dP Automata versus Right-Linear Simple Matrix Grammars

Gheorghe Păun[1,2], 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

Summary. We consider dP automata with the input string distributed in an arbitrary (hence not necessary balanced) way, and we investigate their language accepting power, both in the case when a bound there is on the number of objects present inside the system and in the general case. The relation with right-linear simple matrix grammars is useful in this respect. Some research topics and open problems are also formulated.

1 Introduction

dP automata are a class of computing devices considered in membrane computing area in order to have a distributed language accepting machinery, with the strings to recognize being split among the components of the system and with these components working in parallel on the input strings. In the general case, dP systems consist of a given number of components in the form of a usual symport/antiport P system, which can have their separate inputs and communicate from skin to skin membranes by means of antiport rules like in tissue-like P systems. Such devices were introduced in [7] and further investigated in [3], [8], [9], mainly comparing their power with that of usual P automata and with families of languages in the Chomsky hierarchy. In the basic definition and in all these papers, following the style of the communication complexity area (see, [4]), the so-called *balanced* mode of introducing the input string is considered: the string is split in equal parts, modulo one symbol, and distributed among components.

Here we consider the general case, with no restriction on the input string distribution; each component just takes symbols from the environment when it can do it it, without any restriction on their number. This is a very natural and general set-up, which, however, was only incidentally investigated so far. Two cases are distinguished: with a bound on the size of the system (on the total number of objects present inside) and without such a bound. Both cases are naturally related to a classic family of regulated grammars, the simple matrix grammars of [5] (see also [2]). Actually, as expected, right-linear simple matrix grammars are closely related to dP automata, and we will examine below this connection (looking for mutual simulations among the two types of language identifying machineries). This connection was already pointed out in [8], where the conjecture was formulated that, in the same way as a usual finite automaton can be simulated by a P automaton, a right-linear simple matrix grammar can be simulated by a dP automaton. We confirm here this conjecture (in the general, not the balanced case).

2 Formal Language Theory Prerequisites

The reader is assumed to have some familiarity with basics of membrane computing, e.g., from [6], [10], and of formal language theory, e.g., from [2], [11], but we recall below all notions necessary in the subsequent sections.

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.

Essential below will be the right-linear simple matrix grammars introduced in [5]. Such a grammar of degree $n \geq 1$ is a construct of the form $G = (N_1, \ldots, N_n, T, S, M)$, where $N_1, N_2, \ldots, N_n, T$ are pairwise disjoint alphabets (we denote by N the union of $N_1, \ldots, N_n$), $S \notin T \cup N$, and M contains matrices of the following forms:

(i) $(S \to x), x \in T^*$,
(ii) $(S \to A_1 A_2 \ldots A_n),\ A_i \in N_i, 1 \leq i \leq n$,
(iii) $(A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n),\ A_i, B_i \in N_i, x_i \in T^*, 1 \leq i \leq n$,
(iv) $(A_1 \to x_1, \ldots, A_n \to x_n),\ A_i \in N_i, x_i \in T^*, 1 \leq i \leq n$.

A derivation starting with a matrix of type (ii) continues with an arbitrary numbers of steps which use matrices of type (iii) and ends by applying a matrix of type (iv).

We denote by $L(G)$ the language generated in this way by G and by RSM_n the family of languages $L(G)$ for right-linear simple matrix grammars G of degree at most n, for $n \geq 1$. The union of all these families is denoted by RSM_*. The strict inclusions $RSM_n \subset RSM_{n+1}, n \geq 1$, are known. Moreover, $REG = RSM_1$, $RSM_* \subset CS$, RSM_* is incomparable with LIN and CF, all languages in RSM_* are semilinear, and this family is closed under union, intersection with regular languages, direct and inverse morphisms (but not under intersection, complement and Kleene $+$).

Clearly, a normal form can be easily found for these grammars: in matrices of type (iii) we can ask to have $x_i \in T \cup \{\lambda\}, 1 \leq i \leq n$, and in matrices of type (iv) we can have $x_i = \lambda$ for all $1 \leq i \leq n$.

3 dP Automata

We introduce now the computing devices we investigate in this paper, also giving a relevant example.

