On Controlled P Systems

Kamala Krithivasan[1], Gheorghe Păun[2,3], Ajeesh Ramanujan[1]

[1]Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Chennai-36, India
kamala@iitm.ac.in, ajeeshramanujan@gmail.com

[2]Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and

[3]Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
gpaun@us.es, ghpaun@gmail.com

Summary. We introduce and briefly investigate P systems with controlled computations. First, P systems with *label restricted* transitions are considered (in each step, all rules used have either the same label, or, possibly, the empty label, λ), then P systems with the computations controlled by languages (as in context-free controlled grammars). The relationships between the families of sets of numbers computed by the various classes of controlled P systems are investigated, also comparing them with length sets of languages in Chomsky and Lindenmayer hierarchies (characterizations of the length sets of ET0L and of recursively enumerable languages are obtained in this framework). A series of open problems and research topics are formulated.

1 Introduction

Most investigations in membrane computing deal with cell-like distributed computing devices (P systems) which process multisets of objects (symbols) in the compartments defined by membranes. That is, the data structure used is the multiset, sets with multiplicities associated with their elements; as a consequence, in a natural way, the results of computations are numbers. However, numerous researches were devoted to computations which have as results strings over given alphabets (in this way, the P systems generate/compute languages). Details and references can be found in [5]. A concise presentation of this research direction, also indicating a series of recent developments and several research topics, is provided by [4].

One of the suggestions in [4] is to associate a control language to a P system, in the way already well-known in formal language theory, e.g., in the case of

context-free controlled grammars (see [2]). The difficulty in the case of P systems is the parallelism of computations: arbitrarily many rules can be used in the same step. There are two ways to overcome this difficulty. The first one, followed in [1], assumes the computations sequential, but here we follow the way suggested in [6] and further explored in [7]: in a step one may use only rules with the same label from a given set of labels, maybe also rules having no label (we say that such a rule has an empty label, denoted by λ). (Note that the sequential mode is not obtained as a particular case, considering the rules labeled in a one-to-one manner and without using the empty label: each single rule $r : a \to u$ should be used as many times as a appears in a multiset.)

Several possibilities appear: to allow rules with empty labels or not; in the latter case, to allow steps when only rules with empty labels are used or not; to have a control language which is finite, regular, or from a subregular family of languages other than the finite ones. Part of these possibilities will be considered here, for non-cooperating P systems and for catalytic P systems.

Some delicate issues appear in comparison with the standard definition of successful computations in P systems (where successful means halting). In the case when no rule is labeled with λ, then in the end of the control word the computation ends, hence we do not need to consider the halting condition. On the contrary, when λ steps are possible, the halting condition should be preserved, as the computation can continue forever by means of λ-steps without interacting with the control word. Moreover, the rules are used in the maximally parallel manner, which means that if no rule can be applied, then the maximally applicable multiset of rules is the empty one; this means that no rule (with the specified label is applied), but still we consider this as a step of the computation. In the case of rules with a nonempty label, one symbol of the control word is "consumed", hence a change in the system configuration (taking into account both the objects and the control word) is obtained, but a λ-step where no rule is applied changes nothing and the computation can continue forever. That is why we impose the restriction that after a λ-step when no rule can be applied, no further λ-step is permitted. This is important in ensuring the halting of computations. Note that a rule of the form $\lambda : a \to a$ can be applied forever to a multiset which contains the object a: nothing is changed, but the rule is effectively applied, this is not a λ-step when no rule is applied.

As expected, by imposing restrictions on the way the rules of a P system are used the computing power is increased. This is confirmed for several cases, both by comparing the power of classes of controlled P systems to each other and to (the sets of numbers associated with) classes of languages in Chomsky and Lindenmayer hierarchies. In this framework, new characterizations of the length sets of ET0L languages and of recursively enumerable languages are obtained. Still, many problems remain open, while further related research topics can be considered (we formulate part of these problems and topics in the last section of the paper).

As we mentioned before, P systems with computations controlled by means of regular languages were also considered in [1], but in a restricted case: the computations were considered sequential and, moreover, the halting condition was replaced with a more powerful condition.

2 Prerequisites

We assume the reader to be familiar with basic notions and results in formal language theory (for details, one can consult [9]) and in membrane computing (see, e.g., [5] and the area website from [10]), that is why we introduce here only the notations we use.

For an alphabet V, we denote by V^* the set of all strings over V, including the empty string, denoted with λ; $V^* - \{\lambda\}$ is denoted by V^+.

