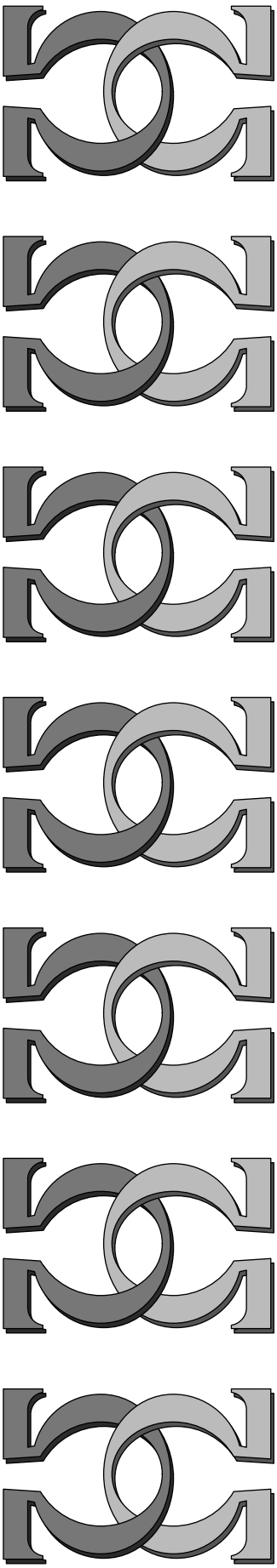

CDMTCS Research Report Series

Computing with Membranes: P Systems with Worm-Objects

Juan Castellanos
Department of Artificial Intelligence,
Faculty of Computer Science, Politechnical
University of Madrid

Gheorghe Păun
Institute of Mathematics of the Romanian
Academy, Bucureşti, Romania

Alfonso Rodríguez-Paton
Department of Artificial Intelligence,
Faculty of Computer Science, Politechnical
University of Madrid

CDMTCS-123
February 2000

Centre for Discrete Mathematics and
Theoretical Computer Science

Computing with Membranes: P Systems with Worm-Objects[1]

Juan CASTELLANOS[a], Gheorghe PĂUN[b], Alfonso RODRÍGUEZ-PATON[a]

[a]Department of Artificial Intelligence, Faculty of Computer Science
Politechnical University of Madrid
Campus de Montegancedo, Boadilla del Monte 28660, Madrid, Spain
E-mails: jcastellanos/arpaton@fi.upm.es

[b]Institute of Mathematics of the Romanian Academy
PO Box 1 – 764, 70700 Bucureşti, Romania
E-mail: gpaun@imar.ro

Abstract. We consider a combination of P systems with objects described by symbols with P systems with objects described by strings. Namely, we work with multisets of strings and consider as the result of a computation the number of strings in a given output membrane. The strings (also called *worms*) are processed by *replication, splitting, mutation*, and *recombination*; no priority among rules and no other ingredient is used. In these circumstances, it is proved that (1) P systems of this type can generate all recursively enumerable sets of numbers, and, moreover, (2) the Hamiltonian Path Problem in a directed graph can be solved in a quadratic time, while the SAT problem can be solved in a linear time.

1 Introduction

P systems are a class of distributed parallel computing models inspired from the way the alive cells process chemical compounds, energy, and information. In short, in the *regions* delimited by a *membrane structure* (see Figure 1 for an illustration of what this means), are placed *multisets* of *objects*, which evolve according to *evolution rules* associated with the regions; the objects can also be *communicated* from a region to another one, according to certain *target indications*, while the membranes can be *dissolved* (and then the objects of the dissolved membrane remain free in the region immediately outside it) and *divided* (then the contents of the divided membrane is copied in each of the resulting membranes); the rules are applied in the maximally parallel manner (in each time unit, all objects which can evolve should evolve); a *computation* consists of transitions among system *configurations*, while a *complete computation* is a halting one; the *result* of a complete computation is either the number of objects present in the halting configuration in a specified *output membrane*, or it is read outside the system, as the sequence of objects

[1]Research supported by the Direcció General de Recerca, Generalitat de Catalunya (PIV), and the Politechnical University of Madrid

leaving the system during the computation. The application of rules can be regulated by a priority relation among them, while the communication can be controlled in various ways. We do not enter here into details; the reader is referred to [10], [11], and, mainly, to Chapter 3 of [2], which contains a first synthesis of the domain.

P systems as above are also called *transition P systems*. They use objects identified by symbols from a given alphabet.

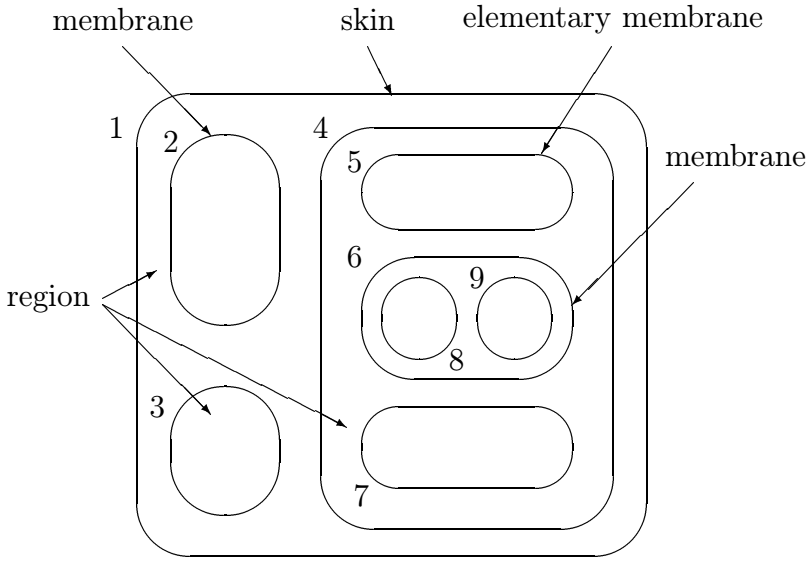

Figure 1: A membrane structure

It is also possible to consider objects described by strings. This was already done in [10]. Important differences from the transition P systems appear. First, we have to consider string processing operations; in the case of rewriting we get *rewriting P systems*, when using splicing, in the sense of [5] (see [15] for a comprehensive investigation of this operation), we obtain *splicing P systems*, [10], [16]. Moreover, we do not work with multisets, but with usual languages. Also the result of a computation is a language.

