P and dP Automata: A Survey

Gheorghe Păun[1,2] and Mario J. Pérez-Jiménez[2]

[1] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, marper@us.es

Abstract. This is a quick survey of basic notions and results related to P automata (P systems with symport/antiport rules working in the accepting mode), with some emphasis on the recently introduced dP automata (a distributed version of the standard P automata), ending with some open problems and research topics which we find of interest in this area.

1 Introduction

Membrane computing is a branch of natural computing aiming to abstract computing models from the structure and the functioning of the biological cell; the basic model of this research area (usually called a *P system*) consists of a hierarchical arrangement of membranes which delimit compartments where multisets of objects evolve according to given rules inspired by biology. Some rules are mimicking the biochemical reactions, other rules correspond to processes specific to cells, such as the selective passage of chemicals across membranes, in the form of symport and antiport operations (couples of molecules pass together, in the same direction in the case of symport and in opposite directions in the case of antiport, through specific protein channels). This is the framework where the present paper is placed: cell-like models, with the multisets of objects processed by communication only (moving them across membranes), using symport and antiport rules. Such systems were initially used in the generative manner (one starts from an initial configuration and one proceeds by a maximally parallel use of rules until reaching a halting configuration, one where no rule can be applied; the contents of a designated membrane in the halting configuration is considered as the result of the computation).

Many variations of this basic model can be found in the membrane computing literature. We mention only the much investigated classes of tissue-like P systems and of spiking neural P systems. The reader is refereed to [15], [17], and to the domain website [21] for details.

The idea of using a P system in the accepting mode has appeared already "from the old times": start a computation by introducing a multiset in a specified membrane and, if (and only if) the computation halts, then this multiset is

accepted. In the systems using only communication rules, such as those based on symport/antiport rules, a string can also be accepted in a natural way: just arrange in a sequence the objects (described by symbols) taken from the environment by the system during a halting computation. This idea was followed first in [6] (the paper was presented during the Workshop on Membrane Computing, Curtea de Argeş, 2002) and, almost concomitantly, in [10].

The devices introduced in the first paper are called *P automata*. They are usual P systems with symport/antiport rules supplemented with certain features: a set of accepting configurations (called "states" in [6]) is given and a mapping which associates a string with a multiset. The computation proceeds as usual in a P system with symport/antiport rules and it is considered successful only if it halts in an accepting state. In each step, the system takes some objects from the environment, hence a sequence of multisets can be associated with a successful computation. This sequence is "translated" into a string by the mapping mentioned above. Several papers were devoted to these devices (in particular, characterizations of regular, context-free, and recursively enumerable languages were obtained, and complexity investigations were carried out); we refer to [5] for details, including references.

A simplified version of P automata was considered in [10]: successful computations are defined by halting only, and the mapping which passes from multisets (of objects introduced in the system) to strings is very simple – either all symbols are introduced in the accepted string (if several symbols are taken in the same step, then any permutation of them is introduced in the string, hence a set of strings can be associated with one computation), or only the objects from a given set, which is like in Chomsky grammars and Lindenmayer systems, where terminal and non-terminal symbols are considered and the strings in a language consists of only terminals.

From now on we will work only with P automata in the sense of [10]. We call *extended* the automata which consider terminal symbols (hence they discard the non-terminal ones). We will give precise definitions in the next section.

Note that any P system is a distributed parallel device, with several compartments/membranes working simultaneously, but the input (in the case of P automata) is taken from the environment only by the skin region. Looking for a computing model which can take parts of a global input and introduce them as "local" inputs in different components and then process these inputs separately in order to answer a "global question", so-called *dP systems* were recently introduced in [16]. In the general case, such systems consist of a given number of components in the form of a usual P system, of any type, which can have their separate inputs and communicate from skin to skin membranes by means of antiport rules like in tissue-like P systems. In this framework, communication complexity issues can be investigated, as in [12]. (Some previous proposals towards a communication complexity of P systems were made in [1], but mainly related to the communication effort in terms of symport/antiport rules in a usual P system, not an explicitly distributed one.) The case of P automata was considered in some details – and this leads to the notion of *dP automata*. The

possibility of accepting languages of various types in Chomsky hierarchy in a distributed way, using a bounded number of communication rules and also with some (linear) speed-up was proven.

The study of dP automata was continued in [9], by comparing their power with that of usual P automata and with families of languages in the Chomsky hierarchy. As expected, due to the distribution (and synchronization), dP automata are strictly more powerful than P automata. Also expected is the fact that each regular languages can be recognized by a P automaton.