A Chomsky grammar is a tuple $G = (N, T, S, P)$, where N is the nonterminal alphabet, V is the terminal alphabet, $S \in N$ is the axiom, and P is the set of rewriting rules. If the rules are of the forms $A \to aB,\ A \to a$, for $A, B \in N, a \in T$, then the grammar is said to be regular. (We omit the rules of the form $A \to \lambda, A \in N$, as they can be removed without changing the generated language, possibly having only a rule $S \to \lambda$ in the case when $\lambda \in L(G)$; however, as usual in formal language theory, in what follows the empty string is ignored when comparing the power of two string processing devices. Correspondingly, number 0 is ignored when comparing the power of two number computing devices.)

We denote by FIN, REG, RE the families of finite, regular, and recursively enumerable languages, respectively. In general, for a family FL of languages, we denote by NFL the family of length sets of languages in FL; formally, $NFL = \{length(L) \mid L \in FL\}$, where $length(L) = \{|x| \mid x \in L\}$ and $|x|$ is the length of the string x. $NREG$ is the family of semilinear sets of numbers (sometimes denoted by $SLIN_1$), and NRE is the family of sets of numbers which can be computed by Turing machines.

A regularly controlled context-free grammar (with appearance checking) is a 6-tuple $G = (N, T, S, P, K, F)$, where $G_0 = (N, T, S, P)$ is a usual context-free grammar (nonterminal alphabet, terminal alphabet, axiom, set of rules), $K \subseteq Lab^*$ is a regular language over an alphabet Lab of labels associated in a one-to-one manner to rules in P (thus, we can imagine that $K \subseteq P^*$, with the rules considered elements of an alphabet) and $F \subseteq Lab$. A derivation in G is a terminal derivation in G_0 which follows a control word $w \in K$, in the appearance checking mode: if a rule $r : A \to u$ is to be used, r not in F, then the rule must be used, otherwise (if A is not present in the sentential form) the derivation is blocked; if $r \in F$ and A appears in the sentential form, then the rule must be used, but if A does not appear in the sentential form, then the rule is skipped and one passes to the next label indicated by the control word w. All terminal words generated in this way form the language $L(G)$. One knows that context-free grammars (using λ-rules) with regular control languages characterize RE (hence, in terms of length set, characterize NRE).

We will also need the notion of an ET0L system (extended tabled interactionless Lindenmayer system). Such a device is a quadruple $\gamma = (V, T, w, P)$, where V is the total alphabet, $T \subseteq V$ is the terminal alphabet, $w \in V^+$ is the axiom, and P is the finite set of tables; a table is a set of rules of the form $a \to u, a \in V, u \in V^*$, which is *complete*, i.e., for each $a \in V$ there is a rule $a \to u$ in the table. The derivation starts from w; in a derivation step $w \Longrightarrow w'$ we use a table in P, and this means rewriting in parallel all symbols from w using the rules in the table. The generated language, $L(\gamma)$, consists of all strings in T^* generated in this way. The families of languages of this form is denoted by $ET0L$. It is known that using or not λ-rules in ET0L systems makes no difference in the generative power, the same family $ET0L$ is obtained, and that $ET0L \subset RE$ and $NET0L \subset NRE$. If the terminal alphabet T is not present, we have a (non-extended) tabled interactionless Lindenmayer system, in short, a T0L system; then all strings generated are accepted in the language $L(\gamma)$ (hence each step of a derivation produces a string). It is known that $T0L \subset ET0L$ and $NREG \subset NT0L \subseteq NET0L$ (see [8]).

In what concerns the classes of P systems we consider in this paper, they are the *cell-like transition P systems* (in short, P systems), specifically, with *non-cooperating* and with *catalytic* rules. Such a system is a tuple $\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$ where O is the alphabet of objects, $C \subseteq O$ is the set of catalysts (this component is present only in the catalytic systems and it is omitted in non-cooperating P systems), μ is the membrane structure, with m membranes, w_i is the multiset of objects present in region i of μ in the initial configuration, and R_i is the set of rules present in region i of μ; these rules are of the forms $a \to u$ and $ca \to cu$, where $a \in O, c \in C, u \in (\{b_{here}, b_{out} \mid b \in O\} \cup \{b_{in_j} \mid b \in O, 1 \leq j \leq m\})^*$; the target indication *here* is omitted. If $C = \emptyset$, hence all rules are of the form $a \to u$, then Π is called *non-cooperating*. The computation proceeds in a maximally parallel way and it provides an output only if it halts, a configuration is reached where no rule can be applied. The result of a computation is the number of objects which are sent out of the system during the computation (objects b which appear in the form b_{out} in the right hand side of rules used in the skin region are sent out of the system, into the environment). The number m of membranes in μ is called the *degree* of the system.

The set of numbers generated by a P system Π is denoted by $N(\Pi)$. The family of sets $N(\Pi)$ generated by P systems of degree at most m is denoted by $NP_m(ncoo)$ when using non-cooperating rules and $NP_m(cat_i)$ when using catalytic rules, with at most i catalysts present in the system. If the number of membranes is not bounded, then the subscript m is replaced by $*$. The following results are known: $NP_*(ncoo) = NREG$, $NP_1(cat_2) = NRE$, but the size of the family $NP_*(cat_1)$ is not known (it is believed that it is strictly included in NRE). As only the case of one catalyst is of interest, in what follows we will investigate only this case.