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.

Using a rule $(u, in), (u, out)$ associated with a membrane i means to bring in the membrane, respectively to send out of it the multiset u; using a rule $(u, out; v, in)$ associated with a membrane i means to send out of the membrane the objects of multiset u and, simultaneously, to bring in the membrane, from the region surrounding membrane i, the objects of multiset v. A communication rule $(s_i, u/v, s_j)$ moves the objects of u from component Π_i to component Π_j, simultaneously with moving the objects in the multiset v in the opposite direction.

Each component Π_i can take symbols from the environment, work on them by using the rules in sets $R_{i,1}, \ldots, R_{i,k_i}$, and communicate with other components by means of rules in R.

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. A problem appears in the case when several objects are read at the same time from the environment, by several rules or by a single rule of the form $(u, out; v, in)$, with $|v| \geq 2$; in such a case any permutation of the symbols brought in the system in the same step are considered as a valid substring of the input string (thus, a computation can recognize several strings, differing to each other by permutations of certain substrings). Note that we impose here no condition on the relative lengths of strings $x_1, x_2, \ldots, x_n$ (as it is done in previous papers dealing with dP automata, under the influence of communication complexity area). We denote by $L(\Delta)$ the language of all strings recognized by Δ in this way, and by LdP_n the family of languages $L(\Delta)$, for Δ of degree at most $n \geq 1$. The union of all these families is denoted by LdP_*.

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. When the system has only one component, then we obtain the usual notion of a P automaton, as investigated in a series of papers (mainly in the extended version, with a terminal alphabet of objects – see the respective chapter in [10] and the references therein). We denote by LP the family of languages recognized by P automata. Hence, $LP = LdP_1$ and, from [3], it is known that $REG \subset LP \subset CS$ and LP is incomparable with CF.

We consider now a somewhat surprising example, of a dP automaton of degree 2, generating a complex language, $L_1 = \{ww \mid w \in \{a, b\}^*\}$. The automaton is given in Figure 1, in the standard way of representing a dP automaton. We have $O = \{a, b, c_1, c_2, d, \#\}$ and $E = \{a, b\}$.

All antiport rules which bring objects from the environment are of weight one, hence the number of objects present in the system is constant, four in each component. In the first step, objects d release c_2a in the skin region of the first component and c_1a in the second. Each symbol a can bring either an a or a b from the environment and, at the same time, the objects c_1, c_1 are interchanged between the two components (otherwise, they release the trap object $\#$, which will oscillate forever across membranes $(1, 1)$, respectively, $(2, 1)$, and the computation never stops). With $c_1\alpha, \alpha \in \{a, b\}$, in the first component and $c_2\beta, \beta \in \{a, b\}$, in the second one, the only continuation which does not release the trap object is possible when $\alpha = \beta$, by using the communication rule $(s_1, c_1\alpha/c_2\alpha, s_2)$ (if one of the symbols α, β brings new symbols from the environment, the corresponding c_1, c_2 should enter the membrane $(1, 2)$ or $(2, 2)$, bringing out the object $\#$). We obtain a configuration as that we started with, hence the process can be iterated. If, at any moment when c_2 is in Π_1 and c_1 is in Π_2, one of the rules $(c_2\alpha, in), \alpha \in \{a, b\}$, is used in the first component, or $(c_1\alpha, in), \alpha \in \{a, b\}$, is used in the second component, then this should be done simultaneously in both components, otherwise again one of c_1, c_2 has to release the trap object. In conclusion, the strings read from the environment by the two components are identical, hence $L(\Delta) = L_1$.