In the present note, we recall the results from [16] and [9]. A theorem from [9] gives a representation of recursively enumerable (RE) languages starting from languages recognized by dP automata, similar to the representation of RE languages in terms of context-sensitive languages; it is not shown in [9] whether this new representation is non-trivial, in the sense that the relation between the family of languages recognized by dP automata and the family of context-sensitive languages is not settled (the inclusion is obvious, but it is not shown to be proper). We clarify here this point, by finding a context-sensitive language which cannot be accepted by a dP automaton. Along the paper as well as in the end of it, we formulate a series of open problems and research topics (especially about dP automata) which we find of interest.

2 dP Automata

We directly introduce the dP automata, by whose particularization we get the notion of a P automaton.

The reader is assumed to be familiar with basics of membrane computing, e.g., from [15], [17], and of formal language theory, e.g., from [19], [20].

In what follows, V^* is the free monoid generated by the alphabet V, λ is the empty word, $V^+ = V^* - \{\lambda\}$, and $|x|$ denotes the length of the string $x \in V^*$. REG, LIN, CF, CS, RE denote the families of regular, linear, context-free, context-sensitive, and recursively enumerable languages, respectively. As usual in membrane computing, the multisets over an alphabet V are represented by strings in V^*; a string and all its permutations correspond to the same multiset, with the number of occurrences of a symbol in a string representing the multiplicity of that object in the multiset. (We work here only with multisets of finite multiplicity.) The terms "symbol" and "object" are used interchangeably, all objects are here represented by symbols.

A *dP automaton* (of degree $n \geq 1$) is a construct

$$\Delta = (O, E, \Pi_1, \ldots, \Pi_n, R),$$

where:

(1) O is an alphabet (of objects);
(2) $E \subseteq O$ (the objects available in arbitrarily many copies in the environment);

(3) $\Pi_i = (O, \mu_i, w_{i,1}, \ldots, w_{i,k_i}, E, R_{i,1}, \ldots, R_{i,k_i})$ is a symport/antiport P system of degree k_i (O is the alphabet of objects, μ_i is a membrane structure of degree k_i, $w_{i,1}, \ldots, w_{i,k_i}$ are the multisets of objects present in the membranes of μ_i in the beginning of the computation, E is the alphabet of objects present – in arbitrarily many copies – in the environment, and $R_{i,1}, \ldots, R_{i,k_i}$ are finite sets of symport/antiport rules associated with the membranes of μ_i; the symport rules are of the form $(u, in), (u, out)$, where $u \in O^*$, and the antiport rules are of the form $(u, out; v, in)$, where $u, v \in O^*$; note that we do not have an output membrane), with the skin membrane labeled with $(i, 1) = s_i$, for all $i = 1, 2, \ldots, n$;

(4) R is a finite set of rules of the form $(s_i, u/v, s_j)$, where $1 \leq i, j \leq n, i \neq j$, and $u, v \in O^*, uv \neq \lambda$.

The systems $\Pi_1, \ldots, \Pi_n$ are called *components* of Δ and the rules in R are called *communication rules*. For a rule $(s_i, u/v, s_j)$, $|uv|$ is the *weight* of this rule.

Each component can take an input, work on it, and communicate with other components. The communication is done by means of rules in R, but, because the environment is common, the components can also communicate, in two steps, through the environment. In the constructions involved in the proofs of the results recalled below this latter possibility is systematically avoided, but from a formal point of view this raises already a research topic: Is any difference between the power and/or the efficiency of dP systems whose components are allowed and those whose components are not allowed to communicate through the environment? How this communication can be avoided? (Some suggestions are given in [16], e.g., to consider a "local environment" for each component, not accessible to other components.)

A halting computation with respect to Δ accepts the string $x = x_1 x_2 \ldots x_n$ over O if the components $\Pi_1, \ldots, \Pi_n$, starting from their initial configurations, using the symport/antiport rules as well as the inter-components communication rules, in the non-deterministic maximally parallel way, bring from the environment the substrings $x_1, \ldots, x_n$, respectively, and eventually halts.

The dP automata are synchronized devices, a universal clock exists for all components, marking the time in the same way for the whole dP automaton.