Both transition and rewriting/splicing P systems are computationaly complete (equal in power to Turing machines), for various variants. In the case of symbol-objects, this means that each set of natural numbers which can be computed by a Turing machine can be generated by a P system, in the case of string-objects this means that each recursively enumerable language can be generated by a P system.

It is of interest to note that in the case when membrane division is considered, NP-complete problems can be solved in linear time; this is the case with SAT (see [13]), the Hamiltonian Path Problem and the Node Covering Problem (see [6]); also DES can be broken in linear time by such systems (see [7]). In all cases, one needs exponentially many membranes; here we try to avoid this, of course, without being possible not to use an exponential resource (here, strings in the membranes of the system).

We combine here the two classes of P systems: we work with multisets of string-objects and we consider the result of a computation as the number of strings in a given

output membrane. Because we need to increase and decrease the number of strings in the regions of the system, rewriting and splicing rules are not useful (they can at most decrease the number of strings, by sending them outside the system, but cannot increase this number). We take a suggestion from [20] and consider *replication, splitting, mutation*, and *merging* (here, *recombination*) operations on strings. The precise meaning of these operations will be defined in the next section. The formal details are significantly different from those in [20], where one works with strings – called *worms*, which also suggested the terminology we use here – composed of binary symbols and which move on a grid, in a cellular automata framework. For instance, recombination means here crossing-over at a common block, in the sense of [19] and the simple splicing systems [9]: x_1ux_2, y_1uy_2 lead to x_1uy_2, y_1ux_2.

Two features of our systems seem to be very powerful: the possibility to exponentially increase the number of strings, by replication and splitting operations, and the context-sensitivity brought by the recombination operation. This operation is of a very restricted type, for instance, in comparison with the general splicing operation, but the fact that the crossing-over block (the string u in the previous example) can be of a length greater than one can be used in a surprisingly efficient manner.

Making use of these features – but no other control on rules application, such as a priority relation (or the membrane thickness control, like in [12]) – we get two (already expected in P system area, but important) results: computational completeness and polynomial solutions to NP-complete problems. The latter result does not use membrane division, but the exponential space is provided by the replication and splitting operations; the number of membranes is fixed, but their contents can increase exponentially.

Our proof of the computational completeness does not provide a bound on the number of membranes; this remains as an *open problem*. Some other open problems are formulated, too.

2 Handling (the Number of) Worms

We start by a preliminary discussion about the string operations we will use in order to increase and decrease the number of strings in the systems we shall define in the next section. The reader can find many analogies with operations on DNA molecules, but we do not enter here such details.

For formal language theory prerequisites we refer to [18]. We only mention that we denote by V^* the free monoid generated by the alphabet V under the operation of concatenation; the empty string is denoted by λ, $V^+ = V^* - \{\lambda\}$ is the set of non-empty strings over V, and $|x|$ is the length of $x \in V^*$.

Given an alphabet V, we consider the following *operations* on strings over V:

1. *Replication.* If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \to u_1||u_2$ is called a *replication rule*. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Longrightarrow_r (w_2, w_3)$ (and we say that w_1 is replicated with respect to rule r) if $w_1 = x_1ax_2,\ w_2 = x_1u_1x_2,\ w_3 = x_1u_2x_2$, for some $x_1, x_2 \in V^*$.

2. *Splitting.* If $a \in V$ and $u_1, u_2 \in V^+$, then $r : a \to u_1 : u_2$ is called a *splitting rule*. For strings $w_1, w_2, w_3 \in V^+$ we write $w_1 \Longrightarrow_r (w_2, w_3)$ (and we say that w_1 is splitted with respect to rule r) if $w_1 = x_1 a x_2,\ w_2 = x_1 u_1,\ w_3 = u_2 x_2$, for some $x_1, x_2 \in V^*$.

3. *Mutation.* A *mutation rule* is a context-free rewriting rule, $a \to u$, over V. For strings $w_1, w_2 \in V^+$ we write $w_1 \Longrightarrow_r w_2$ if $w_1 = x_1 a x_2,\ w_2 = x_1 u x_2$, for some $x_1, x_2 \in V^*$.

4. *Recombination.* Consider a string $z \in V^+$ (as a *crossing-over block*) and four strings $w_1, w_2, w_3, w_4 \in V^+$. We write $(w_1, w_2) \vdash_z (w_3, w_4)$ if $w_1 = x_1 z x_2,\ w_2 = y_1 z y_2$, and $w_3 = x_1 z y_2,\ w_4 = y_1 z x_2$, for some $x_1, x_2, y_1, y_2 \in V^*$.

When no ambiguity appears, the rule r and the block z are not specified when writing $\Longrightarrow$ and $\vdash$. Note that replication and splitting increase the number of strings, mutation and recombination not. Note also that the strings u_1, u_2 from replication and splitting rules, as well as z in the recombination case, are non-empty strings, but mutation rules can delete symbols.

When we will consider such operations in P systems, target indications will be added to rules and crossing-over blocks, indicating the regions where the resulting strings will be placed at the next time unit. Here we briefly consider these operations as formal operations on strings and languages.

For a language $L \subseteq V^*$ and a finite set R of replication/splitting/mutation rules, denote by $R(L)$ the set of all strings obtained by iteratively applying the rules from R, starting from strings in L; similarly, for a finite set $C \subseteq V^+$ of crossing-over blocks, we denote by $C(L)$ the language of all strings obtained by iterated recombinations, starting from the strings of L.

The proof of the following auxiliary result is an easy exercise.