3 Label Restricted P Systems

Consider a catalytic P system $\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$ (of course, if $C = \emptyset$, then we have a non-cooperating system) and associate with each rule in sets $R_1, \ldots, R_m$ a label, which can be either a symbol from an alphabet H or it can be λ; thus, the rules are written in the form $r : u \to v$, with $r \in H$, or $\lambda : u \to v$. We add then the alphabet of labels to the system, in the form $\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H)$ and we say that Π is *labeled*. (Note that this time the labeling is not necessarily one-to-one like in controlled context-free grammars.)

A computation in a labeled P system Π is *label restricted* if in each step one uses (in the maximally parallel manner) only rules with the empty label and rules labeled with the same label in H. A step where only rules $\lambda : u \to v$ are used is called a *λ-step*.

The computations proceed exactly as in a usual P system: we start from the initial configuration, we proceed through maximally parallel steps (which are label restricted), and we get a result (in the environment) after the computation halts. Only halting computations provide a result.

Two cases can be distinguished: using only labels in H (indicated by lr) or also allowing empty labels (indicated by lr_λ). Correspondingly, we obtain four families of sets of numbers: $NP_*(ncoo, lr), NP_*(cat_1, lr)$, and $NP_*(ncoo, lr_\lambda), NP_*(cat_1, lr_\lambda)$, respectively. (Of course, when the number of membranes is bounded, the subscript of NP specifies the bound.)

Note the important detail that lr_λ indicates that rules $\lambda : u \to v$ are allowed and, moreover, λ-steps are allowed. A possible case of interest would be to allow rules with the empty label, but not λ-steps (i.e., to ask that in each step at least a rule with a non-empty label to be used); this case remains as a research topic.

We will mention now, in the form of lemmas, a series of relations about families defined up to now, and later we will synthesize all of them (as well as some results from the literature) in a diagram theorem.

Lemma 1. *$NP_*(\alpha) = NP_1(\alpha)$ and $NP_*(\alpha, \beta) = NP_1(\alpha, \beta)$, $\alpha \in \{ncoo, cat_1\}, \beta \in \{lr, lr_\lambda\}$.*

Proof. The first equality is known, the second one can be proved in the same way: objects in a membrane i are indexed with i, and then all membranes different from the skin membrane can be omitted, the rules are handling indexed objects in the same way as in the compartments of the initial membrane structure. The target indications in the right hand side of rules are easily implemented by changing the subscripts of objects.

According to this lemma, from now on we will use only P systems with only one membrane, and the subscript $*$ in the notation of the generated families is omitted.

Lemma 2. *$NP(ncoo, \alpha) \subseteq NP(cat_1, \alpha), \alpha \in \{lr, lr_\lambda\}$.*

Proof. Directly from the definition.

Lemma 3. $NP(\alpha, lr) \subseteq NP(\alpha, lr_\lambda)$, $\alpha \in \{ncoo, cat_1\}$.

Proof. Directly from the definition.

Lemma 4. $NP(\alpha) \subseteq NP(\alpha, lr)$, $\alpha \in \{ncoo, cat_1\}$.

Proof. Consider a system $\Pi = (O, C, \mu, w_1, R_1)$ and add to it the set $H = \{r\}$, with the same label r associated with all rules in R_1. Denote by Π' the obtained labeled P system. Clearly, no restriction is imposed on using the rules of Π', hence $N(\Pi) = N(\Pi')$.

Lemma 5. $NET0L \subseteq NP(ncoo, lr)$.

Proof. Let $\gamma = (V, T, w, P)$ be an ET0L system with n tables. Label all rules in table i with $r_i, 1 \leq i \leq n$, and let P_1 be the union of all these tables. We construct the labeled P system

$$\Pi = (O, [\quad]_1, w, R_1, H),$$

where

$$O = V \cup \{\#\},$$
$$R_1 = P_1 \cup \{f : A \to \# \mid A \in V - T\} \cup \{f : \# \to \#\} \cup \{f : a \to (a, out) \mid a \in T\},$$
$$H = \{r_i \mid 1 \leq i \leq n\} \cup \{f\}.$$

In each step of a computation in Π we use either only rules from a table of γ or rules with the label f. If these latter rules are used before completing a terminal derivation in γ, then the trap object $\#$ is introduced, and the computation never halts. In the end of the computation, if the derivation in γ is not terminal, the rules with the label f must be used – in this way we check whether the derivation in γ was terminal; simultaneously, all terminal symbols of γ are sent to the environment, hence the computation halts. Thus, we have, $length(L(\gamma)) = N(\Pi)$.