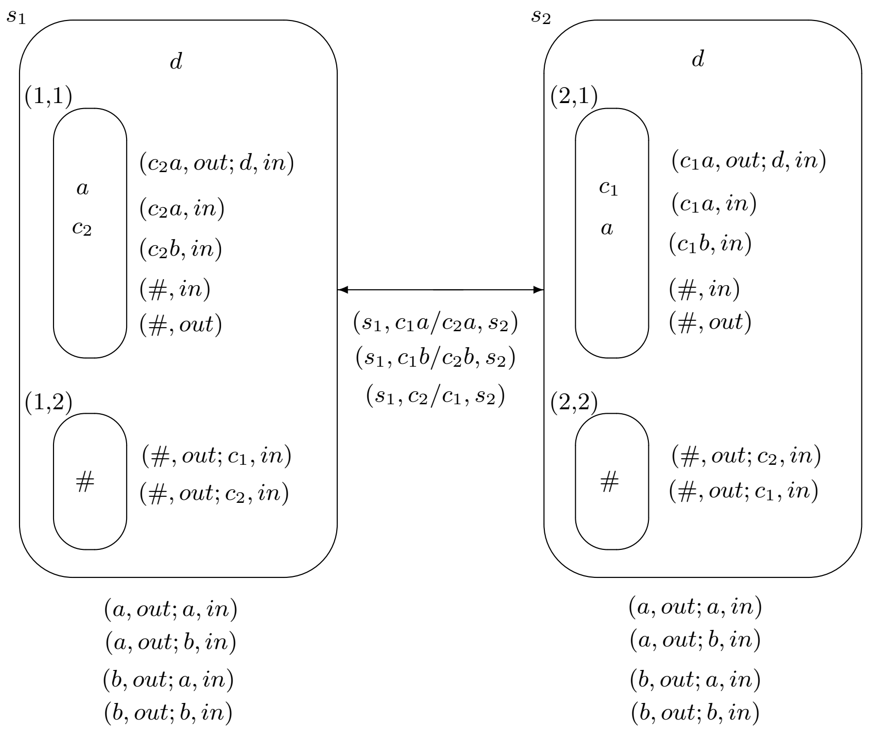


Fig. 1. A dP automaton recognizing the language L_1.

Note the important facts that the system reads the input in a balanced way and that it is *bounded*, the total number of objects present inside is always bounded by a constant (8 in our case) given in advance. This last characteristics is important, so that we denote by $LdP_n^b, n \geq 1$, the family of languages recognized by bounded dP automata of degree at most n; when n is not specified, we replace it by $*$.

4 The Power of dP Automata

We start by reformulating in a more general way a result already suggested by a proof in [9].

Theorem 1. *$LdP_n^b \subseteq RSM_n$, for all $n \geq 1$.*

Proof. Let Δ be a dP automaton of degree n (with the set of objects O) which is bounded. Then, the set of all its configurations is finite. Let $\sigma_0, \sigma_1, \ldots, \sigma_p$ be this set, with σ_0 being the initial configuration. We construct the following right-linear simple matrix grammar:

$$\begin{aligned}
G &= (N_1,\dots,N_n,O,S,M), \text{ with}\\
N_i &= \{(\sigma_j)_i \mid 0 \le j \le p\},\ i = 1,2,\dots,n,\\
M &= \{(S \to (\sigma_0)_1(\sigma_0)_2\dots(\sigma_0)_n)\}\\
&\quad \cup \{(\sigma_i)_1 \to \alpha_1(\sigma_j)_1,\dots,(\sigma_i)_n \to \alpha_n(\sigma_j)_n) \mid\\
&\qquad \text{from configuration } \sigma_i \text{ the dP automaton } \Delta \text{ can pass to}\\
&\qquad \text{the configuration } \sigma_j \text{ by a correct transition, taking from the}\\
&\qquad \text{environment the objects } \alpha_1,\dots,\alpha_n \text{ by its components, where}\\
&\qquad \alpha_s \in O \cup \{\lambda\}, 1 \le s \le n\}\\
&\quad \cup \{(\sigma_h)_1 \to \lambda,\dots,(\sigma_h)_n \to \lambda) \mid \sigma_h \text{ is a halting configuration}\}.
\end{aligned}$$

Note that all nonterminals in the rules of a matrix contain the same "core information", namely the current configuration of the system, hence the complete control of the system working is obtained in this way. The equality $L(\Delta) = L(G)$ is obvious. □

This result cannot be extended to arbitrary dP automata. Actually, we have:

Theorem 2. $LdP_2 - RSM_* \neq \emptyset$.

