:

$$\begin{aligned}
R'_{i(\text{here})} &= \{A \rightarrow \bar{y}X_{i,\text{here}}(\text{out}) \mid A \rightarrow y(\text{here}) \in R_i\} \cup \\
&\quad \{X_i \rightarrow \lambda(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, j \neq i\} \cup \\
&\quad \{X_{j,\text{tar}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, \text{tar} \in \{\text{here}, \text{out}\}\}; \\
R'_{i(\text{here}),\text{check}} &= \{A \rightarrow \dagger(\text{out}) \mid A \rightarrow y(\text{tar}) \in R_i \text{ such that } \text{tar} \neq \text{here}\} \cup \\
&\quad \{\bar{B} \rightarrow B(\text{out}) \mid \forall \bar{B} \in \bar{V}\} \cup \\
&\quad \{X_{i,\text{here}} \rightarrow X_i(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{here}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, j \neq i\} \cup \\
&\quad \{X_{j,\text{out}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\},
\end{aligned}$$

for all $i = 1, \dots, n-1$:

$$\begin{aligned}
R'_{i(\text{out})} &= \{A \rightarrow \bar{y}X_{i,\text{out}}(\text{out}) \mid A \rightarrow y(\text{out}) \in R_i\} \cup \\
&\quad \{X_i \rightarrow \lambda(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, j \neq i\} \cup \\
&\quad \{X_{j,\text{tar}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, \text{tar} \in \{\text{here}, \text{out}\}\}; \\
R'_{i(\text{out}),\text{check}} &= \{A \rightarrow \dagger(\text{out}) \mid A \rightarrow y(\text{tar}) \in R_i \text{ such that } \text{tar} \neq \text{out}\} \cup \\
&\quad \{\bar{B} \rightarrow B(\text{out}) \mid \forall \bar{B} \in \bar{V}\} \cup \\
&\quad \{X_{i,\text{out}} \rightarrow X_0(\text{out}), X_{0,\text{in}} \rightarrow \dagger(\text{out})\} \cup \\
&\quad \{X_j \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\} \cup \\
&\quad \{X_{j,\text{out}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1, j \neq i\} \cup \\
&\quad \{X_{j,\text{here}} \rightarrow \dagger(\text{out}) \mid j = 0, \dots, n-1\}.
\end{aligned}$$

The computation starts inside membrane m'_0 for every string $X_i w$, $i = 0, \dots, n-1$. By using the rule $X_i \rightarrow X_i(\text{in})$, the string $X_i w$ is nondeterministically introduced into an inner membrane; if it reaches one of the membranes $m_{i(\text{tar})}$, with $\text{tar} \in \{\text{here}, \text{out}\}$ for $i = 1, \dots, n-1$, with $\text{tar} \in \{\text{here}, \text{out}, \text{in}\}$ for $i = 0$, then the computation can proceed, otherwise the trap symbol \dagger is introduced and no output will be produced. Any computation in Π' can proceed in four different ways, according to the simulated computation in Π :

- (1) If no rule in R_i can be applied to w and in Π' the string $X_i w$ enters a membrane $m_{i(\text{tar})}$, then no rules with target tar can be applied here, the symbol X_i is erased, the string w exits the membrane and it will remain forever inside membrane m'_0 . Otherwise, the rules $A \rightarrow \bar{y}X_{i,\text{tar}}(\text{out})$ introduce the support symbols $X_{i,\text{tar}}$, which lead to the control of target consistency in membranes $m_{i(\text{tar}),\text{check}}$ (see case (4)).
- (2) If the application of rules in R_i *surely* leads to a deadlock state, then the trap symbol is introduced in membrane $m_{i((\text{tar}),\text{check})}$ (which is used to check whether or not other rules with distinct targets can be applied to the current string) and the string will never contribute to the generated language.
- (3) If the application of rules in R_i *could* lead to a deadlock state, that is only

some nondeterministic choices of rules produce a deadlock state in Π , then such choices are not simulated in Π' , so if we have some output in Π then we have the same output in Π' , otherwise no string will be generated in Π' if it cannot be generated in Π as well.

(4) If the application of rules in R_i does not lead to a deadlock state, then the computation proceeds as follows. Let us assume that the target of applicable rules in m_i ($i = 1, \dots, n - 1$) is *here*, then inside membrane $m_{i(\text{here})}$ we introduce the symbol $X_{i,\text{here}}$ and, after checking for target consistency in $m_{i(\text{here}),\text{check}}$, the string will return to membrane m'_0 with the support symbol X_i , ready to begin another simulation of the rules in R_i . The case for the simulation of rules with target *in* is similar.

