Computing with Membranes: One More Collapsing Hierarchy

Carlos MARTÍN-VIDE
Research group in Mathematical Linguistics and Language Engineering
Rovira i Virgili University, Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: cmv@astor.urv.es

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

Abstract. We prove that the hierarchy on the number of membranes in P systems with worm-objects, as introduced in [1], collapses: systems of degree six suffice in order to generate all recursively enumerable sets of natural numbers.

1 Introduction

P systems are a class of distributed parallel computing models introduced in [3], inspired from the way the alive cells process chemical compounds, energy, and information. In short, in the *regions* delimited by a *membrane structure*, one places *multisets* of *objects*, which evolve according to *evolution rules* associated with the regions; a *computation* consists of transitions among system *configurations*; the *result* of a halting computation is the number of objects present in the final configuration in a specified *output membrane*. Details can be found in [3], [4]. When membrane division is allowed, NP-complete problems can be solved in linear time, [2].

The objects in a P system can be identified by symbols in or by strings on a given alphabet; in the latter case we do not use multisets, but we work with usual languages (the result of a computation is a language, too). A combined variant was proposed in [1] (under the name of P systems with worm-objects): one works with multisets of string-objects and one considers the result of a computation as the number of strings in a given output membrane. As operations able to increase and decrease the number of strings one uses *replication, splitting, mutation*, and *recombination*.

It is proved in [1] that (1) such systems are computationally complete (they can compute all recursively enumerable sets of natural numbers) and that (2) they can solve the SAT problem in linear time (without using membrane division). The proof of the computational completeness from [1] does not provide a bound on the number of membranes; this is formulated as an *open problem* in [1]. We solve this problem here: the hierarchy collapses, six membranes suffice.

2 P Systems with Worm-Objects

We start by recalling the definition of P systems with worm-objects, as introduced in [1].

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 $w_1 = x_1 a x_2,\ w_2 = x_1 u_1 x_2,\ w_3 = x_1 u_2 x_2$, for some $x_1, x_2 \in V^*$.

[1]Research supported by the Direcció General de Recerca, Generalitat de Catalunya (PIV).

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, $r : a \to u$, 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 $(w_1, w_2) \Longrightarrow_z (w_3, w_4)$ 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 $\Longrightarrow$. Note that replication and splitting increase the number of strings, mutation and recombination not.

We work here only with multisets $\sigma : V^* \to \mathbf{N}$ such that only finitely many elements have a non-null multiplicity, thus we can specify σ in the form $A = \{(x_1, s_1), \ldots, (x_k, s_k)\}$, where $x_i, 1 \leq i \leq k$, are those elements of V^* for which $\sigma(x_i) = s_i > 0$.

A membrane structure will be represented by a string of matching labeled parentheses (those corresponding to the tree which describes the membrane structure).

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

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

where:

- V is an alphabet;
- μ is a membrane structure of degree m (that is, with m membranes);
- $A_1, \ldots, A_m$ are multisets of finite support over V^*, associated with the regions of μ;
- 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: $(a \to u_1||u_2; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  b. spliting rules: $(a \to u_1|u_2; tar_1, tar_2)$, for $tar_1, tar_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  c. mutation rules: $(a \to u; tar)$, for $tar \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$;
  d. crossing-over blocks: $(z; tar_1, tar_2)$, 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; it should be an elementary membrane of μ, that is, a membrane containing no other membrane.

The $(n+1)$-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 define transitions from a configuration to another one. This is done as usual in P system area, according to the following *principles*:

1. The work of the system is synchronized, in each time unit, in each region, all strings which can be processed are processed.
2. The rules to be used and the copies of strings to be processed are choosen in a nondeterministic manner.
3. 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.

4. 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.
5. The strings resulting from an operation 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 of 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 region; if there is no such a membrane with label j, then the rule cannot be applied.

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. Its result consists of the number of strings in region i_0 at the end of the computation. 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 $NCP_m, m \geq 1$, 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. We also denote by lRE the family of recursively enumerable sets of natural numbers (they are the length sets of recursively enumerable languages).

3 A Collapsing Hierarchy

In [1] it is proved that each recursively enumerable set of natural numbers is in NCP and one asks whether or not the hierarchy $NCP_1 \subseteq NCP_2 \subseteq \ldots \subseteq NCP$ is infinite. We solve here (in negative) this problem.

Theorem 1. *$lRE = NCP_m = NCP$, for all $m \geq 6$.*

Proof. It is sufficient to prove that the length set of any given recursively enumerable language is in NCP_6. To this aim, let us consider a type-0 Chomsky grammar $G = (N, T, S, P)$ in the Kuroda normal form, that is, with the rules in P of one of the following forms: $A \to AB, A \to a, A \to \lambda, AB \to CD$, for $a \in T$ and $A, B, C, D \in N$. Each non-context-free rule is supposed to be labeled in a one-to-one manner with elements from a given set of labels.