Proof. Let us consider the following dP automaton:

$$\begin{aligned}
\Delta &= (O,E,\Pi_1,\Pi_2,R), \text{ with}\\
O &= \{a,c,d,e,f,\#\},\\
E &= \{a,c,d,e\},\\
\Pi_1 &= (O,[\ \ ]_{s_1},f,E,\{(f,out;a,in),\ (a,out;aa,in)\}),\\
\Pi_2 &= (O,[\,[\ \ ]_{(2,1)}\,]_{s_2},E,\{(f,out;d,in),\ (a,out;c,in),\ (d,out;e,in)\},\\
&\qquad\qquad \{(f,out;f,in)\}),\\
R &= \{(s_1,a/\lambda,s_2)\}.
\end{aligned}$$

For an easier examination of the work of the system, we also represent it graphically, in Figure 2.

Let us look for strings accepted by this dP automaton which are of the form $a^i dc^j e$, for some $i,j \ge 1$.

After introducing the symbol a in the first component, let us assume that for $n \ge 0$ steps we use here the rule $(a,out;aa,in)$, hence we produce 2^n copies of a in Π_1, while the second component uses the rule $(f,out;f,in) \in R_{(2,1)}$. Suppose now that $p \ge 0$ copies of a remains in the first component and the others $r = 2^n - p$ are moved to the second component. Here, all r copies of a must go out, in exchange of objects c, hence the string read by the second component starts with c^r. At the same time or one step before, the second component must introduce the symbol d. This object becomes immediately e, hence the exchange of a for c should be done either in the same step with reading d or at the same time with reading e in

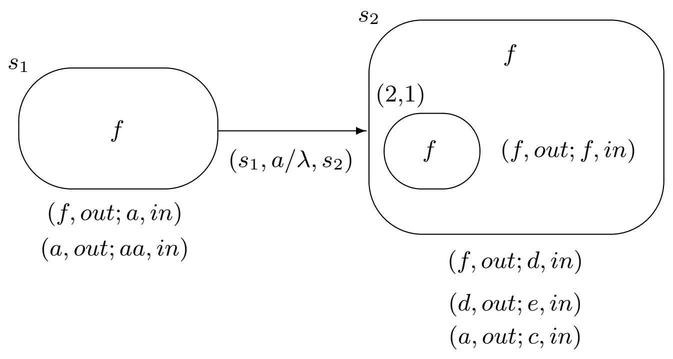


Fig. 2. A dP system recognizing a language not in RSM_*

the second component (because any permutation of the objects is allowed in the string, either variant is possible). However, after e, we do not want to have any symbol, hence all copies of a were already moved to the second component, and thus the work of the first component stops. When introducing the symbol d in the second component, the p copies of a from the first component cannot use the rule $(a, out; aa, in)$, but they must come immediately in the second component, to introduce c here at the same time with introducing e. Therefore, if the string has the form $a^i dc^j e$, then $i = j = 2^n$ for some $n \geq 0$ ($n = 0$ is obtained if the unique a introduced in the first step in Π_1 is immediately sent to component Π_2).

Consequently, $L(\Delta) \cap a^* dc^* e = \{a^{2n} dc^{2n} e \mid n \geq 0\}$, which is not in RSM_*, hence also $L(\Delta)$ is not in RSM_*: this family is closed under intersection with regular languages and contains only semilinear languages. □

Note that the previous construction takes the input string in an almost balanced way, and, if in the first step, the first component uses a rule $(f, out; dea, in)$ instead of $(f, out; a, in)$, then we have a balanced functioning, hence the result in the previous theorem holds true also for the balanced way of defining the recognized string.

We pass now to the counterpart of Theorem 1 announced above.

Theorem 3. *$RSM_n \subseteq LdP^b_{n+1}$, for all $n \geq 1$.*

Proof. Let us consider a right-linear simple matrix grammar $G = (N_1, \ldots, N_n, T, S, M)$ as introduced in Section 2, with the alphabets $N_1, N_2, \ldots, N_n$ (their union is denoted by N) and T. Matrices of the form (i), $(S \to x), x \in T^*$, can be replaced by matrices of forms (ii), (iii) and (iv), in an obvious way, hence we assume that we do not have such matrices. We assume all matrices labeled in a one-to-one way; let $m_j : (A_1 \to x_1 B_1, \ldots, A_n \to x_n B_n)$, with $1 \leq j \leq k$, be all matrices of type (iii), with $A_i, B_i \in N_i, x_i \in T^*, 1 \leq i \leq n$. Similarly, let $m_j : (A_1 \to x_1, \ldots, A_n \to x_n)$, with $k + 1 \leq j \leq p$, be all matrices of type (iv), with $A_i \in N_i, x_i \in T^*, 1 \leq i \leq n$. Without any loss of the generality we can assume that all strings x_i in these matrices are from $T \cup \{\lambda\}$.