Somewhat surprisingly, we also have the following result.

Lemma 6. $NRE = NP(cat_1, lr)$.

Proof. We only have to prove the inclusion $\subseteq$, the opposite one is a consequence of the Turing-Church thesis (it can also be proved by a direct, straightforward but cumbersome construction of a Turing machine simulating a label restricted P system).

Let us consider a regularly controlled context-free grammar $G = (N, T, S, P, K, F)$, with $K \subseteq Lab^*$, where Lab is an alphabet of labels associated in a one-to-one manner with rules in P. Consider a regular grammar $G_K = (N_K, Lab, S_K, P_K)$ generating the language K; assume the rules in P_K to be labeled in a one-to-one manner with symbols in a set Lab_K. We construct the labeled catalytic P system

$$\Pi = (O, \{c\}, [\quad]_1, cS_K SE, R_1, H),$$

where:

$$O = N \cup T \cup N_K \cup \{c, E, t, \#\},$$
$$H = \{(s, r) \mid s \in Lab_K, r \in Lab\} \cup \{f\},$$

and the set R_1 is constructed as follows:

1. For $s : X_K \to rY_K \in P_K$ and $r : A \to u \in P$ such that $r \notin F$, we introduce in R_1 the following rules:

   $(s, r) : X_K \to Y_K$,
   $(s, r) : Z_K \to \#,\ Z_K \in N_K, Z_K \neq X_K$,
   $(s, r) : cA \to cu_{here}u_{out},\ \ u_{here}$ contains all nonterminal symbols of u and u_{out} contains all terminal symbols of u, with the subscript *out*,
   $(s, r) : cE \to c\#$,
   $(s, r) : t \to \#$;

2. For $s : X_K \to rY_K \in P_K$ and $r : A \to u \in P$ such that $r \in F$, we introduce in R_1 the following rules:

   $(s, r) : X_K \to Y_K$,
   $(s, r) : Z_K \to \#,\ Z_K \in N_K, Z_K \neq X_K$,
   $(s, r) : cA \to cu_{here}u_{out},\ \ u_{here}$ contains all nonterminal symbols of u and u_{out} contains all terminal symbols of u, with the subscript *out*,
   $(s, r) : t \to \#$;

3. For $s : X_K \to r \in P_K$ and $r : A \to u \in P$ such that $r \notin F$, we introduce in R_1 the following rules:

   $(s, r) : X_K \to t$,
   $(s, r) : Z_K \to \#,\ Z_K \in N_K, Z_K \neq X_K$,
   $(s, r) : cA \to cu_{here}u_{out},\ \ u_{here}$ contains all nonterminal symbols of u and u_{out} contains all terminal symbols of u, with the subscript *out*,
   $(s, r) : cE \to c\#$,
   $(s, r) : t \to \#$;

4. For $s : X_K \to r \in P_K$ and $r : A \to u \in P$ such that $r \in F$, we introduce in R_1 the following rules:

$(s,r) : X_K \to t$,
$(s,r) : Z_K \to \#,\ Z_K \in N_K, Z_K \neq X_K$,
$(s,r) : cA \to cu_{here}u_{out}$, u_{here} contains all nonterminal symbols of u and u_{out} contains all terminal symbols of u, with the subscript *out*,
$(s,r) : t \to \#$;

5. We also consider the following rules:

$$f : E \to \lambda,$$
$$f : t \to \lambda,$$
$$f : Z_K \to \#,\ Z_K \in N_K,$$
$$f : A \to \#, A \in N,$$
$$f : \# \to \#.$$

We start by introducing both the axiom S of G and the axiom S_K of G_K in the initial configuration of Π, together with the catalyst c and the auxiliary object E. The catalyst has the role of ensuring that the rules of P are simulated in a sequential way, not in a parallel one.

The objects X_K ensure the fact that the computation in Π follows the same sequence of rules in P as requested by a control word from K. Together with simulating a derivation step in G_K (performed by a rule $s : X_K \to rY_K$ or $s : X_K \to r$) we also simulate the rule $r : A \to u$ from P, and this is done by the rules of Π with the label (s,r).

Consider a rule $(s,r) : cA \to cu_{here}u_{out}$ in Π, corresponding to the rule $r : A \to u$ in P. If the object A is present and we use the rule $(s,r) : cA \to cu_{here}u_{out}$, this corresponds to a derivation step in G. If the object A is present and, instead of $(s,r) : cA \to cu_{here}u_{out}$ we use the rule $(s,r) : cE \to c\#$, then no result is obtained, the computation never halts. If the object A is not present and $r : A \to u$ is a rule from F (remember that the rules of P are labeled in a one-to-one manner with symbols in Lab), then the rule cannot be applied, hence nothing is changed (the rule $(s,r) : cE \to c\#$ is not present in this case). If the object A is not present and the rule r is not from F, then the trap-object $\#$ is introduced by means of the rule $(s,r) : cE \to c\#$. This object will evolve forever by means of the rule $f : \# \to \#$, hence the computation will never halt. In any moment, in between steps which use rules (s,r), also rules $f : A \to \#, A \in N$, can be used, but this will lead to a non-halting computation.