Lemma 1. *If $L \subseteq V^*$ is a context-free language and R is a finite set of context-free rules over V, then also $R(L)$ is context-free.*

Theorem 1. *All the previous four operations, with respect to finite sets of rules and of crossing-over blocks, lead to context-free languages when iterated on context-free languages.*

Proof. (1) Consider a context-free language L and a finite set R of replication rules. For each rule $r : a \to u_1 || u_2 \in R$ we consider the rules $r' : a \to u_1$ and $r'' : a \to u_2$. Let R' be the set of these rules. It is clear that if $w \in L$ can lead by a sequence of replications to a string w', then we can also obtain w' by using the rules from R' (note that we do not work with multisets, hence instead of $w_1 \Longrightarrow_r (w_2, w_3)$ we can perform $w_1 \Longrightarrow_{r'} w_2$ and $w_1 \Longrightarrow_{r''} w_3$ separately, because we have as many copies of w_1 as necessary). Conversely, if a string w' can be obtained by iteratively using rules from R', starting from some string $w \in L$, then this can be also done by means of rules in R: for each $w_1 \Longrightarrow_{r'} w_2$, with $r' \in R'$, there is $r \in R$ such that $w_1 \Longrightarrow_r (w_2, w_3)$ or $w_1 \Longrightarrow_r (w_3, w_2)$.

Therefore, $R(L) = R'(L)$, which, according to Lemma 1, implies that $R(L)$ is context-free.

(2) Instead of a splitting rule $r \in R$, $r = u_1 : u_2$, consider the context-free rule $a \to u_1 c u_2$, where c is a new symbol; let R' be the set of these rules. Consider also a gsm g which leads a string of the form $x = x_1 c x_2 c \dots c x_k, k \geq 1$, nondeterministically to any of the strings $x_i, 1 \leq i \leq k$ (that is, $g(x) = \{x_i \mid 1 \leq i \leq k\}$). The language $R'(L)$ is context-free (Lemma 1) and we have $R(L) = g(R'(L))$, an equality which is easy to be proven.

(3) Directly from Lemma 1.

(4) The crossing-over with respect to a string (from a given finite set), in the sense of the recombination operation considered above, is a particular case of the splicing operation. The iterated splicing with respect to a finite set of rules preserves the context-freeness – see [17]. □

Therefore, these operations are not of much interest at the level of languages. We will immediately see that the situation is completely different if we consider them for multisets, in the framework of P systems.

3 P Systems with Worm-Objects; Definition and Examples

Because we will work here only with multisets of a finite support (only finitely many elements will have a non-null multiplicity), we will specify the multisets $\sigma : V^* \longrightarrow \mathbf{N}$ in the form $A = \{(x_1, s_1), \dots, (x_k, s_k)\}$, where $x_i \in V^*$ are those elements for which $\sigma(x_i) = s_i > 0$. The multiset with an empty support (the empty multiset) is denoted by $\emptyset$.

A *P system* (of degree $m, m \geq 1$) *with worm-objects* is a construct

$$\Pi = (V, \mu, A_1, \dots, A_m, (R_1, S_1, M_1, C_1), \dots, (R_m, S_m, M_m, C_m), i_0),$$

where:

- V is an alphabet;
- μ is a membrane structure of degree m (that is, with m membranes);
- $A_1, \dots, A_m$ are multisets of finite support over V^*, associated with the regions of μ (the initial populations of worms);
- for each $1 \leq i \leq m$, R_i, S_i, M_i, C_i are finite sets of replication rules, splitting rules, mutation rules, and crossing-over blocks, respectively, given in the following forms:
  a. replication rules: $(a \to u_1 || u_2; tar_1, tar_2)$ or $(a \to u_1 || u_2; tar_1, tar_2)\delta$, for tar_1, $tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  b. spliting rules: $(a \to u_1 : u_2; tar_1, tar_2)$ or $(a \to u_1 : u_2; tar_1, tar_2)\delta$, for tar_1, $tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;

c. mutation rules: $(a \to u; tar)$ or $(a \to u; tar)\delta$, for $tar \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;

d. crossing-over blocks: $(z; tar_1, tar_2)$ or $(z; tar_1, tar_2)\delta$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;

– $i_0 \in \{1, 2, \ldots, m\}$ specifies the *output membrane* of the system.

The $(m+1)$-tuple $(\mu, A_1, \ldots, A_m)$ constitutes the *initial configuration* of the system. By applicating the operations defined by the components $(R_i, S_i, M_i, C_i), 1 \leq i \leq m$, we can pass from a configuration to another one. This is done as usual in P systems area, according to the following *principles* (instead of a formal definition, we prefer an informal one, followed by examples):

1. The work of the system is synchronized, in each time unit (the clock is the same for the whole system), in each region, all strings which can be processed by means of rules in that region are processed; that is, the operations are applied in a maximally parallel manner.

2. The rules to be used and the copies of strings to be processed are choosen in a non-deterministic manner (observing the restriction of maximal parallelism). A string which enters an operation is "consumed" by that operation, its multiplicity is decreased by one. The multiplicity of strings produced by an operation is accordingly increased.

3. A string is processed by only one operation. For instance, we cannot apply two mutation rules, or a mutation rule and a replication one, to the same string.

4. The strings resulting from an operation (two in the case of replication, splitting, and recombination, one in the case of mutation) are communicated to the region specified by the target indications associated with rules: *here* means that the string remains in the same region where the rule has been applied, *out* means that the string is sent out that region (in this way, a string can leave also the skin membrane), while in_j means that the string is sent to membrane j, providing that this membrane is adjacent to the region where the rule is applied, directly inside this membrane; if there is no such a membrane with label j, then the rule cannot be applied (we can send strings only from a region to an adjacent region, through a single membrane, not at a larger distance).

5. When a rule is applied which also contains the symbol δ, the current membrane is dissolved; all its strings are left free in the membrane directly above it, while its rules and crossing-over blocks are lost. The skin membrane is never dissolved. The application of rules is supposed to take place from bottom-up: we first process all strings in a region which can be processed and then we dissolve the membrane (so, in the upper region, we send the result of all the possible operations).

A sequence of transitions, starting from the initial configuration, is called a *computation*. A computation is *complete* if it halts, no further rule can be applied to strings in

the last configuration. If at the end of a complete computation membrane i_0 is present in the system (it was not dissolved during the computation) and it is an elementary one, then the number of strings from region i_0 is the result of the computation. Note that a non-halting computation provides no output. For a system Π, we denote by $N(\Pi)$ the set of numbers computed in this way. By $NCP_m(\delta), m \geq 1$, we denote the sets of numbers computed by all P systems with at most m membranes. When the number of membranes is not bounded, the subscript is removed. If the dissolving action is not used, then δ is replaced by $n\delta$.

