

P Systems with Replicated Rewriting

Shankara Narayanan KRISHNA, Raghavan RAMA

Department of Mathematics

Indian Institute of Technology, Madras

Chennai 600036, India

E-mail: ramar@iitm.ac.in

Abstract. P Systems are computing models, where objects can evolve in parallel within a hierarchical membrane structure. Recent results show that this model is a promising framework for solving NP-complete problems in polynomial time. (Of course, “solving” NP-complete problems in this framework means trading space for time and using an exponential space. The interest for P systems in this respect lies in the fact that we can generate this space in a very natural, easy, and systematic manner, inherent to P systems.) The present paper considers P systems using strings as the data structure and replicated rewriting as the control structure. This is a variant of P systems which can attack NP-complete problems: here we exemplify this with SAT and HPP, which can be solved in linear time. We also prove that this variant is computationally complete, it characterizes the recursively enumerable languages.

1 Introduction

In the area of natural computing, P systems are new computing models based on the way nature organizes the cellular level in living organisms. Different processes developed at this level can be thought of as computations. In his seminal paper [4], Gh. Păun considers systems based on a hierarchically arranged finite cell structure consisting of several cell membranes embedded in a main membrane called the skin membrane. The membranes delimit regions where objects are placed. In [4], two basic classes of P systems are considered, with symbol-objects and with string-objects. Starting from these main variants, several further variants were considered, see [1, 3, 5, 6, 7, 8].

Here, we consider a variant of P systems using string-objects with the object evolution based on rewriting. It is known that certain variants of P systems, such as P systems with active membranes [6] and P systems with worm-objects [1], are able of solving NP-complete problems in polynomial type. For solving hard problems with P systems which use rewriting rules, we need to replicate strings, in order to get an exponential space in a linear time. This is the starting idea of our paper: to consider rules which replicate strings at the same time when rewriting them. Such P systems with replicated rewriting are introduced in the following section. Then we give an important application of this variant, namely solving NP-complete problems in polynomial time. In the last section, we also prove that this variant is computationally universal, it can generate all recursively enumerable languages.

2 P Systems with Replicated Rewriting

We directly define the variant of P systems we introduce in this paper; for further notions, we refer the reader to [4].

Definition 2.1 *A P System based on replicated rewriting is a construct*

$$\Pi = (V, T, \mu, M_1, M_2, \dots, M_m, R_1, R_2, \dots, R_m),$$

where:

- (1) $m \geq 1$;
- (2) V is an alphabet (the total alphabet of the system);
- (3) $T \subseteq V$ (the terminal alphabet);
- (4) μ is a membrane structure with m membranes, labeled with $1, 2, \dots, m$;
- (5) M_1, \dots, M_m are finite languages over V ;
- (6) R_1, R_2, \dots, R_m are finite sets of developmental rules of the form $X \rightarrow (v_1, tar_1) || (v_2, tar_2) || \dots || (v_n, tar_n)$, $n \geq 1$, where $tar_i \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq m\}$, $X \in V, v_i \in V^*, 1 \leq i \leq n$. If $n > 1$, then the rule is a replicated rewriting rule, otherwise it is just a rewriting rule.

The languages M_i and the sets R_i of rules are associated with the regions of $\mu, 1 \leq i \leq m$.

When a rule $X \rightarrow (v_1, tar_1) || (v_2, tar_2) || \dots || (v_n, tar_n)$ is used to rewrite a string $x_1 X x_2$, we obtain n strings $x_1 v_i x_2$, which are sent to the regions indicated by $tar_i, 1 \leq i \leq n$, respectively. When $tar_i = here$, the string remains in the same region, when $tar_i = out$, the string exits from the current membrane, and when $tar_i = in_j$, the string is sent to membrane j , providing that it is directly inside the membrane where the rule was applied; if there is no such a membrane j inside the membrane where we work, then the rule cannot be applied.

A computation is defined as usual in rewriting P systems: we start from the strings present in the initial configuration of the system and proceed iteratively, by transition steps done by using the rules in parallel, in each membrane to all strings which can be rewritten by local rules; the result is collected outside the system, at the end of halting computations. Note that each string is processed by one rule only, the parallelism refers to processing simultaneously all available strings by all applicable rules in the corresponding regions. If several rules can be applied to a string, at several places each, then we take only one rule and only one possibility to apply it and consider the obtained string as the next state of the object described by the string.