After ending the simulation of a computation in G, we can use an f-rule – and this must be done, as otherwise the computation is not completed (not halted). In this way, we check whether the derivation in G was a terminal one; if not, a rule $f : A \to \#$ can be effectively used and the computation in Π will never halt.

If the derivation in G_K is not correctly simulated, then again the trap object $\#$ is introduced; this is ensured by the rules of the form $(s,r) : Z_K \to \#$ introduced for each s and r.

In the end of the derivation, if the symbol # was introduced, then the rule $f : \# \to \#$ can be used forever, hence the computation in Π will never stop.

If the derivation in G_K is terminal, hence the object t is introduced, but the derivation in G is not terminal, then for each rule $(s, r) : cA \to z$ which is used from now on we have to also use $(s, r) : t \to \#$ and the computation never halts. Conversely, if the derivation in G ends first, then rules $(s, r)X_K \to z$ can be used only if $r \in F$, otherwise also the rule $(s, r) : cE \to c\#$ must be used, but this does not change the multiset generated by G. Thus, the derivations in G_K and G end in the right way.

Therefore, only (and all) terminal derivations in G which follow control words in the language K can be simulated by halting computations in Π, that is, $N(\Pi) = length(L(G))$.

4 Controlled P Systems

In the previous label restricted computations in a labeled P system all sequences of labels are allowed, which corresponds to using H^* as a control language. The control language can be a particular one, and this leads to *controlled P systems*.

Such a system is of the form $\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H, K)$ where $(O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, H)$ is a labeled P system and $K \subseteq H^*$ is a language in a given family FL. A computation in Π has to follow a control word in K. If the empty label is not used, then the computation stops in the moment when the control word ends, irrespective of the fact whether there are rules which can be applied to the reached configuration. Of course, if rules with the empty label (and hence λ-steps) are allowed, then arbitrarily many λ-steps can be performed in between steps where rules with the labels indicated by the word in K are used. This makes necessary the returning to the halting condition in the case lr_λ: after the last step where rules with a label in H is used, we have no control about the end of the computation, arbitrarily many λ-steps can be done. That is why we again impose the restriction that after a λ-step where no rule can be applied we cannot continue with another λ-step (note that such a restriction is not imposed for steps which correspond to non-empty labels: after a step when rules with a label in H should be used, even if no rule is applicable, we pass to the next label in the control word and again it may happen that no rule is applicable, but the control word is "consumed" symbol by symbol).

The computation is done in the maximally parallel manner, in the label restriction framework. This means that when rules with a label r have to applied, a maximal multiset of applicable rules with this label is used. In particular, such a maximal multiset may be empty: no rule with label r can be applied. This is a powerful feature, as it works in a way similar to the appearance checking feature in regulated rewriting (in particular, in controlled context-free grammars).

We denote by $NCP_m(\alpha, \beta, FL)$ the family of sets $N(\Pi)$ generated by controlled P systems Π of degree at most m, with rules of types $\alpha \in \{ncoo, cat_1\}$, with the

rules labeled in the sense of $\beta \in \{lr, lr_\lambda\}$, and with the control language in the family FL. As usual, m is replaced with $*$ when no bound on the number of membranes is considered.

Lemma 7. *$NCP_*(\alpha, \beta, FL) = NCP_1(\alpha, \beta, FL)$ for all $\alpha \in \{ncoo, cat_1\}$, $\beta \in \{lr, lr_\lambda\}$, and any family FL.*

Proof. The same ideas as in the proof of Lemma 1 can be used.

Thus, from now on only P systems with only one membrane will be considered and no subscript is used in the notations of the generated families of sets of numbers.

Lemma 8. *$NCP(ncoo, \alpha, FL) \subseteq NCP(cat_1, \alpha, FL)$ for $\alpha \in \{lr, lr_\lambda\}$, $NCP(\alpha, lr, FL) \subseteq NCP(\alpha, lr_\lambda, FL)$, for $\alpha \in \{ncoo, cat_1\}$ and all FL, and $NCP(\alpha, \beta, FL) \subseteq NCP(\alpha, \beta, FL')$ for $\alpha \in \{ncoo, cat_1\}, \beta \in \{lr, lr_\lambda\}$, for all families $FL \subseteq FL'$.*

Proof. Directly from the definitions.