For each matrix, of any form, $m_j : (A_1 \to u_1, \dots, A_i \to u_i, \dots, A_n \to u_n)$, let us consider the symbol $[m_j, A_i \to u_i]$ (thus identifying the matrix and its ith rule), and let $X_j(i)$ be a shorthand for it. Consider the alphabets

$$M_i = \{X_j(i) \mid 1 \leq j \leq p\}, \text{ for all } 1 \leq i \leq n.$$

We also denote by M'_i the alphabet of primed symbols in M_i.

For a matrix $m_j : (A_1 \to x_1B_1, \dots, A_n \to x_nB_n)$ of type (iii), let us denote $lhs_j = A_1A_2 \dots A_n$ and $rhs_j = B_1B_2 \dots B_n$. Similarly, for a matrix $m_j : (A_1 \to x_1, \dots, A_n \to x_n)$ of type (iv), we denote $lhs_j = A_1A_2 \dots A_n$.

If $rhs_j = lhs_k$, then we write $m_j \rightsquigarrow m_k$. Similarly, we write $S \rightsquigarrow m_j$ if $(S \to A_1A_2 \dots A_n) \in M$ and $A_1A_2 \dots A_n = lhs_j$.

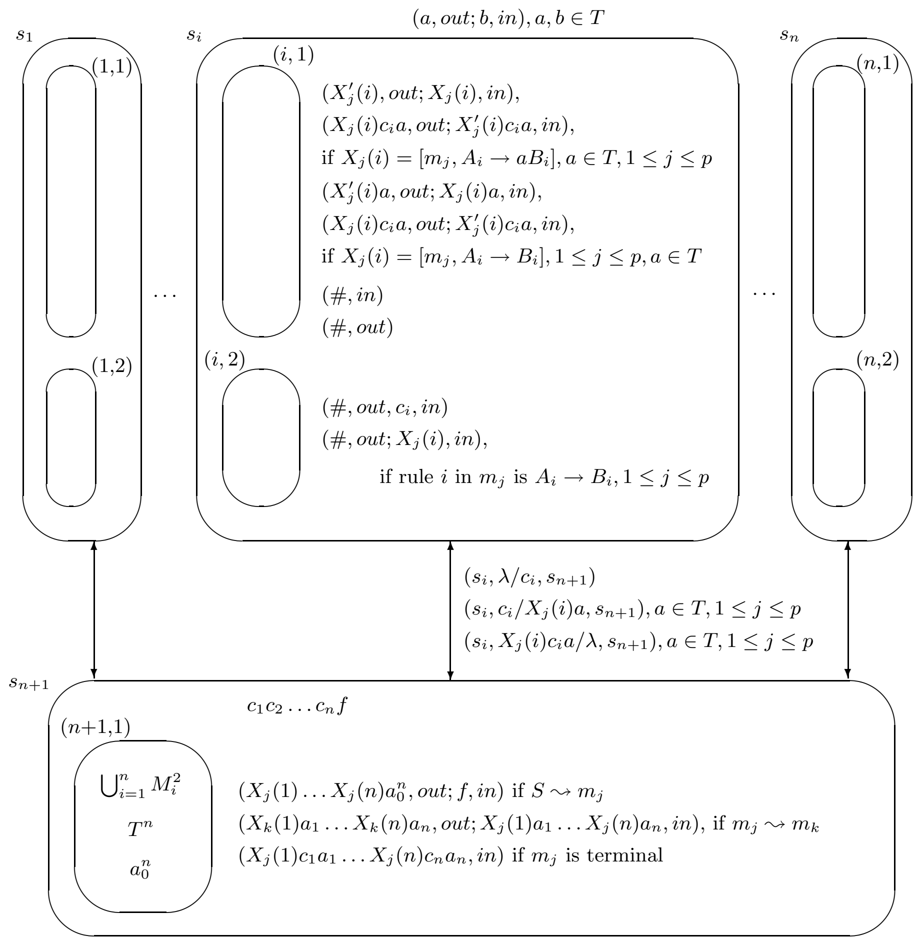
For a set Q, we denote by Q also the multiset consisting of the elements of Q, with the multiplicity one for each of them (hence Q can be considered also as the string composed by the elements of the set, in any ordering).

