

A Short Note on Analysing P Systems with Antiport Rules

Rudolf FREUND and Marion OSWALD
Department of Computer Science
Technische Universität Wien, Wien, Austria
rudi@emcc.at, marion@emcc.at

Abstract

We investigate a variant of purely communicating P systems which are able to analyse multisets or even strings given as sequences of terminal symbols taken from the environment. We show that analysing P systems equipped with only one membrane and antiport rules of radius two only can already recognize any recursively enumerable language of multisets and strings, respectively.

1 Introduction

P systems were introduced in [11] by Gh. Păun as distributed parallel computing devices that are abstracted from cell functioning. The most important features considered in the various models of P systems investigated so far (e.g., see [3], [11], [12]; for a comprehensive overview see [13]; for the actual status of P systems research see [9]) are the membrane structure and specific features of the membranes, especially for the transfer of objects through the membranes. A *membrane structure* consists of membranes hierarchically embedded in the outermost *skin membrane*; every membrane encloses a *region* possibly containing other membranes. In the membranes, multisets of objects can be placed, which evolve according to given evolution rules. Applying the latter ones in a nondeterministic, maximally parallel way, the system passes from one configuration to another one, thereby performing a computation; only halting computations produce a result.

In contrast to various other models of P systems, where the objects themselves can be transformed during a computation, we consider purely communicating systems (as already done, e.g., in [10]), and, moreover, we use these systems for analysing an input sequence of terminal symbols (for a first variant of P automata see [1]).

In the following section we first give some preliminary definitions and define n -register machines, the universal model of computation we use for proving our new results elaborated in this paper; in the third section we introduce analysing P systems with antiport rules. In the fourth section we show that analysing P systems with only one membrane and antiport rules of radius two only can already recognize any recursively enumerable language of multisets and strings, respectively; the proof is based on the fact that P systems with antiport rules quite easily can simulate n -register machines, which result was already established, independently, both in [5] as well as in [7].

2 Preliminary definitions

The set of non-negative integers is denoted by \mathbf{N}_0 , the set of positive integers by \mathbf{N} . An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V , the free monoid generated by V under the operation of concatenation is denoted by V^* ; moreover, we define $V^+ := V^* \setminus \{\lambda\}$, where λ denotes the empty word. A multiset over V is represented as a string over V (and any of its permutations). By $|x|$ we denote the length of the word x over V as well as the number of elements in the multiset represented by x .

For more notions from the theory of formal languages, the reader is referred to [2].

When considering multisets of symbols, a simple universal computational model are register machines (see [8] for some original definitions and [4], [15] for definitions like that we use in this paper).

An n -register machine is a construct $RM = (n, R, i, h)$ where

- n is the number of registers,
- R is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of RM , j, k, l are labels from the set $Lab(RM)$ (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:

(A(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 $k = l$.

(S(r),k,l) If register r is not empty then subtract one from its contents and go to the instruction k , otherwise proceed to instruction l .

HALT Stop the machine. This additional instruction can only be assigned to the final label h .

In their *deterministic variant*, such n -register machines can be used to compute any partial recursive function $f : \mathbf{N}_0^k \rightarrow \mathbf{N}_0^m$; starting with $(n_1, \dots, n_k) \in \mathbf{N}_0$ in registers 1 to k , RM has computed $f(n) = (r_1, \dots, r_m)$ if it halts in the final label h with registers 1 to m containing r_1 to r_m . If the final label cannot be reached, $f(n)$ remains undefined.

A deterministic n -register machine can also analyse an input $(n_1, \dots, n_k) \in \mathbf{N}_0^k$ in registers 1 to k , which is recognized if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, the analysis was not successful.

In their *non-deterministic variant*, n -register machines can compute any recursively enumerable set of natural numbers (or of vectors of natural numbers). Starting with all registers being empty, we consider a computation of the n -register machine to be successful, if it halts with the result being contained in the first (m) register(s) and with all other registers being empty.

From the results proved in [4] (based on the results established in [8]) we immediately conclude the following result:

Proposition 1 *For any recursively enumerable set of vectors of natural numbers $L \subseteq \mathbf{N}_0^k$ there exists a deterministic $(k + 2)$ -register machine M recognizing L .*

Moreover, for sets of strings we have a similar result (also see [6]):

Proposition 2 For any recursively enumerable set of strings L over the alphabet T with $\text{card}(T) = z - 1$ there exists a deterministic 3-register machine M recognizing L in such a way that, for every $w \in T^*$, $w \in L$ if and only if M halts when started with $g_z(w)$ in its first register, where $g_z(w)$ is the z -ary representation of the word w .

3 Analysing P systems with antiport rules

An *analysing P system with antiport rules* is a construct Π of the following form:

$$\Pi = (V, T, \mu, w_1, \dots, w_n, R_1, \dots, R_n)$$

where

- V is an alphabet of *objects*;
- $T \subseteq V$ is the terminal alphabet;
- μ is a *membrane structure* (with the membranes labelled by natural numbers $1, \dots, n$ in a one-to-one manner);
- w_1, \dots, w_n are multisets over V associated with the regions $1, \dots, n$ of μ ;
- R_1, \dots, R_n are finite sets of *antiport rules* associated with the regions/membranes $1, \dots, n$; an antiport rule is of the form $(x, \text{out}; y, \text{in})$, where $x, y \in V^+$, which means that the multiset x is sent out of the membrane and y is taken into the membrane region from the surrounding region. The *radius* of the antiport rule $(x, \text{out}; y, \text{in})$ is defined as $\max\{|x|, |y|\}$.