We construct the P system of degree 6

$$\Pi = (V, \mu, A_1, \ldots, A_6, (R_1, S_1, M_1, C_1), \ldots, (R_6, S_6, M_6, C_6), 4),$$

with:

1. $V = N \cup T \cup \{X, X', Y, Z, c, f, \dagger\} \cup \{B_r, B' \mid r : AB \to CD \in P\}$.
2. $\mu = [_1\, [_2\, [_3\, [_4\,]_4\,]_3\, [_5\, [_6\,]_6\,]_5\,]_2\,]_1$.
3. $A_2 = \{(SY, 1)\}$,
   $A_6 = \{(B_r B', 1) \mid r : AB \to CD \in P\}$,
   $A_1 = A_3 = A_4 = A_5 = \emptyset$.
4. $R_1 = \{(a \to Z || f; in_2, in_2) \mid a \in T\}$,
   $S_1 = \{(a \to a|c; here, here) \mid a \in T\}$,
   $M_1 = \{(A \to A; here) \mid A \in N\}$,
   $C_1 = \emptyset$.
5. $R_2 = \emptyset$,
   $S_2 = \{(A \to CDX|B_r; in_3, in_5) \mid r : AB \to CD \in P\}$,

$M_2 = \{(A \to x; here) \mid A \to x \in P\} \cup \{(B_r \to X; in_3) \mid r : AB \to CD \in P\}$
$\cup \{(a \to \dagger; here) \mid a \in T\} \cup \{(Y \to \lambda; out),\ (f \to f; in_3),\ (\dagger \to \dagger; here)\}$,
$C_2 = \emptyset$.

6. $R_3 = S_3 = \emptyset$,
$M_3 = \{(X' \to \lambda; out),\ (f \to f; in_4)\}$,
$C_3 = \{(X; in_4, in_4)\}$.
7. $R_4 = S_4 = \emptyset$,
$M_4 = \{(X \to X'; out)\} \cup \{(A \to A; here) \mid A \in N\}$,
$C_4 = \emptyset$.
8. $R_5 = S_5 = \emptyset$,
$M_5 = \{(B \to B'; in_6),\ (B' \to \lambda; out) \mid B \in N\} \cup \{(Y \to \dagger; here),\ (\dagger \to \dagger; here)\}$,
$C_5 = \emptyset$.
9. $R_6 = S_6 = \emptyset$,
$M_6 = \{(Y \to \dagger; here,\ (\dagger \to \dagger; here)\}$,
$C_6 = \{(B_r B'; out, here) \mid r : AB \to CD \in P\}$.

Let us examine the work of this system.

First, a general observation: the symbol $\dagger$ is a trap, once introduced it can evolve forever, which means that no output is obtained.

Initially, we have strings only in membranes 2 and 6. Assume that at some moment in membrane 2 we have a string of the form wY (initially, $w = S$). Any context-free rule from P can be simulated by the corresponding mutation rule from M_2. Assume that $w = w_1 A w_2$, for some $w_1, w_2 \in (N \cup T)^*$ such that $r : AB \to CD \in P$. By using the splitting rule $(A \to CDX|B_r; in_3, in_5)$, we produce the strings $w_1 CDX, B_r w_2 Y$, each one in exactly one copy (the processed copy of wY is no longer present). The first string is sent to membrane 3, the second one to membrane 5.

In membrane 3, the string $w_1 CDX$ cannot enter any operation (for recombination we need two strings), it will wait here.

If in membrane 5 we use the rule $(Y \to \dagger; here)$, then computation will never end, hence we have to use a rule of the form $(F \to F'; in_6)$, associated with some rule $r' : EF \to HJ \in P$, and the resulting string is sent to membrane 6. We have to consider two cases: (1) the string sent to membrane 6 is of the form $B_r B' w_2' Y$, (2) this is not true. Note that the first case corresponds to the situation when the starting string wY was of the form $w_1 AB w_2' Y$, for $r : AB \to CD \in P$.

In the second case, the string does not contain the substring $B_r B'$, hence it cannot be recombined in membrane 6 with the corresponding string $B_r B'$ (which waits here from the beginning of the computation, in one copy). Thus, the string must be processed with the rule $(Y \to \dagger; here)$ and the computation will never finish.