Three communication complexity measures were defined in [16], following [1], counting the number of communication steps (parameter $ComN$), of communication rules ($ComR$), or the total weight of communication rules ($ComW$) used during a computation. Based on these measures, the notions of *weak parallelizability* and of *efficient parallelizability* are introduced. For instance, a language $L \subseteq V^*$ is said to be (n, m)-*weakly ComX parallelizable*, for some $n \geq 2, m \geq 1$, and $X \in \{N, R, W\}$, if there is a dP automaton Δ with n components and there is a finite subset F_Δ of L such that each string $x \in L - F_\Delta$ can be written as $x = x_1 x_2 \ldots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the string x is accepted by Δ by a halting computation δ such that $ComX(\delta) \leq m$. A language L is said to be *weakly ComX parallelizable* if it is (n, m)-weakly $ComX$ parallelizable for some $n \geq 2, m \geq 1$.

Note that (i) the string is distributed in equal parts, modulo one symbol, to the components of the dP automaton (like in communication complexity area, [12]; one says that the string is distributed *in a balanced way*) and (ii) the communication complexity, in the sense of measure $ComX$, is bounded by the constant m. It is said nothing about the length of the computation, that is why a stronger version of parallelizability is introduced, the efficient one. In what follows, we allow the dP system to perform communications of an arbitrary complexity, while the length of the computation is not taken into consideration, hence we ignore these aspects.

Specifically, for a dP automaton Δ of degree n we define the language $L(\Delta)$, of all strings $x \in O^*$ such that we can write $x = x_1 x_2 \dots x_n$, with $||x_i| - |x_j|| \leq 1$ for all $1 \leq i, j \leq n$, each component Π_i of Δ takes as input the string $x_i, 1 \leq i \leq n$, and the computation halts.

Note again that, like in the communication complexity area, the string is distributed in equal parts, modulo one symbol, to the components of the dP automaton. This acts like a strong restriction for our devices. If this condition is not imposed, hence any decomposition of the string x can be considered, then a superlanguage of $L(\Delta)$ is obtained. Like in [16], [9], in what follows we only consider the balanced distribution case; the study of the unbalanced case remains a topic for future research (we will mention this question again in the last section).

We denote by LdP_n the family of languages $L(\Delta)$, for Δ of degree at most n. A dP automaton of degree 1 is a usual P automaton – of a non-extended type: all symbols are introduced in the accepted string. If a terminal set of objects is considered, then we obtain an extended P automaton (formally, we have a device $\Pi = (O, T, \mu, w_1, \dots, w_m, E, R_1, \dots, R_m)$, with $T \subseteq O$, working as a usual P automaton and considering only the symbols from T in the accepted strings and ignoring those from $O - T$). We denote by LP the family of languages recognized by non-extended P automata (hence $LP = LdP_1$) and by ELP the family of languages recognized by extended P automata. (Note that we ignore the weight of symport and antiport rules, but these parameters, usual when investigating symport/antiport P systems, can be considered also here.) If the subscript n in LdP_n is arbitrary, then we replace it by $*$.

A terminal alphabet can be considered also for dP automata, but this is not of much interest: $ELdP_1 = ELP$, which is known to equals RE.

3 On the Power of P Automata

Extended P automata were proved already in [10] to be computationally universal:

Theorem 1. $ELP = RE$.

Actually, rather simple P automata (e.g., with only one membrane), also working in the deterministic way, are shown to be able to simulate language accepting register machines. In turn, because in the non-extended case the number of

objects present in the system is comparable with the number of objects taken from the environment (the initial multisets are fixed), hence with the length of the accepted string, we immediately have:

Theorem 2. $LP \subseteq CS$.

However, we know no other result about non-extended P automata reported before the introduction of dP automata, which is somehow strange, because the power of non-extended P automata raise interesting (and intuitively non-trivial) problems.

Some of these problems were addressed in [9]; we will recall the respective results, after recalling an example, which can illustrate the way a P automaton works. The automaton (with six membranes) is given first formally and then in Figure 1, represented as usual in membrane computing (with the rules near the membranes with which they are associated).