Before starting to investigate the size of families $NCP(\alpha), NCP_m(\alpha), m \geq 1, \alpha \in \{\delta, n\delta\}$, let us consider two examples.

Example 1. For the P system (of degree 2)

$$\Pi_1 = (\{a\}, [_1[_2\ ]_2]_1, \emptyset, \{(a,1)\}, (\emptyset,\emptyset,\emptyset,\emptyset), (R_2,\emptyset,\emptyset,\emptyset), 1)$$

with

$$R_2 = \{(a \to a||a; here, here), (a \to a||a; here, here)\delta\},$$

(no operation will take place in region 1, while region 2 has only replication rules) we obtain $N(\Pi_1) = \{2^i \mid i \geq 1\}$. This can be easily seen: the one-letter strings are duplicated at each step; when the rule which introduces the symbol δ is used, membrane 2 is dissolved, the strings are sent to membrane 1, and the computation stops (note that in that moment membrane 1 is an elementary one).

Of course, instead of the replication rule $a \to a||a$ we can use the splitting rule $a \to a : a$ and the result is the same.

Example 2. Consider the P system (of degree 5)

$$\Pi_2 = (V, \mu, A_1, \ldots, A_5, (R_1, S_1, M_1, C_1), \ldots, (R_5, S_5, M_5, C_5), 4),$$

with:

$$\begin{aligned}
V &= \{a, b, c, c', d, f\},\\
\mu &= [_1[_2[_3\ ]_3[_4\ ]_4[_5\ ]_5]_2]_1,\\
A_1 &= A_2 = A_4 = A_5 = \emptyset,\ A_3 = \{(d,1), (ab,1)\},\\
R_1 &= S_1 = M_1 = \emptyset,\ C_1 = \{(a; here, here)\},\\
R_2 &= \{(c \to c||f; here, in_4)\},\ S_2 = \emptyset,\ M_2 = \{(c' \to c; here)\},\\
&\quad C_2 = \{(a; here, in_5),\ (b; here, here)\delta\},\\
R_3 &= \{(a \to a||a; here, here), \{(d \to d||c'; here, here), (d \to b||c'; here, here)\delta\},\\
&\quad S_3 = M_3 = C_3 = \emptyset,\\
R_4 &= S_4 = M_4 = C_4 = \emptyset,\\
R_5 &= S_5 = M_5 = C_5 = \emptyset.
\end{aligned}$$

Let us examine in some detail the work of this system. Initially, we have string-objects only in region 3, namely a copy of ab and one of d. At each time unit, a entails the duplication of ab, while d introduces a copy of c'. Therefore, after n steps ($n \geq 0$)

we will have here n copies of c' and 2^n copies of ab (plus one copy of d). At any moment, d can be reduplicated into b and c' and the membrane is dissolved.

This means that in membrane 2 we will have the multiset characterized by $\{(b,1),(ab,2^{n+1}),(c',n+1)\}$, for some $n \geq 0$. At the next step, all symbols c' are replaced by c, while a recombination takes place among all possible pairs of strings ab. There are 2^n such pairs. If all of them are recombined by using the crossing-over block a, that is, without producing δ, then we get 2^n copies of ab in membrane 2, while 2^n copies of the same string are sent to membrane 5, where nothing happens to them (membrane 5 is a sort of storage, for keeping copies of strings which we do not need in membrane 2). If the recombination is done by using the crossing-over block b, then the membrane is dissolved. At least two copies of the string ab arrive in membrane 1, where the recombination according to a can be done forever. The computation will never stop, we get no result.

In order to get a halting computation, we have to use in region 2 as much as possible the crossing-over block a. This means that the number of copies of ab is always divided by two. At each step, each symbol c will be reduplicated, reproducing itself and sending a copy of f to the output membrane 4; at each step, $n+1$ copies of f enter membrane 4.

Such steps can be done as long as we have at least two copies of ab in region 2. When we have only one string ab, the recombination with respect to a is no longer possible, we have to recombine ab with the copy of b always present here. In parallel, further $n+1$ copies of f are collected in membrane 4. Membrane 2 is dissolved. No rule can be applied in membrane 1, because we have only one occurrence of a in the present strings. The computation is complete. Because we have sent symbols f to membrane 4 during $n+1$ steps (starting from 2^n copies of ab, to 2^0 copies, and then in the step when membrane 2 was dissolved), we get $(n+1)^2$ copies of f. In conclusion, $N(\Pi_2) = \{m^2 \mid m \geq 1\}$.

Note that in the first example we have used one-symbol strings, like in usual transition P systems, but in the second example we have two-symbol worms.

4 Computational Completeness

For a family FL of languages, we denote by lFL the family of length sets of languages in FL, that is, $lFL = \{length(L) \mid L \in FL\}$, for $length(L) = \{|x| \mid x \in L\}$. Obviously, a set of natural numbers, $M \subseteq \mathbf{N}$, is recursively enumerable (can be enumerated by a Turing machine) if and only if it is the length set of a recursively enumerable language. Let us denote by *REG, CF, RE* the families of regular, context-free, and recursively enumerable languages, respectively.

Because $lREG = lCF$ and this family contains only ultimately periodic sequences, it follows from Example 1 that $NCP_2(\delta) - lCF \neq \emptyset$.

If arbitrarily many membranes are used, then we can compute all computable sets of numbers. In the proof of this result we will use the notion of a matrix grammar.

A *matrix grammar with appearance checking* is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in

all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT.

It is known that $CF \subset MAT \subset MAT_{ac} = RE$ and that each one-letter language in the family MAT is regular, [4]. (As a consequence of this last result, from Example 2 we have that $NCP_2 - lMAT \neq \emptyset$. We do not know which is the smallest m such that $NCP_m(n\delta) - lMAT \neq \emptyset$.) Further details about matrix grammars can be found in [3] and in [18].

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are of one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation.

According to Lemma 1.3.7 in [3], for each matrix grammar there is an equivalent matrix grammar in the binary normal form.

We are now ready to give the main result of this section.

Theorem 2. $NCP(\delta) = NCP(n\delta) = lRE$.