We are now ready to construct the dP system we look for (a_0 is an arbitrary symbol of T fixed in advance):

$$\begin{aligned}
\Delta &= (O, E, \Pi_1, \dots, \Pi_{n+1}, R), \text{ with :}\\
O &= \bigcup_{i=1}^{n} (M_i \cup M'_i) \cup T \cup \{c_i \mid 1 \leq i \leq n\} \cup \{d, f, \#\},\\
E &= T,\\
\Pi_i &= (O, [\,[\ ]_{(i,1)}[\ ]_{(i,2)}\,]_{s_i}, \lambda, M'_iTc_i, \#, R_{s_i}, R_{(i,1)}, R_{(i,2)}),\\
&\quad R_{s_i} = \{(a, out; b, in) \mid a, b \in T\},\\
&\quad R_{(i,1)} = \{(X'_j(i), out; X_j(i), in),\\
&\quad (X_j(i)c_ia, out; X'_j(i)c_ia, in) \mid 1 \leq j \leq p,\\
&\quad \text{if } X_j(i) = [m_j, A_i \to aB_i], a \in T\}\\
&\quad \cup \{(X'_j(i)a, out; X_j(i)a, in),\\
&\quad (X_j(i)c_ia, out; X'_j(i)c_ia, in) \mid 1 \leq j \leq p, a \in T,\\
&\quad \text{if } X_j(i) = [m_j, A_i \to B_i]\}\\
&\quad \cup \{(\#, in), (\#, out)\},\\
&\quad R_{(i,2)} = \{(\#, out; c_i, in)\}\\
&\quad \cup \{(\#, out; X_j(i), in) \mid 1 \leq j \leq p, \text{ if } X_j(i) = [m_j, A_i \to B_j]\},\\
&\quad \text{for all } 1 \leq i \leq n,\\
\Pi_{n+1} &= (O, [\,[\ ]_{(n+1,1)}\,]_{s_{n+1}}, c_1 \dots c_nf, M_1^2 \dots M_n^2T^na_0^n, \emptyset, R_{(n+1,1)}),\\
&\quad R_{(n+1,1)} = \{(X_j(1) \dots X_j(n)a_0^n, out; f, in) \mid 1 \leq j \leq p \text{ if } S \rightsquigarrow m_j\}\\
&\quad \cup \{(X_k(1)a_1 \dots X_k(n)a_n, out; X_j(1)a_1 \dots X_j(n)a_n, in)\\
&\quad \mid 1 \leq j, k \leq p, a_i \in T, 1 \leq i \leq n, \text{ if } m_j \rightsquigarrow m_k\}
\end{aligned}$$

$$\cup\ \{(X_j(1)c_1a_1 \dots X_j(n)c_na_n, in) \mid a_i \in T, 1 \leq i \leq n, \text{ if } m_j \text{ is a terminal matrix}\},$$
$$R = \{(s_i, \lambda/c_i, s_{n+1}), (s_i, c_i/X_j(i)a, s_{n+1}, (s_i, X_j(i)c_ia/\lambda, s_{n+1}) \mid 1 \leq j \leq p, 1 \leq i \leq n, a \in T\}.$$

This dP system, with one component Π_i and with Π_{n+1} given in full details, is represented in Figure 3.