Note that we do not consider here further features, such as priorities among rules or the possibility of dissolving membranes, because we do not need them in the following sections.

The language generated by a system Π is denoted by $L(\Pi)$ and it consists of all strings over T^* sent out of the system during a halting computation. We denote by $ERRP$ the family of languages generated as above by P Systems based on replicated rewriting.

Theorem 3.2 *The Hamiltonian Path Problem can be solved using replicated rewriting P systems in a time linear in the number of nodes of the given graph.*

Proof. Let $G = (V, E)$ be a graph, with $V = \{a_1, \dots, a_n\}$ the set of nodes and E the set of edges. We construct the P system

$$\Pi = (V, V, \mu, M_1, \dots, M_{n+2}, R_1, \dots, R_{n+2}),$$

where:

$$\begin{aligned} V &= \{a_i, a'_i, c_i \mid 1 \leq i \leq n\}, \\ \mu &= [{}_1[{}_2 \cdots [{}_{n+2}]_{n+2} \cdots]_2]_1, \\ M_{n+2} &= \{c_1 a_1, c_1 a_2, \dots, c_1 a_n\}, \quad M_i = \{\lambda\}, \text{ for all } 1 \leq i \leq n+1, \\ R_{n+2} &= \{a_j \rightarrow (a'_j a_{l_1}, out) \mid \dots \mid (a'_j a_{l_k}, out) \mid 1 \leq j \leq n, \\ &\quad \{a_j, a_{l_s}\} \in E, 1 \leq s \leq k\}, \\ R_{n+1} &= \{c_i \rightarrow (c_{i+1}, in_{n+2}) \mid 1 \leq i \leq n\} \\ &\quad \cup \{c_{n+1} \rightarrow (\lambda, out)\}, \\ R_i &= \{a'_i \rightarrow (a_i, out)\}, \text{ for } 1 \leq i \leq n. \end{aligned}$$

This system works as follows. In the initial configuration, we have the strings $c_1 a_1, c_1 a_2, \dots, c_1 a_n$ in the central membrane. First we increment c_i and the string moves to membrane $n+1$. Here, we prolong the paths from every node: if we have a path described by a string $c_{p+2} a'_{i_1} a'_{i_2} \dots a'_{i_p} a_i$, then we produce all strings of the form $c_{p+2} a'_{i_1} a'_{i_2} \dots a'_{i_p} a'_i a_{l_s}$, where a_{l_s} runs over all nodes in the graph which are adjacent to a_i . The paths generated in this way are moved immediately to membrane $n+1$. The process is iterated n steps, that is, until c_j reaches the subscript $n+1$. This guarantees that all possible paths of length n are produced. All these strings are sent to membrane n by using the rule $c_{n+1} \rightarrow (\lambda, out)$ from membrane $n+1$. The strings are now checked with respect to the Hamiltonian property: a string can exit a membrane j if and only if the node a_j is present in it, $1 \leq j \leq n$. Therefore, a Hamiltonian path exists iff and only if any string will leave the system. If this is the case, it happens after $3n+1$ steps: in $2n$ steps we generate all paths of length n , in membranes $n+1, n+2$, in one step we send the obtained strings to membrane n , then further n steps are necessary in order to send a string out of the system. Consequently, HPP is solved in $3n+1$ steps. \square

4 The Computational Universality

As expected, our systems can generate all recursively enumerable languages.

Theorem 4.1 $RE = ERRP$.

Proof. The inclusion $ERRP \subseteq RE$ follows from the Church-Turing thesis. We prove the other inclusion as follows. Let $G = (N, T, S, M, F)$ be a matrix grammar with appearance checking in binary normal form, see [2]. Then the matrices in M are of the following forms:

- (1) $(S \rightarrow XA), X \in N_1, A \in N_2$.
- (2) $(X \rightarrow Y, A \rightarrow x), X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*$.
- (3) $(X \rightarrow Y, A \rightarrow \#), X, Y \in N_1, A \in N_2$, and $\#$ is a special symbol.
- (4) $(X \rightarrow \lambda, A \rightarrow x), X \in N_1, A \in N_2, x \in T^*$.

There is only one matrix of type 1 and F consists exactly of all rules $A \rightarrow \#$ appearing in matrices of type 3. The symbol $\#$ is a trap-symbol, once introduced, it is never removed. All derivations can terminate only by a matrix of type 4. Assume that there are $l - 1$ matrices, labeled by m_2, m_3, \dots, m_l . Let m_2, m_3, \dots, m_k be matrices of types 2 and 4, and let m_{k+1}, \dots, m_l be matrices of type 3.