Proof. The inclusion $NCP(\delta) \subseteq lRE$ follows from Turing-Church thesis or can be proved directly, in a straightforward (but involving a long construction) way. The inclusion $NCP(n\delta) \subseteq NCP(\delta)$ follows from the definitions. So, we only have to prove the inclusion $lRE \subseteq NCP(n\delta)$. To this aim, we make use of the equality $RE = MAT_{ac}$. More exactly, we have $lRE = lMAT_{ac} = \{length(L) \mid L \in MAT_{ac}, L \subseteq a^*\}$. Consequently, it is sufficient to consider matrix languages over the one-letter alphabet.

Let $G = (N, \{a\}, S, M, F)$ be a matrix grammar with appearance checking in the binary normal form, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and matrices of the four forms mentioned above. Assume that we have p matrices of the form $(X \to \alpha, A \to x)$, with $X \in N_1, Y \in N_1 \cup \{\lambda\}, x \in (N_2 \cup \{a\})^*$, and q matrices of the form $(X \to Y, A \to \#)$,

$X, Y \in N_1, A \in N_2$. (That is, we consider separately the matrices having rules used in the appearance checking mode and the matrices not having such rules.)

We construct the P system (of degree $s = p + 2 \cdot q + 4$)

$$\Pi = (V, \mu, A_1, \ldots, A_s, (R_1, S_1, M_1, C_1), \ldots, (R_s, S_s, M_s, C_s), 4),$$

with the following elements:

1. $V = N_1 \cup N_2 \cup \{E, a, c, \dagger\} \cup \{E_j \mid 1 \leq j \leq q\}$.

2. $\mu = [_1[_2[_3[_{h_1}\ ]_{h_1}\ \cdots [_{h_p}\ ]_{h_p}\ [_{g_1}[_{g'_1}\ ]_{g'_1}]_{g_1}\ \cdots [_{g_q}[_{g'_q}\ ]_{g'_q}]_{g_q}\ [_4\ ]_4]_3]_2]_1$
(the skin membrane is labeled with 1; with each matrix $m_i : (X \to \alpha, A \to x), 1 \leq i \leq p$, we associate a membrane with the label h_i, and with each matrix $m_i : (X \to Y, A \to \#), 1 \leq i \leq q$, we associate two membranes, with the labels g_i, g'_i).

3. $A_1 = \{(aa, 1)\}$,
$A_3 = \{(XAcE, 1)\}$, for $(S \to XA)$ the initial matrix of G;
all other initial multisets are empty.

4. $R_1 = S_1 = \emptyset$,
$M_1 = \{(Z \to Z; here) \mid \text{for all } Z \in N_1 \cup N_2\}$,
$C_1 = \{(aa; here, here)\}$.

5. $R_2 = \{(a \to a||a; out, in_4)\}$,
$S_2 = \{(a \to a:c; here, here)\}$,
$M_2 = \{(Z \to Z; here) \mid \text{for all } Z \in N_1 \cup N_2\}$,
$C_2 = \emptyset$.

6. $R_4 = S_4 = M_4 = C_4 = \emptyset$.

7. For each matrix $m_i : (X \to \alpha, A \to x), 1 \leq i \leq p$ (not to be used in the appearance checking mode), we introduce in M_3 the mutation rule
$(X \to \alpha; in_{h_i})$,
and in M_{h_i} the mutation rules
$(A \to x; out)$, $(E \to E; here)$.

8. For each matrix $m_j : (X \to Y, A \to \#), 1 \leq j \leq q$ (to be used in the appearance checking mode), we introduce in S_3 the splitting rule
$(X \to E_j:Y; here, in_{g_j})$,
and in M_3 the mutation rule
$(E_j \to E; in_{g_j})$,
while in M_{g_j} we introduce the mutation rules
$(A \to \dagger; here)$, $(\dagger \to \dagger; here)$,
and in C_{g_j} the recombination rule $(E; out, in_{g'_j})$.
At the same time, in $M_{g'_j}$ we introduce the mutation rules
$(Y \to \dagger; here)$, $(\dagger \to \dagger; here)$.

9. In S_3 we also introduce the splitting rule $(c \to c:c; out, out)$.

10. No other replication rules, splitting rules, mutation rules, or crossing-over blocks appear in the membranes of the system Π.

The shape of the membrane structure is presented in Figure 2.

Let us examine in detail the work of the system Π.

Assume that at a given moment in membrane 3 we have a string of the form $XwcE$, for some $w \in (N_2 \cup \{a\})^*$; initially, $X = X_0$ and $w = A$, for $(S \to X_0 A) \in M$.

If the splitting rule $(c \to c : c; out, out)$ is used, then we send to membrane 2 the strings Xwc and cE. If the string w contains occurrences of the terminal a, then the rules $(a \to a : c; here, here), (a \to a||a; out, in_4)$ must be used; eventually, substrings of $XwcE$ arrive in region 1, where the mutation rule $(X \to X; here)$ can be used forever. The same happens if we start from membrane 3 with a string of the form wcE and w contains at least one occurrence of a symbol from N_2.

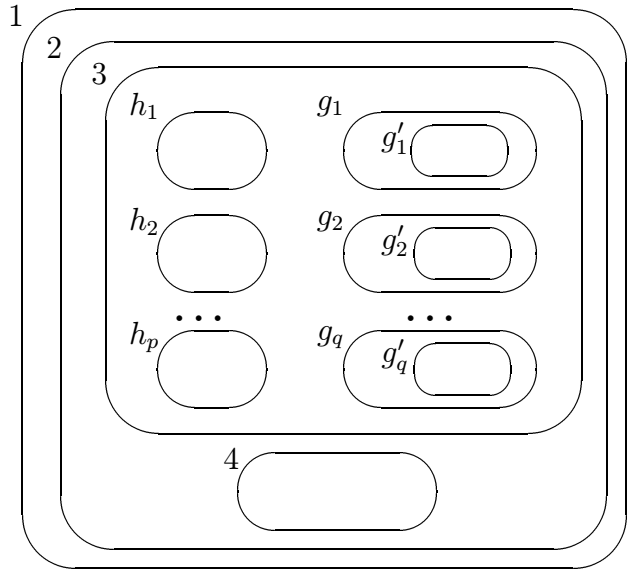

Figure 2: The membrane structure in the proof of Theorem 2

Thus, assume that we do not apply the rule $(c \to c:c; out, out)$, but we start simulating matrices of G.