$$
\begin{aligned}
\Pi &= (O, \mu, w_1, w_2, w_3, w_4, w_5, w_6, E, R_1, R_2, R_3, R_4, R_5, R_6),\\
O &= \{a, b, c, d, e, f, g, \#\},\\
\mu &= [\,[\,[\quad]_3[\quad]_4\,]_2[\quad]_5[\quad]_6\,]_1,\\
w_1 &= c,\\
w_2 &= de,\\
w_3 &= \lambda,\\
w_4 &= \#,\\
w_5 &= fgg,\\
w_6 &= \#,\\
E &= \{a, b, c, d\},\\
R_1 &= \{(c, out; aa, in),\ (a, out; c, in),\ (d, out; bb, in),\ (b, out; d, in)\},\\
R_2 &= \{(e, out),\ (ae, in),\ (d, out; fe, in),\ (be, in),\ (g, in),\ (\#, in),\ (\#, out)\},\\
R_3 &= \{(gab, in),\ (g, out)\},\\
R_4 &= \{(\#, out; ga, in),\ (\#, out; gb, in)\},\\
R_5 &= \{(f, out; c, in),\ (gg, out; d, in)\},\\
R_6 &= \{(\#, out; e, in),\ (eg, in)\}.
\end{aligned}
$$

This automaton recognizes the non-regular language $L(\Pi) = \{(a^2c)^s(b^2d)^s \mid s \geq 1\}$ – we denote it by L_1, for a later reference.

We start by bringing the object e out of membrane 2, at the same time with introducing two copies of a from the environment. If e will enter membrane 6, then the computation never halts, hence the object e should be "kept busy" by means of a copy of object a, which brings e back to membrane 2 (while the other copy of a exits, in exchange with one copy of object c). This process is repeated for a number of times and then, instead of going out, the object c enters membrane 5, releasing from here the object f. Together with e, object f enters membrane 2, releasing d. From now on, d plays the same role as c before and b

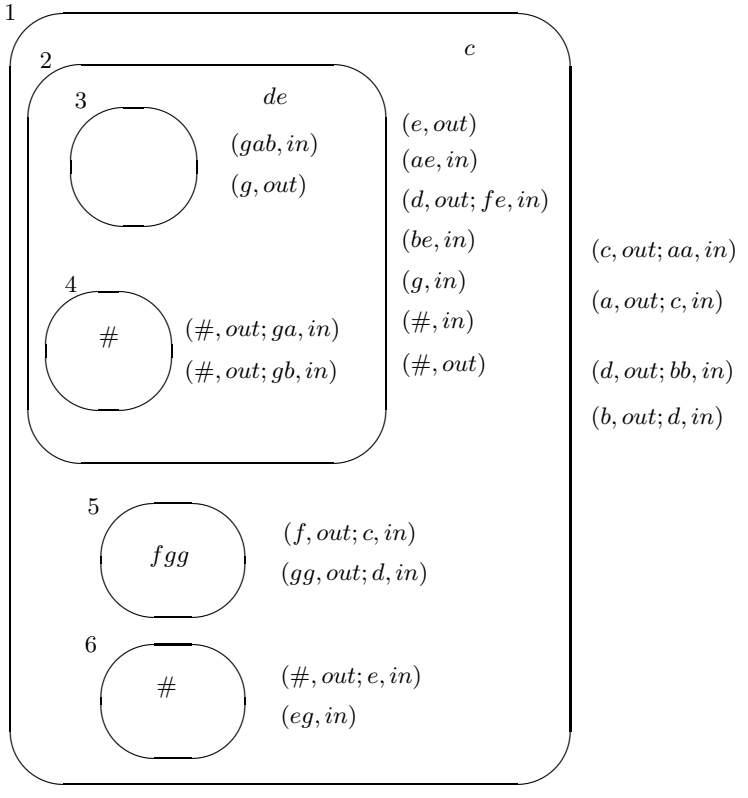

Fig. 1. A P automaton recognizing the language L_1

plays the role of a, hence we bring inside a string of the form $(bbd)^s$, $s \geq 1$. After a while, also d enters membrane 5, releasing the two copies of g. One is used for moving the object e inside membrane 6 without releasing the trap object #, while the other copy of g enters membrane 2, and start here checking whether the number of copies of a and b stored here are equal. If this is not the case, then g together with exceeding a or with an exceeding b enters membrane 4, and the trap object # is brought to region 2, hence the computation never stops.

Therefore, we have $L_1 \in LP$. The idea of the system in Figure 1 can be extended so that we can check the equality of three blocks of repeated symbols, hence also non-context-free languages can be obtained. We summarize these remarks as:

Theorem 3. *LP contains linear non-regular, as well as non-context-free languages.*

Thus, the non-extended P automata can recognize "complex" languages – but they fail to recognize other "simple" languages. Here are two necessary conditions for a language to be in LP proved in [9].

We start with an easy result, refuting however many languages.