If the target of applicable rules in m_i ($i = 1, \dots, n - 1$) is *out*, the only difference is that we introduce the symbol X_0 after checking for target consistency in $m_{i(\text{out}),\text{check}}$; at the next step the produced string X_0w' can be rewritten only inside of the membranes $m_{0(\text{tar})}$, that is we simulate the rules in R_0 .

The computations inside the membranes in Π' which simulate the skin membrane m_0 are similar to all the others. In particular, in membrane $m_{0(\text{in}),\text{check}}$ we introduce again the symbol X_i in order to simulate the nondeterministical choice of target *in* in m_0 .

Finally, the output is controlled by the symbol $X'_{0,\text{out}}$, which is introduced in membrane $m_{0(\text{out}),\text{check}}$ only if some rules with target *out* are applicable in membrane $m_{0(\text{out})}$, that is only if some rules can be applied in m_0 and no deadlock occurs. In this case, the rule $X'_{0,\text{out}} \rightarrow \lambda(\text{out})$ deletes the support symbols $X'_{0,\text{out}}$ and the output is produced.

It follows that $L(\Pi') = L(\Pi)$. \square

Directly from the previous theorems it follows that:

Corollary 4 $EParRP((S), D) = EParRP((S), nD)$.

5.3 Relations among Parallel P Systems

We begin here the analysis of the relations among P systems with and without deadlock, which use different parallel rewriting methods. In Besozzi et al. (2002B) we proved that parallel rewriting P systems without target conflicts, which use maximal or table parallelism, are equivalent. Here we extend those results to parallel P systems with deadlock. The relations among P systems which use other methods are still to be analyzed.

Theorem 5 (i) $EParRP_n^k((M), \Delta) \subseteq EParRP_n^k((T), \Delta)$, for $\Delta \in \{D, nD\}$;
(ii) $EParRP_n^k((T), D) \subseteq EParRP_*^{k+1}((M), D)$;
(iii) $EParRP_n^k((T), nD) \subseteq EParRP_*^{k+3}((M), nD)$.

Proof. (i). Consider a P system Π such that $L(\Pi) \in EParRP_n^k((M), \Delta)$. In order to prove the inclusion, construct an equivalent P system Π' of type $EParRP_n^k((T), \Delta)$ which has the same alphabets and membrane structure as Π . It suffices now to put all the rules of any membrane of Π inside a single table in the corresponding membrane of Π' .

(ii). Let $\Pi = (V, T, \mu, M_0, \dots, M_{n-1}, R_0, \dots, R_{n-1})$ be a system such that $L(\Pi) \in EParRP_n^k((T), D)$. Assume that m_0 is the skin membrane in μ . We show how to construct a system $\Pi' = (V', T, \mu', M'_0, \dots, M'_m, R'_0, \dots, R'_m)$, such that $L(\Pi') \in EParRP_m^{k+1}((M), D)$ and $L(\Pi') = L(\Pi)$. We have $V' = V \cup \{X, X_t, X_{\text{here}}, X_{\text{in}}, X_{\text{out}}, \dagger\}$, where $V \cap \{X, X_t, X_{\text{here}}, X_{\text{in}}, X_{\text{out}}, \dagger\} = \emptyset$.

Consider a generic membrane m_i of Π , for any $i = 0, \dots, n-1$, which can contain a set of strings M_i , a set of tables of rules R_i and, possibly, a set $\{m_{i,1}, \dots, m_{i,h}\}$ of other membranes. We show how to simulate this generic membrane in the system Π' , and we point out that the simulation of all other membranes follows the same recursive description below.

The membrane m'_i in Π' , corresponding to m_i in Π , is obtained by replacing every string w in M_i with a string Xw , where X is a symbol not in V . The set of rules of the membrane m'_i will be:

$$R'_i = \{X \rightarrow X_t(in), X_{\text{here}} \rightarrow X_t(in), X_{\text{in}} \rightarrow X(in), X_{\text{out}} \rightarrow X(out)\}.$$

(In the skin membrane ($i = 0$), the last rule is replaced with $X_{\text{out}} \rightarrow \lambda(out)$).