$(a, out; b, in), a, b \in T$
s_1
$(1,1)$
$(1,2)$
s_i
$(i,1)$
$(X'_j(i), out; X_j(i), in),$
$(X_j(i)c_ia, out; X'_j(i)c_ia, in),$
if $X_j(i) = [m_j, A_i \to aB_i], a \in T, 1 \leq j \leq p$
$(X'_j(i)a, out; X_j(i)a, in),$
$(X_j(i)c_ia, out; X'_j(i)c_ia, in),$
if $X_j(i) = [m_j, A_i \to B_i], 1 \leq j \leq p, a \in T$
$(\#, in)$
$(\#, out)$
$(i,2)$
$(\#, out, c_i, in)$
$(\#, out; X_j(i), in),$
if rule i in m_j is $A_i \to B_i, 1 \leq j \leq p$
s_n
$(n,1)$
$(n,2)$
$(s_i, \lambda/c_i, s_{n+1})$
$(s_i, c_i/X_j(i)a, s_{n+1}), a \in T, 1 \leq j \leq p$
$(s_i, X_j(i)c_ia/\lambda, s_{n+1}), a \in T, 1 \leq j \leq p$
s_{n+1}
$c_1c_2 \dots c_nf$
$(n+1,1)$
$\bigcup_{i=1}^n M_i^2$
T^n
a_0^n
$(X_j(1) \dots X_j(n)a_0^n, out; f, in)$ if $S \rightsquigarrow m_j$
$(X_k(1)a_1 \dots X_k(n)a_n, out; X_j(1)a_1 \dots X_j(n)a_n, in)$, if $m_j \rightsquigarrow m_k$
$(X_j(1)c_1a_1 \dots X_j(n)c_na_n, in)$ if m_j is terminal

Fig. 3. The dP system in the proof of Theorem 3

The components $\Pi_i, 1 \leq i \leq n$, simulate the corresponding "component" of the grammar G, while Π_{n+1} is a "synchronizer" of the other components, it takes no objects from the environment. All rules which bring objects from the environment are uniport rules, hence the system is bounded, the number of objects inside it remains constant during the computation.

We start by sending objects c_i from Π_{n+1} to components Π_i, simultaneously releasing from membrane $(n+1,1)$ some objects $X_j(i), 1 \leq i \leq n$, for a matrix m_j which can follow immediately after an initial matrix of G; each symbol $X_j(i)$ is accompanied by a copy of the symbol a_0, arbitrarily chosen from T.

In the next step, we have to exchange the symbol c_i from Π_i with $X_j(i)a_0$ from Π_{n+1} (if c_i remains unused in Π_i, then it will release the trap object # from membrane $(i,2)$, and the computation will never halt).

In the next step, c_i comes back to Π_i, and in this component we have two possibilities:

(1) The rule i from m_j is of the form $A_i \to aB_i$, and then we use a rule $(a_0, out; b, in)$, for some $b \in T$, and $(X'_j(i), out; X_j(i), in)$.

Now, we check whether the simulation of the rule in G is correct (hence b was the right symbol to take from the environment, i.e., $a = b$): c_i cannot return to Π_{n+1} alone and cannot stay unused in Π_i. The only continuation which does not lead to an infinite computation is to use the rule $(X_j(i)c_ia, out; X'_j(i)c_ia, in)$. These three objects, $X_j(i)c_ia$, can now move together to Π_{n+1}. The only continuation is to move again c_i in Π_i, for all i, and to exchange $X_j(1)\dots X_j(n)$ for some $X_k(1)\dots X_k(n)$ in Π_{n+1}, for $m_j \rightsquigarrow m_k$.

We return in this way to a situation similar to that we have started with: object c_i in Π_i and $X_k(i)$ in Π_{n+1}.

(2) If the rule i from m_j is of the form $A_i \to B_i$, and we use a rule $(a_0, out; b, in)$, for some $b \in T$, then the computation will never stop: we do not have a rule for introducing $X_j(i)$ alone in membrane $(i,1)$, hence $X_j(i)$ must release the trap object from membrane $(i,2)$. Therefore, we have to use the rule $(X'_j(i)a, out; X_j(i)a, in)$ from $R_{(i,1)}$ (at the same time, the object c_i comes to Π_i). As above, the three objects $X_j(i)c_ia$ can move together to Π_{n+1}, where, while c_i moves to Π_i, we exchange $X_j(1)\dots X_j(n)$ for some $X_k(1)\dots X_k(n)$ in Π_{n+1}, for $m_j \rightsquigarrow m_k$.

