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 $|w|_a$. $\Psi_\Sigma(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$ is called a Parikh vector associated with the string $w \in \Sigma^*$, where $\Sigma = \{a_1, a_2, \ldots a_n\}$. 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 $M : \Sigma \to \mathbb{N}$. 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 $supp(M) = \{a \in \Sigma | M(a) \geq 1\}$. 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 $\{a_1, a_2, \ldots, a_n\}$ can be represented by the string $a_1^{M(a_1)} a_2^{M(a_2)} \ldots a_n^{M(a_n)}$. We say that multiset M_1 is included in multiset M_2 if $\forall a \in supp(M_1) : M_1(a) \leq M_2(a)$. 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 $\forall a \in supp(M_2) : (M_2 - M_1)(a) = \max(M_2(a) - M_1(a), 0)$. The **union** of two multisets $M_1 \cup M_2$ is a multiset where $\forall a \in supp(M_1) \cup supp(M_2) : (M_1 \cup M_2)(a) = M_1(a) + M_2(a)$. The product of

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

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 d be a node of $T \setminus \{r_T\}$. The node adjacent to d on the only path from d to r_T is called a **parent node** of d and is denoted by $parent_T(d)$. 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 $f : V(T_1) \to V(T_2)$ is an **isomorphism** iff $\{(f(u), f(v))|(u, v) \in E(V(T_1))\} = E(V(T_2))$ and $f(r_{T_1}) = r_{T_2}$.

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 $\mathbb{N}^\Sigma$ 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 $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, where:

- Σ is a set of objects,
- C_0 is initial membrane configuration,
- $R_1, R_2, \ldots R_m$ are finite sets of rewriting rules associated with the labels $1, 2, \ldots, m$ and can be of forms:
  - $u \to w$, where $u \in \Sigma^+$, $w \in (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})^*$ and $1 \leq j \leq m$
  - $u \to w\delta$, where $u \in \Sigma^+$, $w \in (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})^*$ and $1 \leq j \leq m$
  - $u \to [_j v]_j$, 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: $abb \to ab \downarrow_2 c \uparrow c$. 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 $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, configuration $C = (T, l, c)$, membrane $d \in V(T)$ with label $j = l(d)$ the rule $r \in R_j$ is **applicable** iff:

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

Active P system with a limit on total number of membranes is a tuple $(\Sigma, L, C_0, R_1, R_2, \ldots, R_m)$, where $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$ is an active P system and $L \in \mathbb{N}$ is a limit on total number of membranes. Anytime during the computation, a confuguration (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 $r = u \to [_k v]_k$, 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 $(n_1, \ldots, n_m) \in N_0^m$ 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. On the other hand, for non-deterministic machines, there are two possible meanings for halting. We could be interested either in:

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

We will prove the (un)decidability of these problems on active P systems with limit on total number of membranes. The results are quite interesting, because:

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

Proof. The proof of this theorem for sequential active P systems in [3] uses simulation of register machines and during the simulation, every configuration has at most three membranes, hence the universality holds also for sequential active P systems with limit on total number of membranes. □

4.1 Existence of infinite computation

We will propose an algorithm for deciding existence of infinite computation. Basic idea is to consider the minimal coverability graph ([4]), where nodes are configurations and edge leads from the configuration C_1 to the configuration C_2, whenever there is a rule applicable in C_1, which results in C_2. The construction in [4] is performed on Petri nets, where the configuration consists just of a vector of natural numbers. The situation is the same for single-membrane sequential P systems. We need to modify the construction for active P systems.

A configuration $C_2 = (T_2, l_2, c_2)$ **covers** configuration $C_1 = (T_1, l_1, c_1)$ iff $\exists$ isomorphism $f : T_1 \to T_2$ such that $\forall d \in T_1$ the following properties hold: $l_1(d) = l_2(f(d)) \wedge c_1(d) \subseteq c_2(f(d))$.

We will denote this with $C_1 \leq C_2$.

Lemma 1. *For sequential active P system with limit on total number of membranes, if* $C_2 = (T_2, l_2, c_2)$ **covers** *configuration* $C_1 = (T_1, l_1, c_1)$*, then there is an isomorphism* $f : T_1 \to T_2$ *such that if a rule* r *is applicable in membrane* $d \in T_1$*, then* r *is applicable in* $f(d)$*.*

Proof. Suppose r is applicable in d. Then left side of the rule is contained within the contents of the membrane $u \subseteq c_1(d)$. Because $C_1 \leq C_2$, then $c_1(d) \subseteq c_2(f(d))$ and then $u \subseteq c_2(f(d))$.

