

Decidability of the termination problem for sequential P systems with active membranes

Michal Kováč

Faculty of Mathematics, Physics and Informatics, Comenius University

Abstract. Abstract

1 Introduction

Membrane systems (P systems) were introduced by Păun (see [1]) as distributed parallel computing devices inspired by the structure and functionality of cells. One of the objectives is to relax the condition of using the rules in a maximally parallel way in order to find more realistic P systems from a biological point of view. In sequential systems, only one rewriting rule is used in each step of computation.

2 Preliminaries

Here we recall several notions from the classical theory of formal languages.

An **alphabet** is a finite nonempty set of symbols. Usually it is denoted by Σ . A **string** over an alphabet is a finite sequence of symbols from alphabet. We denote by Σ^* the set of all strings over an alphabet Σ . By $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ we denote the set of all nonempty strings over Σ . A **language** over the alphabet Σ is any subset of Σ^* .

The number of occurrences of a given symbol $a \in \Sigma$ in the string $w \in \Sigma^*$ is denoted by $|w|_a$. $\Psi_\Sigma(w) = (|w|_{a_1}, |w|_{a_2}, \dots, |w|_{a_n})$ is called a Parikh vector associated with the string $w \in \Sigma^*$, where $\Sigma = \{a_1, a_2, \dots, a_n\}$. For a language $L \subseteq \Sigma^*$, $\Psi_\Sigma(L) = \{\Psi_\Sigma(w) | w \in L\}$ is the Parikh mapping associated with L . If FL is a family of languages, PsFL is denoted the family of Parikh images of languages in FL.

A multiset over a set Σ is a mapping $M : \Sigma \rightarrow \mathbb{N}$. We denote by $M(a)$, $a \in \Sigma$ the multiplicity of a in the multiset M . The **support** of a multiset M is the set $\text{supp}(M) = \{a \in \Sigma | M(a) \geq 1\}$. It is the set of items with at least one occurrence. A multiset is **empty** when its support is empty. A multiset M with finite support $\{a_1, a_2, \dots, a_n\}$ can be represented by the string $a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)}$. We say that multiset M_1 is included in multiset M_2 if $\forall a \in \text{supp}(M_1) : M_1(a) \leq M_2(a)$. We denote it by $M_1 \subseteq M_2$. If $M_1 \subseteq M_2$, the **difference** of two multisets $M_2 - M_1$ is defined as a multiset where $\forall a \in \text{supp}(M_2) : (M_2 - M_1)(a) = \max(M_2(a) - M_1(a), 0)$. The **union** of two multisets $M_1 \cup M_2$ is a multiset where $\forall a \in \text{supp}(M_1) \cup \text{supp}(M_2) : (M_1 \cup M_2)(a) = M_1(a) + M_2(a)$. The product of

multiset M with natural number $n \in \mathbb{N}$ is a multiset where $\forall a \in \text{supp}(M) : (n \cdot M)(a) = n \cdot M(a)$.

Next, we recall notions from graph theory.

A **rooted tree** is a tree, in which a particular node is distinguished from the others and called the root node. Let T be a rooted tree. We will denote its root node by r_T . Let t be a node of $T \setminus \{r_T\}$. The node adjacent to t on the only path from t to r_T is called a **parent node** of t and is denoted by $\text{parent}(t)$. We will denote the set of nodes of T by $V(T)$ and set of its edges by $E(T)$. Let T_1, T_2 be rooted trees. A bijection $f : V(T_1) \rightarrow V(T_2)$ is an **isomorphism** iff $\{(f(u), f(v)) | (u, v) \in E(V(T_1))\} = E(V(T_2))$.

3 Active P systems

The fundamental ingredient of a P system is the **membrane structure** (see [2]). It is a hierarchically arranged set of membranes, all contained in the **skin membrane**. Each membrane determines a compartment, also called region, which is the space delimited from above by it and from below by the membranes placed directly inside, if any exists. Clearly, the correspondence membrane – region is one-to-one, that is why we sometimes use interchangeably these terms. Membrane structure can be also viewed as a rooted tree with the skin membrane as the root node.

Let Σ be a set of objects. Recall that \mathbb{N}^Σ contains all multisets of objects from Σ . **Membrane configuration** is a tuple (T, l, c) , where:

- T is a rooted tree,
- $l \in \mathbb{N}^{V(T)}$ is a mapping that assigns for each node of T a number (label), where $l(r_T) = 1$, so the skin membrane is always labeled with 1,
- $c \in (\mathbb{N}^\Sigma)^{V(T)}$ is a mapping that assigns for each node of T a multiset of objects from Σ , so it represents the contents of the membrane.

Active P system is a tuple $(\Sigma, C_0, R_1, R_2, \dots, R_m)$, where:

- Σ is a set of objects,
- C_0 is initial membrane configuration,
- R_1, R_2, \dots, R_m are finite sets of rewriting rules associated with the labels $1, 2, \dots, m$ and can be of forms:
 - $u \rightarrow w$, where $u \in \Sigma^+$, $w \in (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})^*$ and $1 \leq j \leq m$
 - $u \rightarrow w\delta$, where $u \in \Sigma^+$, $w \in (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})^*$ and $1 \leq j \leq m$
 - $u \rightarrow [{}_j v]_j$, where $u \in \Sigma^+$, $v \in \Sigma^*$ and $1 \leq j \leq m$

Although rewriting rules are defined as strings, u, v and w represent multisets of objects from Σ . For the first two forms, each rewriting rule may specify for each objects on the right side, whether it stays in the current region (we will omit the symbol \cdot), moves through the membrane to the parent region (\uparrow) or to a specific child region (\downarrow_j , where j is a label of a membrane). We denote these transfers with an arrow immediately after the symbol. An example of such rule is the following: $abb \rightarrow ab \downarrow_2 c \uparrow c$. Symbol δ at the end of the rule means that after

the application of the rule, the membrane is dissolved and its contents (objects, child membranes) are propagated to the parent membrane. Active P systems differs from classical (passive) P systems in ability to create new membranes by rules of the third form.