Lemma 1. *For every language $L \subseteq V^*, L \in LP$, which is not regular there is a string $w \in L$ which can be written in the form $w = w_1abw_2$, for some $w_1, w_2 \in V^*$ and $a, b \in V$ (not necessarily distinct) such that $w_1baw_2 \in L$.*

This lemma implies, for instance, that the linear language

$$L_2 = \{(ab)^n(ac)^n \mid n \geq 1\}$$

is not in LP. Actually, a more general consequence of Lemma 1 is drawn in [9]:

Theorem 4. *All families of languages which include strictly the family of regular languages and are closed under λ-free morphisms contain languages which are not in LP.*

We pass now to the second necessary condition for a language to be in LP.

Lemma 2. *Let V be an alphabet with at least two elements and $f : V^* \longrightarrow V^*$ an injective mapping. The language $L_f = \{wf(w) \mid w \in V^*\}$ is not in the family LP.*

The proof is based on the observation that the number of configurations of a P automaton which has brought inside m symbols is bounded by a polynomial in m, but there are more than 2^m different strings of length m over an alphabet with more than two symbols (hence exponentially many), which makes impossible the matching between the two halves of the strings. We will extend this proof idea to dP automata in the next section.

As a consequence of the previous lemma, for instance, the context-sensitive language, $L_3 = \{wf(w) \mid w \in \{a, b\}^*\}$ for $f(a) = a', f(b) = b'$, is not in LP.

Pleasantly enough (and somewhat expected), P automata can recognize all regular languages:

Theorem 5. *$REG \subset LP$.*

4 On the Power of dP Automata

Let us first note that the language L_2 is in LdP_2 and the same is true for L_3; this language is recognized by the dP automaton (of degree 2, with arbitrarily many communications) indicated in Figure 2, hence we have

Theorem 6. *$LdP_n - LP \neq \emptyset$ for all $n \geq 2$.*

The following theorem is classic in formal language theory – see, e.g., [20]:

Theorem 7. *For every language $L \in RE, L \subseteq V^*$, there is a language $L' \in CS$ and two symbols $a, c \notin V$ such that: (i) $L' \subseteq L\{c\}a^*$, (ii) for each $w \in L$ there is $i \geq 0$ such that $wca^i \in L'$.*

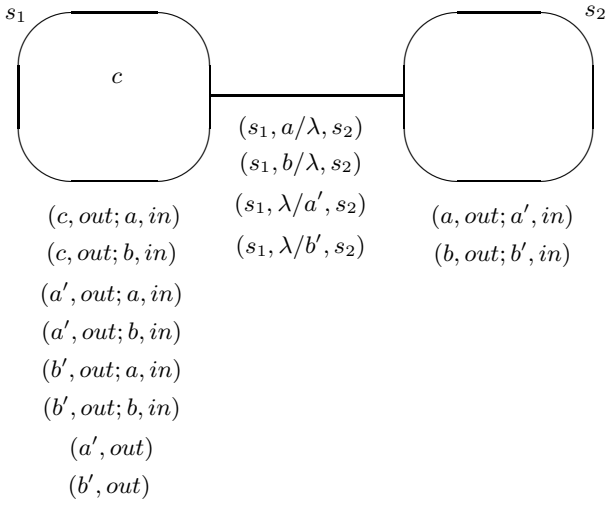

Fig. 2. A dP automaton accepting the language L_3

Otherwise stated, the two languages are "the same" up to a tail of arbitrary length added to strings in L.

Because the initial configuration of a dP automaton is given and the objects brought into the system from the environment are part of the recognized string, the workspace of the automaton is linearly bounded with respect to the string, hence Theorem 2 can be extended to:

Theorem 8. *$LdP_* \subseteq CS$.*

In [9] it is conjectured that the above inclusion is proper. We confirm here this hypothesis:

Lemma 3. *The language $L_4 = \{(ww')^s \mid w \in \{a,b\}^+, s \geq 2\}$, where w' is obtained from w by priming the symbols a and b, is not in the family LdP_*.*

Proof. Assume that $L_4 = L(\Delta)$ for some dP automaton Δ with n components, $\Pi_1, \ldots, \Pi_n, n \geq 2$. Consider the sublanguage H_n of L_4 consisting of strings with n blocks ww', i.e.,

$$H_n = \{(ww')^n \mid w \in \{a,b\}^+\} \subset L_4.$$

Because of the balanced distribution of inputs to the n components of Δ, each Π_i has to take from the environment a string ww'. Consider the strings w of length m, for some arbitrarily large m, and examine the state of the dP automaton in the moment when component Π_1 has "read" from the environment the symbols of w. (There is a step when exactly the symbols of w were introduced in Π_1: the next symbol is primed and, if it enters the system at the same time with a symbol

from w, which is not primed, then a substring $c'd$ can appear, $c, d \in \{a, b\}$, which is contradictory.)