Lemma 9. *$NFL \subseteq NCP(ncoo, lr, FL)$ for all families FL.*

Proof. Let $L \subseteq V^*$ be a language in a family FL. We construct the controlled P system

$$\Pi_L = (\{b\} \cup V, [\ \ ]_1, b, \{a : b \to ba_{out} \mid a \in V\}, V, L).$$

Taking a string $w \in L$ as a control word, the rules of Π_L produce symbol by symbol the string w and halts when the string ends (no λ-step is possible). Therefore, $N(\Pi_L) = length(L)$.

In what follows, we consider only control languages in the families FIN and REG.

Lemma 10. *$NCP(\alpha, lr, FIN) = NFIN, \alpha \in \{ncoo, cat_1\}$.*

Proof. All computations in a controlled P system where no rule has the empty label and the control language is finite are finite, hence the inclusion $\subseteq$ follows. The opposite inclusion follows from Lemma 9.

Lemma 11. *$NCP(ncoo, lr_\lambda, FIN)$ contains non-semilinear sets.*

Proof. Consider the system

$$\Pi = (\{a\}, [\ \ ]_1, a, \{\lambda : a \to aa,\ r : a \to a_{out}\}, \{r\}, \{r\}).$$

After $n \geq 0$ λ-steps (during this time we produce 2^n copies of object a), if we use the rule $r : a \to a_{out}$ (as requested by the control language), then all copies of a should be sent out (hence the rule $\lambda : a \to aa$ should not be used at this step), otherwise the computation will never halt (the rule $r : a \to a_{out}$ cannot be used for a second time). Therefore, $N(\Pi) = \{2^n \mid n \geq 0\}$, which is not a semilinear set.

Lemma 12. *$NCP(ncoo, lr, REG)$ contains non-semilinear sets.*

Proof. Consider the system

$$\Pi = (\{a\}, [\ \ ]_1, a, \{r : a \to aa,\ r' : a \to a_{out}\}, \{r, r'\}, \{r^* r'\}).$$

After $n \geq 0$ steps when we produce 2^n copies of object a by means of rule r, we have to also use rule r', which sends all objects to the environment, hence the computation halts. Therefore, $N(\Pi) = \{2^n \mid n \geq 0\}$.

Lemma 13. *$NP(\alpha) \subseteq NCP(\alpha, lr_\lambda, FIN)$, $\alpha \in \{ncoo, cat_1\}$.*

Proof. Take a P system $\Pi = (O, C, [\ \ ]_1, w_1, R_1)$ and construct the controlled system

$$\Pi' = (\{a\} \cup O, C, [\ \ ]_1, a, \{r : a \to w_1\} \cup \{\lambda : u \to v \mid u \to v \in R_1\}, \{r\}, \{r\}).$$

After one initial step, when producing the initial multiset of Π, the system Π' continues exactly as in Π, with arbitrarily many λ-steps. The computation in Π' halts if and only if the corresponding computation in Π halts, hence we get $N(\Pi) = N(\Pi')$.

Lemma 14. *$NCP(ncoo, lr_\lambda, REG) \subseteq NET0L$.*

Proof. Consider a controlled P system $\Pi = (O, [\ \ ]_1, w_1, R_1, H, K)$ with $K \in REG$. Let $G = (N, H, S, P)$ be a regular grammar for the language K. We construct the following ET0L system:

$$\gamma = (V, \{b\}, Sw_1, U), \text{ where}$$
$$V = N \cup H \cup O \cup \{r_t \mid X \to r \in P, X \in N, r \in H\} \cup \{b, \#\},$$

and U contains the following tables (in each case, so-called completion rules are added, i.e., rules of the form $a \to a$ for all symbols $a \in V$ for which no rule was already specified in the table; we do not explicitly specify these completion rules; remember that in each rule $a \to u$ of Π, the string u is composed of symbols $a \in O$ and $a_{out}, a \in O$; in the rules of the tables below, $b(u)$ denotes the string obtained by replacing each a_{out} from $u \in (O \cup \{a_{out} \mid a \in O\})^*$ by b):

1. $\{X \to rY\}$
   $\cup \{Z \to \# \mid Z \in N, Z \neq X\}$
   $\cup \{r_t \to \#\}$
   $\cup \{a \to b(u) \mid r : a \to u \in R_1\}$
   $\cup \{a \to b(u) \mid \lambda : a \to u \in R_1\}$,
   for each rule $X \to rY \in P,\ X, Y \in N, r \in H$;
2. $\{X \to r_t\}$
   $\cup \{Z \to \# \mid Z \in N, Z \neq X\}$
   $\cup \{r_t \to \#\}$
   $\cup \{a \to b(u) \mid r : a \to u \in R_1\}$
   $\cup \{a \to b(u) \mid \lambda : a \to u \in R_1\}$,
   for each rule $X \to r \in P, X \in N, r \in H$;

3. $\{X \to X \mid X \in N\} \cup \{r_t \to r_t\}$
   $\cup \{a \to b(u) \mid \lambda : a \to u \in R_1\}$;
4. $\{r_t \to \lambda\}$
   $\cup \{Z \to \# \mid Z \in N\}$
   $\cup \{a \to \# \mid$ there is a rule $\lambda : a \to u \in R_1\}$
   $\cup \{a \to \lambda \mid$ there is no rule $\lambda : a \to u \in R_1\}$
   $\cup \{r \to \lambda \mid r \in H\}$.