If we replace the symbol X with some $\alpha \in N_1 \cup \{\lambda\}$, corresponding to a matrix $m_i : (X \to \alpha, A \to x)$, then the obtained string, αwcE, is sent to membrane h_i. The rule $(E \to E; here)$ can be applied forever. The string can leave this membrane only by applying the mutation rule $(A \to x; out)$. This precisely corresponds to simulating the matrix m_i.

If in membrane 3 we use the splitting rule $(X \to E_j : Y, here, in_{g_j})$, for some $Y \in N_1$ such that we have a matrix $m_j : (X \to Y, A \to \#)$ in G, then we get the string E_j in membrane 3 and the string $YwcE$ is sent to the membrane with the label g_j. If the symbol A appears in w, then the trap-symbol † is introduced and the rule † → † is used

forever in this membrane. If the symbol A does not appear, then no rule can be applied in membrane g_j, the string $YwcE$ waits unchanged one step. At this time, in membrane 3 we use the rule $(E_j \to E; in_{g_j})$ and a copy of E is sent to membrane g_j. Now, the recombination with respect to E can be performed. This can be done in two ways:

$$(YwcE, E) \vdash (YwcE, E), \quad \text{and} \quad (E, YwcE) \vdash (E, YwcE).$$

In the first case we send the string $YwcE$ out and the string E to membrane g'_j, in the second case the two strings change the destinations. In the second case, the computation never stops, because of the rules $(Y \to \dagger; here), (\dagger \to \dagger; here)$ from membrane g'_j. In the first case, we have returned to membrane 3 with the string $YwcE$, which is a correct simulation of the matrix m_j.

We proceed in this way, by iteratively simulating matrices of G. If we get a string of the form $a^n cE$, then we can apply the rule $(c \to c:c; out, out)$ and send out of membrane 3 the strings $a^n c, cE$. The latter string will enter no further operation. The former string can be cut by using the rule $(a \to a:c; here, here)$ in strings of the form a, ca, cac. If all the n occurrences of a are separated in this way, then each piece can be processed with the rule $(a \to a||a; out, in_4)$. In this way, a copy of each piece is sent to membrane 4 and another copy is sent to membrane 1. Because we have to get a halting computation, all the fragments must be processed in this way. This means that exactly n strings are sent to the output membrane, hence the result of the computation is n, the length of the string from $L(G)$ whose derivation was simulated by the system Π.

If the splitting of the string $a^n c$ is not complete, that is we have fragments of the form $x_1 aax_2$, and we apply to them the rule $(a \to a||a; out, in_4)$, then, clearly, we will send outside membrane 2 a copy of the string $x_1 aax_2$. This string can be recombined with the string aa (which waits here from the beginning of the computation). The two obtained strings will again contain the substring aa, hence they can be recombined again. The process never stops.

Consequently, each derivation in G can be simulated in Π and each halting computation in Π corresponds to a terminal derivation in G; moreover, we can only stop when for the string a^n derived by G we send to membrane 4 exactly n strings. In conclusion, $length(L(G)) = N(\Pi)$. □

The proof of Theorem 2 does not give a bound on the number of membranes in the system Π. It remains as an *open problem* whether or not the hierarchies $NCP_1(\alpha) \subseteq NCP_2(\alpha) \subseteq \ldots \subseteq NCP(\alpha) = lRE, \alpha \in \{\delta, n\delta\}$, is or not an infinite one. (Note that the usual argument proving that such hierarchies collapse by starting from universal grammars is not applicable in this case, because no universal matrix grammar with appearance checking is known; in particular, in the proof we have made an essential use of the binary normal form, which assumes that there is a unique and short "initial string", XA, for the matrix $(S \to XA)$; universal results are based on introducing the "code" of a particular grammar as a starting string of the universal grammar.)

Another *open problem* of interest is whether or not the maximal length of strings appearing in a P system with worm-objects induces an infinite hierarchy on the family of computed sets of numbers.

5 Solving HPP in Quadratic Time

Consider a directed graph $\gamma = (U, E)$ and two distinct nodes in U, i_{in} and i_{out}. The Hamiltonian Path Problem (HPP, for short) for γ asks whether or not there is a path from i_{in} to i_{out} which passes exactly once through all nodes of the graph. Note that we do not ask for actually finding a Hamiltonian path, but only whether or not such a path exists. This was the problem also addressed in the Adleman's historical experiment, [1]; we will solve here the problem in a quadratic time by following an algorithm similar in many respects to that used in [1].

Assume that U contains n nodes, identified with the numbers $1, 2, \ldots, n$ (hence the Hamiltonian paths will consist of $n-1$ arcs) and that the maximum outdegree of the graph (the number of arcs having the origin in a given node) is equal to k. Because the arcs of the form (i, i) are useless, we can ignore them, hence we can suppose that we have $k \leq n-1$.

By a simple renumbering, assume that $i_{in} = 1$ and that $i_{out} = n$.

We construct the P system with worm-objects Π_γ (of degree n), associated with γ, with the following components:

$$
\begin{aligned}
V &= \{\langle i, r\rangle, [i, r], \langle i, r; j_1, \ldots, j_s\rangle \mid \\
&\quad 1 \leq i \leq n, 0 \leq r \leq k, \{j_1, \ldots, j_s\} \subseteq \{j \in U \mid (i, j) \in E\}\}, \\
\mu &= [_{n-1}[_n\,]_n[_{n-2} \cdots [_2[_0\,]_0]_2 \cdots]_{n-2}]_{n-1}, \\
A_0 &= \{(\langle 1, 0\rangle, 1)\};\ \text{all other initial multisets are empty}, \\
R_0 &= \{(\langle i, r\rangle \to [i, r]\langle j_1, r+1\rangle || \langle i, r; j_2, \ldots, j_s\rangle; here, here), \\
&\quad (\langle i, r; j_h, \ldots, j_s\rangle \to [i, r]\langle j_h, r+1\rangle || \langle i, r; j_{h+1}, \ldots, j_s\rangle; here, here), \\
&\quad (\langle i, r; j_{s-1}, j_s\rangle \to [i, r]\langle j_{s-1}, r+1\rangle || [i, r]\langle j_s, r+1\rangle; here, here) \mid \\
&\quad \text{for all } 1 \leq i \leq n-1, 0 \leq r \leq n-2, \text{ where } j_1, \ldots, j_s \text{ are all the nodes} \\
&\quad \text{such that } (i, j_l) \in E, 1 \leq l \leq s\}, \\
M_0 &= \{(\langle n, n-1\rangle \to [n, n-1]; out)\}, \\
S_0 &= C_0 = \emptyset, \\
R_i &= S_i = C_i = \emptyset,\ \text{for all } 2 \leq i \leq n-1, \\
M_i &= \{([i, j] \to [i, j]; out) \mid 1 \leq j \leq n-1\},\ \text{for all } 2 \leq i \leq n-2, \\
M_{n-1} &= \{([n, n-1] \to [n, n-1]; in_n)\}, \\
R_n &= S_n = M_n = C_n = \emptyset.
\end{aligned}
$$