Inside m'_i , we add some new membranes denoted by $m'_{i,1}, \dots, m'_{i,s}$, one for each table $t_r \in R_i$, for $r = 1, \dots, s$. Each new membrane $m'_{i,r}$ will contain the following rules:

$$R'_{i,r} = \{A \rightarrow yX_{\text{tar}}(out) \mid A \rightarrow y(tar) \in t_r\} \cup \{X_t \rightarrow \lambda(out), X \rightarrow \dagger(out)\}.$$

Finally, we add the rule $X_t \rightarrow \dagger(out)$ in each membrane $m'_{i,1}, \dots, m'_{i,h}$, which are all placed inside m'_i and correspond to the membranes $m_{i,1}, \dots, m_{i,h}$ originally placed in membrane m_i . (Observe that, if $i \neq 0$, then the rule $X_t \rightarrow \dagger(out)$ will be placed also inside m'_i because of the recursive construction of the system).

From the construction of μ' , it follows that the number of membranes we need in Π' to simulate each membrane in Π depends on the number of tables in each R_i , hence m cannot be a priori bounded. Instead, we can say that the depth is increased from the value k to the new value $k+1$.

Let us now see how the system works. Consider a string Xw in membrane m'_i , at the first step of a computation we always have to apply the rule $X \rightarrow X_t(in)$,

which introduces the new symbol X_t and sends the string inside an inner membrane. If $X_t w$ enters a membrane $m'_{i,j}$, $j = 1, \dots, h$, then the symbol \dagger is introduced and that string will never contribute to the output. Instead, if $X_t w$ enters a membrane $m'_{i,r}$, $r = 1, \dots, s$, then the computation can proceed. In this way, we can nondeterministically perform the simulation of a table from the corresponding membrane m_i in Π .

Inside membrane $m'_{i,r}$, for $r = 1, \dots, s$, we simulate the rules of the table t_r : we apply all the rules $A \rightarrow yX_{tar}(out)$ which can be applied and which correspond to rules $A \rightarrow y(tar)$ belonging to t_r , and in parallel we delete the symbol X_t . If some rules can be applied, then the rewritten string w' returns to membrane m'_i and the computation can proceed; if no rules can be applied, then no support symbols will be present in the string w and the computation halts in membrane m'_i . Observe that w' contains some possibly different symbols X_{tar} , with $tar \in \{here, in, out\}$, which describe the target indication of the corresponding applicable rules in t_r . We show now that if the application of a table in m_i leads to a deadlock state, then the simulation of that table leads either to a deadlock state in m'_i or to the introduction of the symbol \dagger . So, in both situations, no output will be generated.

Let us assume that both X_{here} and X_{in} are present in w' , then in parallel the rules $X_{here} \rightarrow X_t(in)$, $X_{in} \rightarrow X(in)$ introduce the symbols X , X_t , and the string enters a nondeterministically chosen membrane. Wherever the string is at the next step, the trap symbol \dagger will be introduced because of the rule $X \rightarrow \dagger(out)$ in $m'_{i,r}$ or $X_t \rightarrow \dagger(out)$ in $m'_{i,j}$.

In all other cases, that is when X_{here}, X_{out} or X_{in}, X_{out} or $X_{here}, X_{out}, X_{in}$ are simultaneously present in the string, a deadlock state occurs because in membrane m'_i the rules defined over such symbols have mixed target indications.

If the application of a table in m_i does not lead to a deadlock state, it means that in membrane $m'_{i,r}$ one or more occurrences of X_{tar} , for only one tar in the set $\{here, in, out\}$, will be introduced in the string w' . We show that also this situation is correctly simulated.

If the introduced symbols are X_{here} , then in membrane m'_i the rule $X_{here} \rightarrow X_t(in)$ will introduce the symbol X_t again, and the string is ready for a new simulation of a table from m_i .

If the introduced symbols are X_{in} , then in membrane m'_i the rule $X_{in} \rightarrow X(in)$ will introduce the symbol X again. If the string enters a membrane $m'_{i,j}$, then the computation proceeds, otherwise no output will be produced because of the introduction of \dagger in any other membrane.

If the introduced symbols are X_{out} , then in membrane m'_i the rule $X_{out} \rightarrow X(out)$ will introduce the symbol X again, and the string will exit the current membrane. In particular, if m'_i is the skin membrane of Π' , then no support symbol will be introduced, the string will exit the system and, if it is a terminal string, it will contribute to the output, otherwise it will be ignored.

Hence we can correctly simulate every computation in Π by a computation in

Π' , and it follows that $L(\Pi') = L(\Pi)$.