In each derivation step of γ one both generates a label $r \in H$, according to the rules of the regular grammar G, and one simulates all rules with that label or rules with the empty label (tables of type 1). λ-steps can be intercalated in-between steps which use at least one rule with a label in H (tables of type 3). In the end of the derivation in G one introduces the symbol r_t and one simulates rules with the label r and rules with label λ (tables of type 2).

After introducing a label r_t, one can continue by simulating arbitrarily many λ-steps (tables of type 3). The string can become a terminal one only by using the table of type 4. This table checks whether the control word is completed, that is, the derivation in G is terminal (if this is not the case, then a rule $Z \to \#, Z \in N$, has to be used), whether the computation in Π halted (if not, then rules $a \to \#$ can be used), erases all symbols $a \in O$ for which there is no rule $\lambda : a \to u \in R_1$, all $r \in H$, as well as r_t for $r \in H$. Note that all objects a_{out} were replaced by b, which is the terminal symbol of γ.

The tables cannot be used in the wrong moment, because of the rules of the form $Z \to \#$.

Consequently, $N(\Pi) = length(L(\gamma))$, and this completes the proof.

5 A Synthesis Theorem

Putting together all previous lemmas as well as known relations between families in Chomsky and Lindenmayer hierarchies, we obtain the following result.

Theorem 1. *The relations from Figure 1 hold. The arrows indicate inclusions while the dotted arrows correspond to strict inclusions.*

We have not mentioned in this diagram the position of the family $NT0L$. For it we have the following relations: $NREG \subset NT0L \subseteq NCP(ncoo, lr, REG) \subseteq NET0L$. The inclusion $NT0L \subseteq NCP(ncoo, lr, REG)$ can be proved in a way similar to the proof of Lemma 5.

Lemma 15. $NT0L \subseteq NCP(ncoo, lr, REG)$.

Proof. Let $\gamma = (V, w, P)$ be a T0L system with n tables. Label all rules in table i with $r_i, 1 \leq i \leq n$, let H be the set of all these labels, and let P_1 be the union of all the tables (with labeled rules). Consider the labeled P system

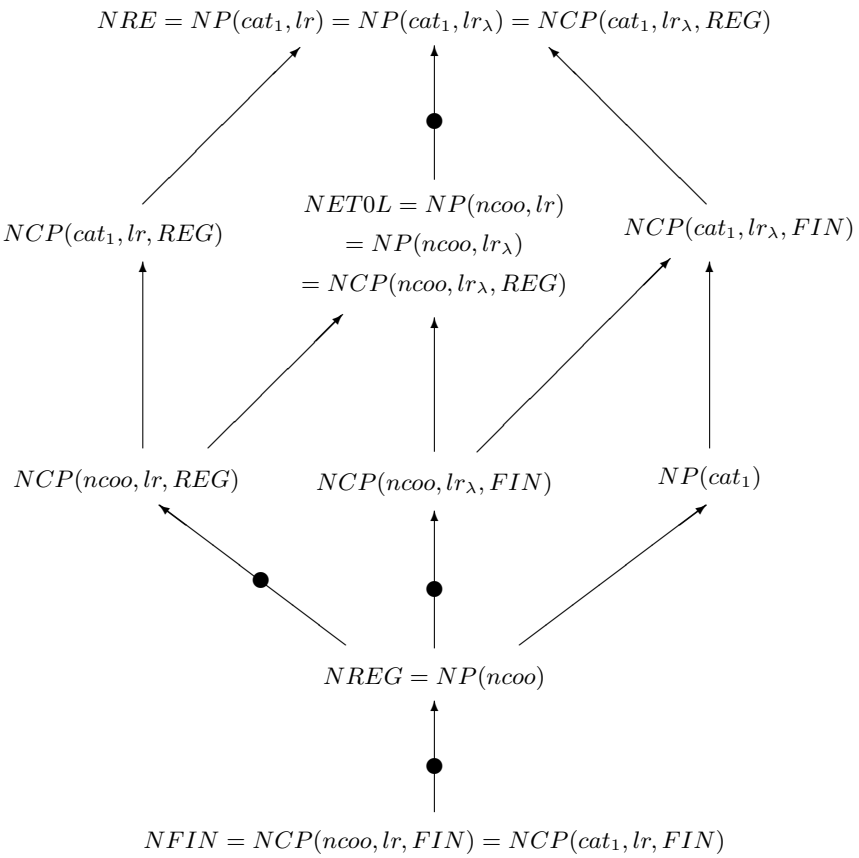

Fig. 1. The hierarchy of families of sets of numbers generated by controlled transition P systems

$$\Pi = (V, [\ \ ]_1, w, R_1, H \cup \{f\}, H^*\{f\}),$$
$$R_1 = P_1 \cup \{f : a \to (a, out) \mid a \in V\}.$$