The membrane structure of the system Π_γ is given in Figure 3. (The skin membrane is labeled with $n-1$.) The system works in the following way. We start from the unique object $\langle 1, 0\rangle$, present in the inner membrane, that with the label 0, by repeatedly using replication rules. These rules always prolong a string which represents a path in the graph starting from node 1. For instance, if we have a string of the form $x = [1, 0][i_2, 1] \ldots [i_r, r-1]\langle j, r\rangle$, then we can pass to the strings $x_1 = [1, 0][i_2, 1] \ldots [i_r, r-1]\langle j_1, r\rangle$, $x_2 = [1, 0][i_2, 1] \ldots [i_r, r-1]\langle i_r, r-1; j_2, \ldots, j_s\rangle$, where $(i_r, j_1) \in E, \ldots, (i_r, j_s) \in E$. Similarly, from a string of the form $y = [1, 0][i_2, 1] \ldots [i_r, r-1]\langle i_r, r-1; j_2, \ldots, j_s\rangle$

we can pass to two strings $y_1 = [1,0][i_2,1]\ldots[i_r,r-1]\langle j_2,r\rangle$, $y_2 = [1,0][i_2,1]\ldots[i_r,r-1]\langle i_r,r-1;j_3,\ldots,j_s\rangle$. In both cases, the path has been correctly continued.

No such a path can be continued if either it reaches node n or we have already made $n-1$ steps (hence the path already passes through n nodes, any further step will surely repeat a node). In this way, we can generate all paths in the graph γ starting in node 1 and of length (as the number of arcs) at most $n-1$.

Only strings which end with $\langle n,n-1\rangle$ can be sent outside membrane 0.

In each membrane $i, 2 \leq i \leq n-2$, a string can be only processed if it contains a symbol of the form $[i,t]$, for some $1 \leq t \leq n-1$ (this means that node i was vizited at time t). If this is the case, then the string is sent unmodified to the next membrane. If a string reaches the skin membrane, then it can be sent to the output membrane providing that it contains the symbol $[n,n-1]$ (this means that the corresponding path ends in node n).

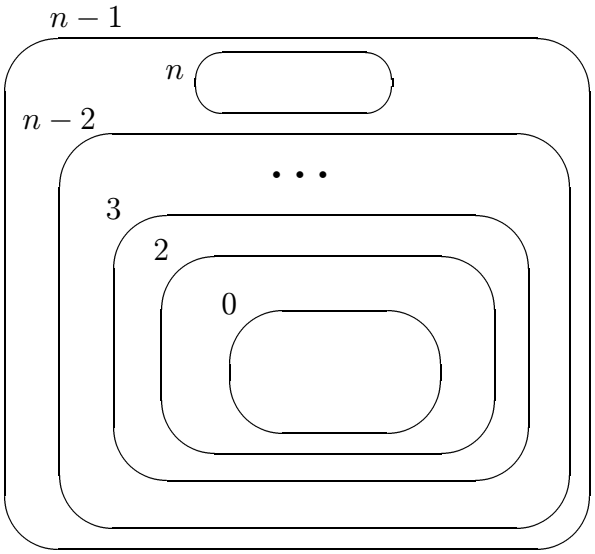

Figure 3: The membrane structure for solving HPP

Therefore, the computation always stops, but we do not always have a string in the output membrane. This happens (that is, $N(\Pi_\gamma) \neq \emptyset$) if and only if at least a Hamiltonian path exists in the graph γ, from node 1 to node n.

Let us now compute the maximum number of steps a computation in Π_γ can have. It is clear that after obtaining a symbol $\langle i,r\rangle$ we need at most $k-1$ steps for obtaining all the possible continuations of a path which has reached node i (the outdegree of the graph is k): in each step we prolong one path, while when we have only two further paths to continue, we prolong both of them at the same time; write the tree representing all paths in γ of length at most $n-1$, such that each path from the root of the tree to a leaf corresponds to a path in γ and conversely; this tree has at most n levels; in $k-1$ computation steps in Π_γ we pass (at least) from a level to the next one. Consequently, we cover the tree (which means that we generate all paths in γ of length less than or equal to $n-1$) in at most $(k-1)n$ steps. In one more step we send out of membrane 0 the strings which are ended with $\langle n,n-1\rangle$, then we need further $n-2$ steps in order

to send a string to the output membrane. In total, we perform at most $n^2 - n - 1$ steps (make use of the fact that $k \leq n - 1$).

In conclusion, we have an answer whether or not the Hamiltonian Path Problem for γ has a solution after at most $n^2 - n - 1$ steps performed by the P system Π_γ. This conclusion deserves to be stated as a theorem:

Theorem 3. *The HPP problem can be solved by P systems with worm-objects (without membrane dissolving) in a quadratic time with respect to the number of nodes.*

The two phases of our computation, generating all candidate paths (in membrane 0) and then check whether or not at least a path is Hamiltonian (in membranes $2, 3, \ldots, n-1$), are similar to the two phases of Adleman's algorithm [1], with the difference that we need a quadratic time for producing the candidate solutions (Adleman performs this in a constant parallel biochemical time); however, we grow only candidate solutions of a prescribed length.