We construct the P system

$$\Pi = (V, T, \mu, M_1, M_2, \dots, M_l, R_1, R_2, \dots, R_l),$$

where:

$$\begin{aligned} V &= N \cup T, \\ \mu &= [\]_1 [\]_2 [\]_3 \cdots [\]_k [\]_{k+1} \cdots [\]_l [\]_1, \\ M_1 &= \{XA\}, \text{ for } (S \rightarrow XA) \text{ the initial matrix of } G, \\ M_i &= \{\lambda\}, \text{ for all } i = 2, 3, \dots, l. \end{aligned}$$

The sets of rules are constructed in the following way.

For a matrix $m_i : (X \rightarrow \alpha, A \rightarrow x)$, with $\alpha \in N_1 \cup \{\lambda\}$, $2 \leq i \leq k$, we introduce the rule $X \rightarrow (\alpha, in_i)$ in the set R_1 , and the rule $A \rightarrow (x, out)$ in R_i .

For a matrix $m_i : (X \rightarrow Y, B \rightarrow \#)$, $k+1 \leq i \leq l$, we introduce the rule $X \rightarrow (Y, in_i)$ in R_1 , and the rules $Y \rightarrow (Y, here) || (Y, out)$ and $B \rightarrow B$ in R_i .

We also introduce the rules $a \rightarrow (a, out)$, $a \in T$, in R_1 .

The membranes $2, 3, \dots, k$ are used to simulate the matrices of types 2 and 4, in the obvious way: by using a rule $X \rightarrow (\alpha, in_i)$, for $m_i : (X \rightarrow \alpha, A \rightarrow x)$ in G , the string is sent to membrane i , where the rule $A \rightarrow x$ is simulated and the string is sent back to the skin membrane.

The membranes $k+1, \dots, l$ are used for simulating the matrices of type 3: for a matrix $m_{k+j} : (X \rightarrow Y, B \rightarrow \#)$, we use the rule $X \rightarrow (Y, in_{k+j})$ in the skin membrane, and the string is sent to the corresponding membrane $k+j$, where we use the replication rule $Y \rightarrow (Y, here) || (Y, out)$. A copy of the string exits, one remain in membrane $k+j$; if the symbol B is present, then the rule $B \rightarrow B$ can be used forever, hence the computation will not halt. If the string does not contain the symbol B , then no rule can be used in membrane $k+j$ and we continue in the skin membrane in a correct manner, as after having simulated the matrix m_{k+j} .

The process can be iterated, hence each derivation in G can be simulated in Π .

In any moment, a rule $a \rightarrow (a, out)$ can be used and the string is sent out, but it is accepted in the language $L(\Pi)$ only if it is terminal. Thus, $L(G) = L(\Pi)$. \square

5 Conclusions

We have introduced a variant of P systems with string-objects processed by replicated rewriting rules. We have shown that such systems are able to solve NP-complete problems in linear time; we exemplify this with SAT and HPP.

It is worth noting that our variant uses only rewriting and replication, while the P systems with worm-objects from [1] also use splitting and crossovering operations (but they work with multisets of string-objects, while we use languages, usual sets of strings).

It is an *open problem* whether or not the universality result can be strengthened by proving that P systems with a small number of membranes can characterize *RE*.

References

- [1] J. Castenillos, Gh. Păun, A. Rodriguez-Paton, P systems with worm objects, *IEEE 7th. Intern. Conf. on String Processing and Information Retrieval, SPIRE 2000*, La Coruna, Spain, 64–74.
- [2] J. Dasow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
- [3] J. Dassow, Gh. Păun, On the power of membrane computing, *J. of Universal Computer Sci.*, 5, 2 (1999), 33–49
- [4] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
- [5] Gh. Păun, Computing with membranes – A variant: P systems with polarized membranes, *Intern. J. Foundations of Computer Science*, 11, 1 (2000), 167–182.
- [6] Gh. Păun, P systems with active membranes: Attacking NP-complete problems, *J. Automata, Languages and Combinatorics*, 6, 1 (2001).
- [7] Gh. Păun, G. Rozenberg, A. Salomaa, Membrane computing with external output, *Fundamenta Informaticae*, 41, 3 (2000), 259–266.
- [8] Gh. Păun, Y. Sakakibara, T. Yokomori, P systems on graphs of restricted forms, submitted, 1999.