Also in this case we return to a situation similar to that we have started with: object c_i in Π_i and $X_k(i)$ in Π_{n+1}.

The process can be continued. Checking the correctness of the simulation of the rules in G is done in components Π_i, the fact that the rules which are simultaneously checked form a matrix of G is ensured by the component Π_{n+1}.

When a terminal matrix is simulated, component Π_{n+1} halts the computation by using the rule $(X_j(1)c_1a_1\dots X_j(n)c_na_n, in)$ (if we do not "hide" also the objects c_i in membrane $(n+1,1)$, then these objects have to go to components Π_i, where no rule can use them other than the trap-releasing ones).

We conclude that $L(G) = L(\Delta)$. $\square$

5 Final Remarks

Let us first synthesize all previous results and remarks in a diagram – see Figure 4. The arrows indicate inclusions; if the arrow is marked with a dot, then that inclusion is known to be proper. The inclusions $RSM_n \subset RSM_{n+1}, n \geq 1$, are known to be proper, hence also the hierarchy $LdP^b_n, n \geq 1$, is infinite, but we do not know languages proving the strictness of inclusions $LdP^b_n \subseteq RSM_n \subseteq LdP^b_{n+1}, n \geq 1$, with the exception of the inclusion $RSM_1 \subset LdP^b_2$, because $RSM_1 = REG$ and LdP^b_2 contains non-regular languages (see, e.g., the example in Section 3). Similarly, we do not know whether the inclusions $LdP_n \subseteq LdP_{n+1}, n \geq 2$, are proper – but we *conjecture* that this is the case.

$$\begin{tikzcd}
 & LdP_* \\
RSM_* = LdP^b_* \arrow[ur, "\bullet"{marking, pos=0}] & \\
\cdots \arrow[u] & \cdots \arrow[uu] \\
RSM_3 \arrow[u] & LdP_3 \arrow[u] \\
LdP^b_3 \arrow[u] \arrow[ur, "\bullet"{marking, pos=0}] & \\
RSM_2 \arrow[u] & LdP_2 \arrow[uu] \\
LdP^b_2 \arrow[u] \arrow[ur, "\bullet"{marking, pos=0}] & \\
 & LdP_1 = LP \arrow[uu, "\bullet"{marking, pos=0}] \\
LdP^b_1 = LP^b = REG = RSM_1 \arrow[uu, "\bullet"{marking, pos=0}] \arrow[ur, "\bullet"{marking, pos=0}] &
\end{tikzcd}$$

Fig. 4. The hierarchy of the families RSM_n, LdP^b_n, and LdP_n

Further open problems and research topics about dP systems can be found in the papers mentioned in the bibliography – the study of dP automata is one of the recently introduced and most active branches of membrane computing.

Acknowledgements

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

References

1. E. Csuhaj-Varju, G. Vaszil: About dP automata
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer, Berlin, 1989.
3. R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata. *Annals of Bucharest University. Mathematics-Informatics Series*, 2010 (in press).
4. J. Hromkovic: *Communication Complexity and Parallel Computing: The Application of Communication Complexity in Parallel Computing.* Springer, Berlin, 1997.
5. O. Ibarra: Simple matrix grammars. *Information and Control*, 17 (1970), 359–394.
6. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
7. Gh. Păun, M.J. Pérez-Jiménez: Solving problems in a distributed way in membrane computing: dP systems. *Int. J. of Computers, Communication and Control*, 5, 2 (2010), 238–252.
8. Gh. Păun, M.J. Pérez-Jiménez: P and dP automata: A survey. *Rainbow of Computer Science* (C.S. Calude, G. Rozenberg, A. Salomaa, eds.), LNCS, Springer, Berlin, 2010 (in press).
9. Gh. Păun, M.J. Pérez-Jiménez: An infinite hierarchy of languages defined by dP Systems. *Theoretical Computer Sci.*, in press.
10. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2010.
11. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* 3 volumes, Springer, Berlin, 1998.
12. The P Systems Website: `http://ppage.psystems.eu`.