At this moment, the whole dP automaton contains a number of symbols bounded from above by $t_0 + m + 2m(n-1)$, where t_0 is the number of objects present in the initial configuration, m objects are introduced by Π_1 and each of the other $n-1$ components have introduced at most $2m$ objects each. If these objects were identical, then they are distributed in the regions of the system – assume that their number is k – in a number of ways which is bounded from above by $(t_0+m+2m(n-1))^k$. Here we have four objects a, b, a', b', as well as some possibly different objects present in the initial configuration. In total, a fixed number, let us say r. Thus, all these objects can be distributed in the k regions of Δ in a number of ways which is at most $(t_0 + m + 2m(n-1))^{kr}$. Consequently, there are polynomially many configurations of Δ reached after having the string w read by Π_1.

However, there are 2^m strings of length m over $\{a, b\}$, hence, for a large enough m, there are strings $w_1, w_2 \in \{a, b\}^+$ of length m such that $w_1 \neq w_2$, but the dP automaton reaches the same configuration after reading w_1 or w_2. This means that after reading w_1, Π_1 can continue by reading w_2' and the computation stops like when Π_1 started by reading w_2, hence a string $w_1 w_2' z$ is accepted (we do not care about the form of z), which is not in L_4, a contradiction. □

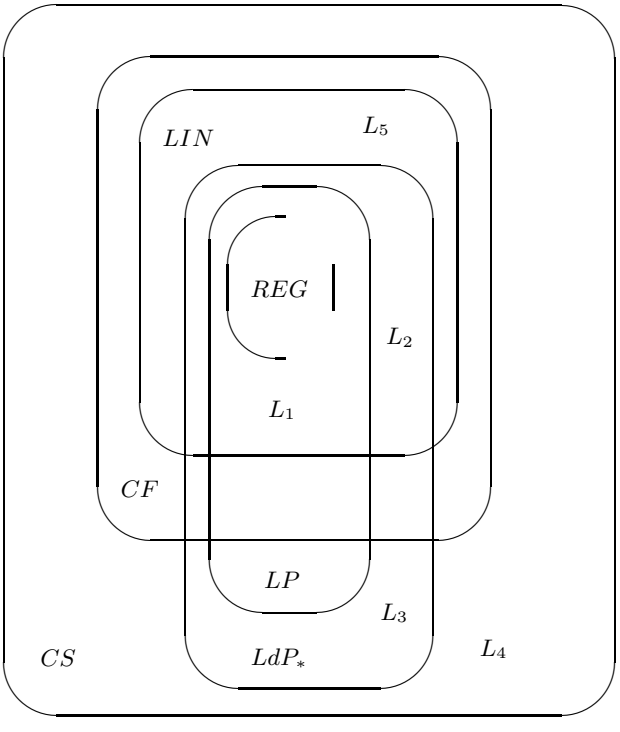

Fig. 3. The place of the families LP and LdP in Chomsky hierarchy

Therefore, the following counterpart of Theorem 7, proved in [9], is of interest:

Theorem 9. *For every language $L \in RE, L \subseteq V^*$, there is a language $L' \in LdP_2, X \in \{N, R, W\}$, and an alphabet U disjoint of V such that: (i) $L' \subseteq LU^*$, (ii) for each $w \in L$ there is $y \in U^*$ such that $wy \in L'$.*

We believe that a similar result is valid also for languages recognized by P automata, but this time with the "tail" placed in the left hand of the string. Specifically, a result of the following form is *conjectured* in [9]:

For every language $L \in RE, L \subseteq V^*$, there is a language $L' \in LP$, and an alphabet U disjoint of V as well as $c, e \notin V \cup U$ such that: (i) $L' \subseteq U^*\{c\}L\{e\}$, (ii) for each $w \in L$ there is $y \in U^*$ such that $ycwe \in L'$.

Moreover, it is conjectured that

$$L_5 = \{w\ mi(w) \mid w \in \{a, b\}^*\} \notin LdP_*,$$

hence, actually, there are linear languages which cannot be accepted by dP automata ($mi(x)$ denotes the mirror image of the string x).