In each step of a computation in Π we use only rules from a table of γ, chosen according to a control word in H^*. When the control word ends, we sent all objects to the environment, and the computation stops. As H^* contains all possible sequences of tables, we have $length(L(\gamma)) = N(\Pi)$.

Note that a similar language H^* cannot be used in the ET0L case, because we have to check whether the computation in Π ends when the ET0L system has generated a terminal string.

6 Further Research Topics

First of all, it is an open problem whether or not the inclusions in Figure 1 which were not shown to be proper are strict and whether the families not linked by a path in this diagram are comparable. Note the difference between families $NCP(\alpha, lr, FL)$ and $NP(\alpha, lr_\lambda)$: the halting is different in the two cases and this makes difficult their comparison (when we do not have λ-steps and we use a control word, the computation halts when the control word ends, without checking whether further rules could be applied to the obtained configuration).

Then, besides the lr and lr_λ cases considered here, further possibilities seem to be of interest. First, we may impose that in each step when a nonempty label r is indicated by the control word, at least one rule with this label is used (hence we do not allow steps where the maximal multiset of applicable rules is empty – or it contains only rules with the empty label). Also, in the lr_λ case, we may not allow λ-steps after ending the control word.

Besides considering sets of numbers computed by P systems, we can also consider vectors of numbers, counting the multiplicity of different objects in the output (this corresponds to Parikh sets of languages). Whether or not results different from those in Figure 1 are obtained for vectors remains to be checked. (The question is not trivial, in view of the fact that it is an open problem which of the next inclusions is proper: $NREG \subseteq NP(cat_1) \subseteq NRE$). Another direction of research is to consider other classes of P systems instead of transition P systems. For symport/antiport systems we cannot obtain too much, as these systems are universal even with severe restrictions on the complexity of the systems in terms of the number of membranes and the size of rules. Still, the idea of label restricted computations is useful in the case of small symport/antiport systems: the following example was given in [4]:

$$\Pi = (\{a, b\}, [\ \ ]_1, a, \{a, b\}, \{r_1 : (a, out; aa, in),\ r_2 : (a, out; b, in)\}, 1).$$

After using for some $n \geq 0$ steps rule r_1 (2^n copies of a are obtained in the system), the second rule should be used – otherwise the computation cannot stop. The output is obtained in membrane 1 and we have $N(\Pi) = \{2^n \mid n \geq 0\}$, which is not semilinear. The same result is obtained if the control language $r_1^* r_2$ is added to the system Π.

Similarly, one has to consider the case of spiking neural P systems – the class of P systems where the idea of control words (and hence of label restricted computations) was introduced, [6].

Acknowledgements. The work of Gh. Păun has been supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200, co-financed by FEDER funds. Useful discussions with Rudi Freund during 11th Brainstorming Week on Membrane Computing, Sevilla, 4-8 February 2013, are gratefully acknowledged.

References

1. A. Alhazov, R. Freund, H. Heikenwälder, M. Oswald, Y. Rogozhin, S. Verlan: Sequential P systems with regular control. *Membrane Computing International Conference. CMC 2012, Budapest, Hungary. Invited and Selected Papers* (E.Csuhaj-Varju, M. Gheorghe, G. Vaszyl, eds.), Springer, LNCS 7762, 2013, in press.
2. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer, Berlin, 1989.
3. Gh. Păun: Computing with membranes. *J. Comput. Syst. Sci.*, 61 (2000), 108–143 (see also TUCS Report 208, November 1998, `www.tucs.fi`).
4. Gh. Păun, M.J. Pérez-Jiménez: Languages and P systems: Recent developments, *Computer Sci. J. of Moldova*, 20, 2 (59), 2012, 112–132.
5. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *The Oxford Handbook of Membrane Computing*. Oxford University Press, 2010.
6. A. Ramanujan, K. Krithivasan: Control words of spiking neural P systems. *Romanian J. of Information Science and Technology*, to appear.
7. A. Ramanujan, K. Krithivasan: Control words of transition P systems. Paper in preparation, 2012.
8. G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
9. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. 3 volumes, Springer, Berlin, 1998.
10. The P Systems Website: `http://ppage.psystems.eu`.