Consequently, we can proceed further only when the string is $B_r B' w_2' Y$. The use of the rule $(Y \to \dagger; here)$ will make the computation never end, but we can avoid this by using the recombination. We again have two cases, depending on the order of using the strings:

$$a.\quad (B_r B' w_2' Y, B_r B') \vdash (B_r B', B_r B' w_2' Y)\ \ (out, here),$$
$$b.\quad (B_r B', B_r B' w_2' Y) \vdash (B_r B' w_2' Y, B_r B')\ \ (out, here).$$

If the string $B_r B'$ is sent out and the string $B_r B' w_2' Y$ remains in membrane 6, then this second string has to be processed by using the rule $(Y \to \dagger; here)$ (we cannot use the recombination, because we no longer have here a copy of $B_r B'$, it has been consumed at the previous recombination); the computation will never end. Therefore, we have to proceed as in case b: in membrane 6 we reproduce the string $B_r B'$, maybe needed at subsequent steps, and the string $B_r B' w_2' Y$ is sent to

membrane 5. We know that the simulation of the rule $r : AB \to CD$ was started in a correct manner, the symbols AB were present in adjacent positions in the starting string.

In membrane 5 we remove the symbol B' (by $(B' \to \lambda; out)$) and we send to membrane 2 the string $B_r w_2 Y$. Here, the symbol B_r is replaced with X and the obtained string, Xw_2Y, is sent to membrane 3. Now, the recombination in membrane 3 is possible, we get the strings $w_1CDXw_2'Y, X$, which are sent to membrane 4. Here, the symbol X is replaced by X' and the strings are sent simultaneously to membrane 3, where X' is erased, and the obtained strings, λ and $w_1CDw_2'Y$, are sent to membrane 2. The empty string will never be processed again, the second string is exactly the result of simulating the rule $r : AB \to CD$.

The process can be iterated (hence any derivation in G can be simulated in Π). At any moment, in membrane 2 we can use the rule $(Y \to \lambda; out)$. A string $w \in (N \cup T)^*$ is sent to membrane 1. We have two cases:

(1) If the string w is composed of terminal symbols only, then it can be splitted by the rules $(a \to a|c; here, here), a \in T$, then the fragments can be sent to membrane 2 by using the rule $(a \to Z||f; in_2, in_2), a \in T$. All fragments containing the symbol f are sent to membrane 3 and, from here, to membrane 4, the output one, contributing to the number generated by the computation. If any fragment sent to membrane 2 contains at least one occurrence of a symbol from T, then the computation will never end. Indeed, if we replicate a string $u \in T^*$ with $|u| \geq 2$, then both the obtained strings will contain at least one terminal symbol. For instance, take $u = u_1b_1u_2b_2u_3$, with $b_1, b_2 \in T$ and apply the rule $(b_1 \to Z||f; in_2, in_2)$, then the strings $u_1Zu_2b_1u_3, u_1fu_2b_2u_3$ are sent to membrane 2. The second string can be sent to membrane 3 by using the rule $(f \to f; in_3)$, but $u_1Zb_2u_2u_3$ will evolve forever in membrane 2 by the rules $b_2 \to \dagger, \dagger \to \dagger$.

This observation is very important: in order to correctly close the computation, the string $w \in T^*$ sent to membrane 3 should be splitted symbol by symbol and only such pieces can be sent to membrane 2. That is, we produce exactly $|w|$ copies of the symbol f, hence the result of the computation will be exactly $|w|$, as desired.

(2) If the string w contains non-terminal symbols with respect to G, then the computation never ends. First, the string is cut by the rules $(a \to a|c; here, here), a \in T$, in parts. If any part contains only symbols from N and occurrences of c, then rules $(A \to A; here)$ from membrane 1 can be applied forever. Fragments which contain at least a terminal symbol can be processed by means of rules $(a \to Z||f; in_2, in_2), a \in T$; copies of them, with a replaced with Z and f, are sent to membrane 2. There is at least a fragment containing a nonterminal symbol; by reduplication, both the string which contains the symbol Z and the string which contains the symbol f will contain a nonterminal occurrence. The string which contains the symbol f will eventually arrive in membrane 4, and it can be processed here forever by rules of the form $(A \to A; here), A \in N$.

This completes the analysis of the work of the system Π. We obtain $N(\Pi) = \{|w| \mid w \in L(G)\}$, hence $lRE \subseteq NCP_6$. □

It remains as an *open problem* to find which of the inclusions $NCP_i \subseteq NCP_{i+1}$, for $i = 1, 2, 3, 4, 5$, is proper.

References

[1] J. Castellanos, A. Rodriguez-Paton, Gh. Păun, Computing with membranes: P systems with worm-objects, submitted, 2000.

[2] 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), 357–367.

[3] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, in press.

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