The results from the previous two sections are synthesized in the diagram from Figure 3, based on a similar diagram from [9]; the languages L_1, L_2, L_3, L_4 are specified above and L_5 is only conjectured.

5 Further Research Topics

Of course, many problems and research topics remain to be considered. Several were mentioned in [16] and [9], many others can be imagined.

We recall first some questions from [16]. For instance, we mentioned the notions of parallelizability (recognizing the strings of a language by using a finite number of communication steps) and of efficient parallelizability (to also speed-up the computation, in comparison with a non-distributed automaton). These problems make sense both for the case of the balanced distribution of the string (which is taken as an hypothesis in communication complexity area) and for the arbitrary distribution (which makes sense from the computational complexity point of view). Is it a difference between the two cases? (Anyway, the speed-up obtained by distribution on a given number of "processors" cannot be more than linear, but this is still of interest in some practical cases.) In general, it is of interest to transfer to dP systems, in particular, to dP automata, notions and techniques currently used in communication complexity area, [12].

Then, focusing on P and dP automata as language accepting devices, there are many problems of a classic language theory type which are natural to be raised. For instance, the properties of the language families LP, LdP_*, and $LdP_n, n \geq 2$ (e.g., closure, decidability, descriptional complexity) are of interest (especially because they are not equal to families in Chomsky hierarchy). A related issue is to compare these families with other language families, such as Lindenmayer languages [18], Marcus contextual languages [14], families from the regulated rewriting area [7], etc. Does the number of components

induce an infinite hierarchy of the recognized languages? (We only know that $LP = LdP_1 \subset LdP_2 \subseteq LdP_3 \subseteq \ldots \subseteq LdP_*$.) Are there languages which can be recognized by a dP automaton of degree n but not by an automaton of a greater degree? (The problem makes sense only for the balanced case.)

A large research area appears if we want to accept multisets instead of strings, and the problems appear already from the definition. We recall from [9] some hints in this respect.

One way to accept multisets is to reduce this case to accepting strings, taking into account that a multiset can be represented as a string. However, several questions appear in this framework. Consider an alphabet of objects, $A = \{a_1, a_2, \ldots, a_n\}$ and a multiset $M : A \longrightarrow \mathbf{N}$ over A. Any string $w \in A^*$ such that $\Psi_A(w) = (M(a_1), \ldots, M(a_n))$ represents the multiset M (Ψ_A is the Parikh mapping associated with A). Otherwise stated, all permutations of such a string w represent the same multiset. Is the permutation we choose significant from the point of view of accepting it by means of a dP automaton? A way to decrease this wildness of equivalent representations is to look for specified permutations of a string. In particular, we can take the *canonical representation* of M, that is $w_A(M) = a_1^{M(a_1)} \ldots a_n^{M(a_n)}$, hence based on the given ordering of the elements of A. However, we can take any other ordering of A as initially given, and the multiset is the same, but then the canonical representation will be different. Is this important from the point of view of accepting a multiset, by means of a string representation of it, by a dP automaton? A different way of using a P automaton, hence also a dP automaton, in order to recognize a multiset is to start by introducing the multiset in a specified region – to be closer to the case of string recognition, let us assume that this is the skin region – and to accept it if the computation halts. The question now is how to distribute the multiset among the components of the dP automaton, and again we have the two possibilities mentioned above: distributing the objects in the ordering imposed by a given ordering of the alphabet A, and distributing the objects of the multiset in an arbitrary manner – again with two cases for each possibility: a balanced distribution or an arbitrary distribution. (Balanced here is defined in terms of the cardinality of multisets, which is consistent with the definition of a balanced distribution for strings.)

A similarly large panoply of research issues can be based on using different ingredients and features in the considered automata. For instance, what about working in the asynchronous manner or with other types of parallelism, different from the maximal one considered above? What about taking some suggestions from [2], and consider dP automata with identical (or similar, e.g., of the same degree) components, maybe surrounded by separate environments? Then, we can add further tools for controlling the computations, such as promoters, inhibitors, channel states (for the inter-components communication rules), and so on. The use of promoters/inhibitors is particularly attractive, because they can help in easily halting the computations.