For active P system $(\Sigma, C_0, R_1, R_2, \dots, R_m)$, configuration $C = (T, l, c)$, membrane $d \in V(T)$ with label $j = l(d)$ the rule $r \in R_j$ is **applicable** iff:

- $r = u \rightarrow w$ and $u \subseteq c(d)$ and $\forall (a, \downarrow_k) \in w \exists d_2 \in V(T) : l(d_2) = k \wedge \text{parent}(d_2) = d$,
- $r = u \rightarrow w\delta$ and $u \subseteq c(d)$ and $\forall (a, \downarrow_k) \in w \exists d_2 \in V(T) : l(d_2) = k \wedge \text{parent}(d_2) = d$ and $d \neq r_T$,
- $r = u \rightarrow [{}_kv]_k$ and $u \subseteq c(d)$

Active P system with a limit on total number of membranes is a tuple $(\Sigma, L, C_0, R_1, R_2, \dots, R_m)$, where $(\Sigma, C_0, R_1, R_2, \dots, R_m)$ is an active P system and $L \in \mathbb{N}$ is a limit on total number of membranes. Anytime during the computation, a configuration (T, l, c) is not allowed to have more than L membranes, so the following invariant holds: $|V(T)| \leq L$.

This is achieved by adding a constraint for rule of the form $r = u \rightarrow [{}_kv]_k$, which is defined to be applicable iff $u \subseteq c(d)$ and $|V(T)| < L$. If the number of membranes is equal to L , there is no space for newly created membrane, so in that case such rule is not applicable.

A **computation step** of P system is a relation \Rightarrow on the set of configurations such that $C_1 \Rightarrow C_2$ iff there is an applicable rule in a membrane in C_1 such that applying that rule would result in C_2 .

A **computation** of a P system consists of a sequence of steps. The step S_i is applied to result of previous step S_{i-1} . So when $S_i = (C_j, C_{j+1})$, $S_{i-1} = (C_{j-1}, C_j)$.

There are two possible ways of assigning a result of a computation:

1. By considering the multiplicity of objects present in a designated membrane in a halting configuration. In this case we obtain a vector of natural numbers. We can also represent this vector as a multiset of objects or as Parikh image of a language.
2. By concatenating the symbols which leave the system, in the order they are sent out of the skin membrane (if several symbols are expelled at the same time, then any ordering of them is considered). In this case we generate a language.

The result of a computation is clearly only one multiset or a string, but for one initial configuration there can be multiple possible computations. It follows from the fact that there exist more than one maximal multiset of rules that can be applied in each step.

3.1 Register machines

As a referential universal language acceptor we will use Minsky's register machine. Such a machine runs a program consisting of numbered instructions of several simple types.

Definition 1. A *n*-register machine is a tuple $M = (n, P, i, h)$, where:

- n is the number of registers,
- P is a set of labeled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of M , and j, k, l are labels from the set $Lab(M)$ (which numbers the instructions in a one-to-one manner),
- i is the initial label, and
- h is the final label.

The machine is capable of the following instructions:

- $(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l ; in the deterministic variants usually considered in the literature we demand $k = l$.
- $(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k , otherwise proceed to instruction l .
- $halt$: This instruction stops the machine. This additional instruction can only be assigned to the final label h .

A deterministic m -register machine can analyze an input $(n_1, \dots, n_m) \in N_0^m$ in registers 1 to m , which is recognized if the register machine finally stops by the halt instruction with all its registers being empty (this last requirement is not necessary). If the machine does not halt, the analysis was not successful.

4 Termination problems

In this section we recall the halting problem for Turing machines. The problem is to determine, given a deterministic Turing machine and an input, whether the machine running on that input will halt. It is one of the first known undecidable problems[citation needed]. On the other hand, for non-deterministic machines, there are two possible meanings for halting. We could be interested either in:

- whether there exists a finite computation (the machine can halt), or
- whether there exists an infinite computation (the machine can run forever)

For universal models the existence of finite computation is undecidable, because otherwise we could decide emptiness of language generated by any Turing machine[citation needed].

Theorem 1. *Active P systems with limit on total number of membranes are universal.*

Proof. The proof for active P system in [citation needed] uses simulation of register machines and during the simulation, every configuration has at most three membranes.

Theorem 2. *Active P systems with limit on total number of membranes have decidable existence of infinite computation.*

Proof. We will propose an algorithm for deciding existence of infinite computation. Basic idea is to consider reachability graph, where nodes are configurations and edges lead from the configuration C_1 to the configuration C_2 , when there is a rule applicable in C_1 , which results in C_2 . Such reachability graph can be infinite. In order to use it for deciding some properties, we need to reduce the number of nodes in the graph to a finite number.

For single membrane P systems it can be done in the same way as for Petri nets [citation needed]. For active membrane systems we define a partial order on configurations as follows:

A configuration C_1 covers configuration C_2 , iff

From Dickson's lemma [citation needed] or from Karp-Miller reachability theorem [citation needed] implies that for a every infinite sequence of configurations there are two such that one covers the other.

5 Conclusion

We have studied . . .

References

1. Păun, G.: Computing with membranes. *Journal of Computer and System Sciences* **61**(1) (2000) 108 – 143
2. Păun, G.: Introduction to membrane computing. In Ciobanu, G., Păun, G., Pérez-Jiménez, M., eds.: *Applications of Membrane Computing*. Natural Computing Series. Springer Berlin Heidelberg (2006) 1–42