(iii). The inclusion $EParRP_n^k((T), nD) \subseteq EParRP_n^k((T), D)$ follows from the definitions. From (ii) we know that $EParRP_n^k((T), D)$ is included in $EParRP_*^{k+1}((M), D)$, and from Theorem 3 in Besozzi et al. (2002A) (which states that $EParRP_n^k((M), D) = EParRP_{7n}^{k+2}(M), nD)$) it follows that $EParRP_*^{k+1}((M), D) \subseteq EParRP_*^{k+3}((M), nD)$. So inclusion (iii) holds. \square

The following three equivalences can be proved by using, respectively, Corollary 3 in Besozzi et al. (2002A) (which states that $EParRP((M), D) = EParRP(M), nD)$), the definitions of parallel P systems and point (i) of Theorem 5 and, lastly, point (ii) of Theorem 5.

Corollary 6 $EParRP((M), D) = EParRP((M), nD) = EParRP((T), D) = EParRP((T), nD)$.

6 Final Remarks

We have considered several families of languages generated by parallel rewriting P systems, obtained by choosing different ways of rewriting strings and by considering the possibility of having deadlock states or not.

We have shown that the family of languages generated by extended Lindenmayer systems with tables is properly included in the family of languages generated by maximally parallel P systems with deadlock.

We have proved that, in the case of symbol parallelism, the possibility of having deadlock states does not modify the computational power of parallel P systems; in particular, we have also shown how to construct a system of depth 2 which does not have any deadlock state and is equivalent to a system of depth 2 which can have, instead, deadlock states.

Moreover, we have compared the power of different parallelism methods. We have shown that the family of languages generated by P systems with and without deadlock and with a maximal parallel application of rules coincides with the family of languages generated by P systems with and without deadlock and with table parallelism.

Several research topics remain to be investigated, for instance concerning comparisons with systems using parallelism methods not considered here (see Besozzi et al. (2002A)), with or without deadlock, with different interpretation of deadlock and, lastly, their relations with respect to Lindenmayer systems or other language generating devices.

References

- Besozzi, D., Ferretti, C., Mauri, G., Zandron, C., 2002(A). Parallel Rewriting P Systems with Deadlock, Pre-Proceedings of DNA8 Conference (M. Hagiya, A. Ohuchi, eds.), Hokkaido University, Japan, June 2002, pp. 171–183.
- Besozzi, D., Mauri, G., Zandron, C., 2002(B). Parallel Rewriting P Systems without Target Conflicts, Pre-Proceedings of MolCoNet Workshop on Membrane Computing, Curtea de Argeş, Romania, August 2002, pp. 103–118.
- Dassow, J., Păun, G., 1989. Regulated Rewriting in Formal Language Theory, Springer-Verlag, Berlin.
- Fajstrup, L., Goubault, E., Raussen, M., 1998. Detecting Deadlocks in Concurrent Systems, Concurrency Theory (CONCUR'98), LNCS 1466, pp. 332–347, Springer-Verlag, Berlin.
- Ferretti, C., Mauri, G., Zandron, C., Păun, G., 2001. On Three Variants of Rewriting P Systems, Theoretical Computer Science, to appear.
- Krishna, S. N., 2001. Languages of P systems: Computability and Complexity, PhD Thesis, IIT Madras.
- Krishna, S. N., Rama, R., 2001. A Note on Parallel Rewriting in P Systems, Bulletin of the EATCS, 73, pp. 147–151.
- Krishna, S. N., Rama, R., 2000. On the Power of P Systems Based on Sequential/Parallel Rewriting, Intern. J. Computer Math., 77, 1-2, pp. 1–14.
- Păun, G., 1998/2000. Computing with Membranes, Journal of Computer and System Sciences, 61 (2000), 1, pp. 108–143 (see also Turku Center for Computer Science-TUCS Report No 208, 1998, www.tucs.fi).
- Rozenberg, G., Salomaa, A. (Eds.), 1997. Handbook of Formal Languages, Springer-Verlag, Heidelberg.
- Rozenberg, G., Salomaa, A., 1980. The Mathematical Theory of L Systems, Academic Press, New York.
- Tanenbaum, A. S., 2001. Modern Operating Systems, Prentice Hall.
- Zandron, C., 2001. A Model for Molecular Computing: Membrane Systems, PhD Thesis, Università degli Studi di Milano, Dipartimento di Scienze dell'Informazione.