6 Solving SAT in Linear Time

Now, we can proceed as Lipton [8], extending the previous procedure to the SAT problem. Because the truth-assignments to n variables $x_1, \ldots, x_n$ correspond to the paths from a given node to another given node in a graph with outdegree 2 (see [8]), the solution is obtained in linear time: n steps for generating all truths-assignments and m steps to check the truth value of each of the m clauses. (Note that, in general, if we have a graph with n nodes and the outdegree bounded by a constant k, then also the HPP problem is solved in linear time: the system Π_γ in the proof of Theorem 3 performs at most $kn - 1$ steps.)

Theorem 4. *The SAT problem can be solved by a P system with worm-objects (and without using the dissolving action) in a linear time (depending both on the number of variables and the number of clauses).*

Although the reader can surely figure out the P system proving this result, for the sake of the completeness we present the construction.

Let $\sigma = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each clause $C_i, 1 \leq i \leq m$, is a disjunction $C_i = y_1 \vee y_2 \vee \ldots \vee y_r$, with each y_j being either a propositional variable, x_s, or its negation, $\sim x_s$, for $s \in \{1, 2, \ldots, n\}$.

We construct the P system Π_σ (of degree $m + 2$) with the following components:

$$
\begin{aligned}
V &= \{t_i, f_i \mid 1 \leq i \leq n\},\\
\mu &= [_m[_{m+1}\,]_{m+1}[_{m-1}\ \cdots\ [_2[_1[_0\,]_0]_1]_2\ \cdots\ ]_{m-1}]_m,\\
A_0 &= \{(t_1, 1), (f_1, 1)\};\ \text{all other initial multisets are empty},\\
R_0 &= \{(t_i \to t_i t_{i+1} || t_i f_{i+1}; here, here),\\
&\qquad (f_i \to f_i t_{i+1} || f_i f_{i+1}; here, here) \mid \text{ for all } 1 \leq i \leq n-2\}\\
&\quad \cup \{(t_{n-1} \to t_{n-1} t_n || t_{n-1} f_n; out, out),\\
&\qquad (f_{n-1} \to f_{n-1} t_n || f_{n-1} f_n; out, out)\},
\end{aligned}
$$

$$
\begin{aligned}
S_0 &= M_0 = C_0 = \emptyset, \\
M_j &= \{(t_i \to t_i; out) \mid 1 \le i \le n, \text{ if } C_j \text{ contains } x_i\} \\
&\quad \cup \{(f_i \to f_i; out) \mid 1 \le i \le n, \text{ if } C_j \text{ contains } \sim x_i\}, \\
R_j &= S_j = C_j, \text{ for all } 1 \le j \le m-1, \\
M_m &= \{(t_i \to t_i; in_{m+1}) \mid 1 \le i \le n, \text{ if } C_m \text{ contains } x_i\} \\
&\quad \cup \{(f_i \to f_i; in_{m+1}) \mid 1 \le i \le n, \text{ if } C_m \text{ contains } \sim x_i\}, \\
R_m &= S_m = C_m.
\end{aligned}
$$

The way this system works is obvious: in membrane 0 we generate all truth-assignments in the form of strings of length n composed of $t_i, f_i, 1 \le i \le n$, in all possible combinations (this takes n steps); a string can exit a membrane $j, 1 \le j \le m-1$, if and only if the clause C_j asumes the value *true* for the truth-assigment corresponding to the string; this means that a string reaches the skin membrane if and only if all clauses $C_1, \ldots, C_{m-1}$ are satisfied; a string with this property enters the output membrane if and only if it also satisfies clause C_m. Checking the truth values of clauses takes m steps, hence the computation halts after at most $n + m$ steps.

References

[1] L. M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226 (November 1994), 1021–1024.

[2] C. Calude, Gh. Păun, *Computing with Cells and Atoms*, Taylor and Francis, London, 2000 (Chapter 3: "Computing with Membranes").

[3] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.

[4] D. Hauschild, M. Jantzen, Petri nets algorithms in the theory of matrix grammars, *Acta Informatica*, 31 (1994), 719–728.

[5] T. Head, Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49 (1987), 737–759.

[6] S. N. Krishna, R. Rama, A variant of P systems with active membranes: Solving NP-complete problems, *Romanian J. of Information Science and Technology*, 2, 4 (1999).

[7] S. N. Krishna, R. Rama, Computing with P systems, submitted, 2000.

[8] R. J. Lipton, Using DNA to solve NP-complete problems, *Science*, 268 (April 1995), 542–545.

[9] A. Mateescu, Gh. Păun, G. Rozenberg, A. Salomaa: Simple splicing systems. *Discrete Appl. Math.*, 84 (1998), 145–163

[10] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, in press, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[11] Gh. Păun, Computing with membranes. An introduction, *Bulletin of the EATCS*, 67 (Febr. 1999), 139–152.

[12] Gh. Păun, Computing with membranes – A variant: P Systems with Polarized Membranes, *Intern. J. of Foundations of Computer Science*, 11, 1 (2000), and *Auckland University, CDMTCS Report* No 098, 1999 (www.cs.auckland.ac.nz/CDMTCS).

[13] Gh. Păun, P systems with active membranes: Attacking NP complete problems, submitted 1999, and *Auckland University, CDMTCS Report* No 102, 1999 (www.cs.auckland.ac.nz/CDMTCS).

[14] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, to appear, and *Turku Center for Computer Science-TUCS Report* No 218, 1988 (www.tucs.fi).

[15] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing. New Computing Paradigms*, Springer-Verlag, Heidelberg, 1998.

[16] Gh. Păun, T. Yokomori, Membrane computing based on splicing, *Preliminary Proc. of Fifth Intern. Meeting on DNA Based Computers* (E. Winfree, D. Gifford, eds.), MIT, June 1999, 213–227.

[17] D. Pixton, Splicing in abstract families of languages, *Technical Report of SUNY Univ. at Binghamton, New York*, 1997.

[18] G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages*, 3 volumes, Springer-Verlag, Berlin, 1997.

[19] M. P. Schützenberger: On finite monoids having only trivial subgroups. *Inform. Control*, 8 (1965), 190–194

[20] M. Sipper, Studying Artificial Life using a simple, general cellular model, *Artificial Life Journal*, 2, 1 (1995), 1–35.