There are three possible forms of the rule r.

- If $r = u \to w$, then because r is applicable in d, $\forall (a, \downarrow_k) \in w \exists d_2 \in V(T_1) : l_1(d_2) = k \wedge parent_{T_1}(d_2) = d$. Because $C_1 \leq C_2$, then for $f(d_2) \in V(T_2)$ the following holds: $l_2(f(d_2)) = l_1(d_2) = k$ and $parent_{T_2}(f(d_2) = f(d)$. Hence r is applicable in $f(d)$.
- If $r = u \to w\delta$, then $d \neq r_{T_1}$. Since f is an isomorphism, then also $f(d) \neq r_{T_2}$. Other properties follows from the previous case.
- If $r = u \to [_k v]_k$, then $|V(T_1)| < L$. Isomorphism preserves number of nodes, hence $|V(T_2)| = |V(T_1)| < L$ and r is applicable in $f(d)$. □

Now, we will define the encoding of a configuration $C = (T, l, c)$ into a tuple of integers.

A membrane $d \in T$ will be encoded as $n + m$-tuple $enc(d) \in \mathbb{N}^{(n+m)}$, where first n numbers will be actual counts of objects: $enc(d)_i = c(d)(a_i)$ for $i \leq n$. Next m numbers will encode the membrane label: $enc(d)_{n+j} = 0$ for $l(d) \neq j \leq m$ and $enc(d)_{n+l(d)} = 1$.

The entire tree will be encoded into concatenated sequences of encoded nodes in the inorder traversal order. This sequence is then padded with zeroes to have length $(n + m)L$ as that is the maximal length of encoded tree.

Since there are only finitely many non-isomorphic trees with at most L nodes ([5]), there is a constant z such that we can uniquely assign the tree an order number $o(T) \leq z$.

The configuration will be encoded in tuple which consists of z parts. All but the part with index $o(T)$ will contain just zeros. The part with index $o(T)$ will contain the encoding of the tree.

Lemma 2. *For configurations* $C_1 = (T_1, l_1, c_1)$ *and* $C_2 = (T_2, l_2, c_2)$, $enc(C_1) \leq enc(C_2) \Rightarrow C_1 \leq C_2$.

Proof. If the $enc(C_1) \leq enc(C_2)$, then $o(T_1) = o(T_2)$, so the trees are isomorphic. For every membrane $d \in T_1$, $l_1(d) = l_2(f(d))$ and $c_1(d) \subseteq c_2(f(d))$. Hence, $C_1 \leq C_2$. □

Lemma 3. *For sequential active P system with limit on total number of membranes L for every inifite sequence of configurations* $\{C_i\}_{i=0}^{\infty} \exists i < j : C_i \leq C_j$.

Proof. Suppose an infinite sequence $\{enc(C_i)\}_{i=0}^{\infty}$. We use a variation of Dickson's lemma ([6]): Every infinite sequence of tuples from $\mathbb{N}^k$ contains an increasing pair. Applied to our sequence, there are two positions $i < j : enc(C_i) \leq enc(C_j)$. From lemma 2, $C_i \leq C_2$. □

Theorem 2. *Existence of infinite computation for active P systems with limit on total number of membranes is decidable.*

Proof. Algorithm for deciding the problem will traverse the reachability graph. When it encounters a configuration that covers another configuration, from lemma 1 follows that the same rules can be applied repeatedly, so the algorithm will halt with the answer YES. Otherwise, the algorithm will answer NO. Algorithm will always halt, because if there was an infinite computation, from lemma 3 there would be two increasing configurations which is already covered in the YES case. □

4.2 Existence of finite computation

Theorem 3. *Existence of finite computation for active P systems with limit on total number of membranes is undecidable.*

Proof. Either reduce it to detection of empty language or to reachability.

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
3. Ibarra, O., Woodworth, S., Yen, H.C., Dang, Z.: On sequential and 1-deterministic p systems. In Wang, L., ed.: Computing and Combinatorics. Volume 3595 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 905–914

4. Finkel, A.: The minimal coverability graph for petri nets. In Rozenberg, G., ed.: Advances in Petri Nets 1993. Volume 674 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1993) 210–243
5. Cayley, P.: On the analytical forms called trees. American Journal of Mathematics **4**(1) (1881) pp. 266–268
6. Figueira, D., Figueira, S., Schmitz, S., Schnoebelen, P.: Ackermannian and primitive-recursive bounds with dickson's lemma. In: Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science. LICS '11, Washington, DC, USA, IEEE Computer Society (2011) 269–278