For instance, the "difficult" language $L_6 = \{a^n b^{2^n} \mid n \geq 1\}$ (it is non-semilinear) can be recognized by a dP automaton of degree 2 as that in

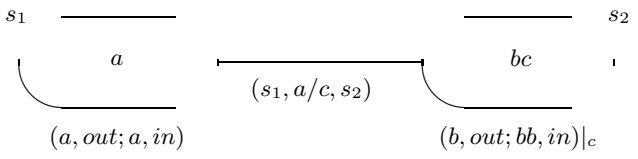

Fig. 4. A simple dP automaton with promoters

Figure 4, where $(b, out; bb, in)|_c$ means that c is a promoter of the antiport rule $(b, out; bb, in)$, the rule can be applied only if (at least one occurrence of) c is present in the membrane; the promoter is not involved in the rule (is not "consumed", it can promote at the same time any number of different rules). The automaton takes the input in a non-balanced way: a^n is read by the first component and b^{2^n} by the second one. The computation stops when the promoter c goes to the first component, in exchange of the unique object a present here.

The power of the promoter is visible. We do not know whether this feature can be removed, or whether the input can be read in a balanced way (probably not, hence this can be an example of a language which can be recognized only in the non-balanced way); note also that we perform only one communication.

Instead of using c as a promoter, we can use a as an inhibitor of the rule $(b, out; bb, in)$ and the work of the system is similar.

There also are several research issues related to the computational complexity of P and dP automata (for P automata, such investigations were already done, e.g., in [4]), or dealing with infinite strings or infinite alphabets (some references for P automata are [11], [8]). And, of course, we can also take into consideration the descriptional complexity, especially the weight of symport and antiport rules.

Like in [9], we conclude with the belief that P and dP automata deserve further research efforts.

Acknowledgements

Work supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. Adorna, H., Păun, G., Pérez-Jiménez, M.J.: On communication complexity in evolution-communication P systems. In: Proc. 8th Brainstorming Week on Membrane Computing, Sevilla, and Romanian J. Information Theory and Applications (February 2010) (to appear)
2. Colomer, M.A., Lavín, S., Marco, I., Margalida, A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Sanuy, D., Serrano, E., Valencia-Cabrera, L.: Studying the evolution of Pyrenean Chamois by using P systems. In: Pre-proc. Conf. on Membrane Computing, CMC11, Jena, Germany (August 2010)
3. Csuhaj-Varjú, E.: P automata. In: Mauri, G., et al. (eds.) WMC 2004. LNCS, vol. 3365, pp. 19–35. Springer, Heidelberg (2005)

4. Csuhaj-Varjú, E., Ibarra, O.H., Vaszil, G.: On the computational complexity of P automata. Natural Computing 5, 109–126 (2006)
5. Csuhaj-Varjú, E., Oswald, M., Vaszil, G.: P automata. In: [17], ch. 6, pp. 144–167
6. Csuhaj-Varjú, E., Vaszil, G.: P automata or purely communicating accepting P systems. In: Păun, G., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) WMC 2002. LNCS, vol. 2597, pp. 219–233. Springer, Heidelberg (2003)
7. Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer, Berlin (1989)
8. Dassow, J., Vaszil, G.: P finite automata and regular languages over countably infinite alphabets. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 367–381. Springer, Heidelberg (2006)
9. Freund, R., Kogler, M., Păun, G., Pérez-Jiménez, M.J.: On the power of P and dP automata. In: Annals of Bucharest University. Mathematics-Informatics Series (2010) (in press)
10. Freund, R., Oswald, M.: A short note on analysing P systems. Bulletin of the EATCS 79, 231–236 (2002)
11. Freund, R., Oswald, M., Staiger, L.: ω-P automata with communication rules. In: Martin-Vide, C., et al. (eds.) WMC 2003. LNCS, vol. 2933, pp. 203–217. Springer, Heidelberg (2004)
12. Hromkovic, J.: Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing. Springer, Berlin (1997)
13. Oswald, M.: P Automata. PhD Thesis, TU Vienna (2003)
14. Păun, G.: Marcus Contextual Grammars. Kluwer, Dordrecht (1997)
15. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
16. Păun, G., Pérez-Jiménez, M.J.: Solving problems in a distributed way in membrane computing: dP systems. Int. J. of Computers, Communication and Control 5(2), 238–252 (2010)
17. Păun, G., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing. Oxford University Press, Oxford (2010)
18. Rozenberg, G., Salomaa, A.: The Mathematical Theory of L Systems. Academic Press, New York (1980)
19. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, 3 volumes. Springer, Berlin (1998)
20. Salomaa, A.: Formal Languages. Academic Press, New York (1973)
21. The P Systems Website, `http://ppage.psystems.eu`