Starting from the *initial configuration*, which consists of μ and w_1, \dots, w_n , the system passes from one configuration to another one by nondeterministically in a maximally parallel way applying rules from R_i . A sequence of transitions is called a *computation*; it is *successful*, if and only if it halts. A multiset or a string w over an alphabet T is recognized by the analysing P system Π if and only if there is a successful computation of Π such that the (sequence of) terminal symbols taken from the environment is exactly w . (If more than one terminal symbol is taken from the environment in one step then any permutation of these symbols constitutes a valid subword of the input string.)

4 Results

Based on the proof techniques used, e.g., in [4], [6], we can immediately show the following results using Proposition 2.

Theorem 3 *Let $L \subseteq T^*$ be a recursively enumerable set. Then L can be recognized by a P system with antiport rules in only one membrane using antiport rules of the forms $(x, out; y, in)$ with $(|x|, |y|) \in \{(1, 2), (2, 1)\}$ only (i.e., we only need rules of radius 2).*

Proof (sketch). According to Proposition 2, we only have to elaborate how we can read the input string w , generate the encoding $g_z(w)$ and then how to simulate the instructions of a 3-register machine; in fact, the main emphasis lies on the simulation of an n -register machine:

- An Add-instruction $j : (A(i), k, l)$ is simulated by the rules $(j, out; ka_i, in)$ and $(j, out; la_i, in)$.
- A conditional Subtract-instruction $j : (S(i), k, l)$ is simulated by the following rules:
 - $(ja_i, out; k, in)$
 - $(j, out; j'j'', in)$
 - $(j'a_i, out; f, in)$ $(f, out; f'f'', in)$ and $(f'f'', out; f, in)$
 - $(j'', out; j''', in)$
 - $(j'j''', out; l, in)$

The condition of maximal parallelism guarantees that the rule $(j'a_i, out; f, in)$ is applied in parallel with $(j'', out; j''', in)$, which leads to a non-halting computation by the introduction of the failure symbol (trap symbol) f . Only if in the current configuration no symbol a_i is present in the skin membrane, the object j' can wait one step for being used in the rule $(j'j''', out; l, in)$ together with the symbol j''' introduced by the rule $(j'', out; j''', in)$.

- The halting instruction $h : HALT$ is simulated by just doing nothing with the halting symbol h anymore.

Now let us start with the singleton q in the initial configuration. For every $a \in T$ we take $(q, out; qa, in)$. Let us assume we have represented the encoding of the input sequence v taken in so far by $g_z(v)$ symbols A . The

encoding of $g_z(va)$ obviously is given by $z * g_z(v) + g_z(a)$. This encoding step is accomplished by the following subprogram of a register machine; its first part represents the multiplication by z :

$$\begin{aligned} q_a &: (S(1), q_{a,1}, q'_a) \\ q_{a,i} &: (A(2), q_{a,i+1}, q_{a,i+1}) \text{ for } 1 \leq i < z \\ q_{a,z} &: (A(2), q_a, q_a) \\ q'_a &: (S(2), q'_{a,1}, q''_{a,1}) \\ q'_{a,1} &: (A(1), q'_a, q'_a) \end{aligned}$$

Now let $k = g_z(a)$; then we finish with the following instruction:

$$q''_{a,i} : (A(1), q''_{a,i+1}, q''_{a,i+1}) \text{ for } 1 \leq i \leq k - 1$$

The input of the next terminal symbol starts with the antiport rule $(q''_{a,k}a, out; q, in)$.

Obviously, the instructions of the subprogram above can be translated into antiport rules as already elaborated at the beginning of the proof. The numbers of symbols A and B , respectively, correspond with the contents of registers 1 and 2, respectively.

If no further input symbols should be taken in, we use the following antiport rules to start the simulation of the 3-register machine indicated in Proposition 2:

$$\begin{aligned} (q, out; q'q'', in) \\ (q'q'', out; q_0, in) \end{aligned}$$

where q_0 corresponds with the initial label of the register machine.

Obviously, the halting symbol h (representing the halting instruction $h : HALT$) appears in the skin membrane of the analysing P system if and only if the register machine accepts the input $g_z(w)$. \square

Observe that, in contrast to P systems with antiport rules as defined in [10], we need not specify the environment, because we assume every symbol to appear in an unlimited number there.

The string to be recognized is given by the sequence of terminal symbols a taken from the environment by antiport rules of the form $(q, out; q_a a, in)$. Obviously, this string can also be interpreted as a representation of the corresponding multiset (or the corresponding vector of natural numbers, respectively), which establishes results similar to Theorem 3 for recursively enumerable multisets over T (and the corresponding sets of vectors of natural numbers, respectively).

5 Conclusion

We have investigated analysing P systems with antiport rules which surprisingly already obtain their maximal recognizing power with the simplest membrane structure and rules with radius two only. According to the features of the 3-register machine constructed in Proposition 2 a successful computation of an analysing P system recognizing the string w ends up in a final configuration with only the halting symbol h in the skin membrane, which in some sense corresponds with the situation of an automaton accepting by a final state (also compare with the definition of acceptance by P automata as defined in [1]). On the other hand, we could also “accept by empty membrane” using the symport rule (h, out) (for the definition of a symport rule see [10]).

Acknowledgements

We should like to thank Gheorghe Păun for all the fruitful discussions during and after the Workshop on Membrane Computing (WMC-CdeA2002) taking place under the auspices of the European Molecular Computing Consortium - MolCoNet project IST-2001-32008 - at Curtea de Argeş, Romania, in August 2002 that led to the ideas of analysing P systems as they are presented in this paper.

References

- [1] E. Csuhaj-Varjú and G. Vaszil, P automata, in [14], 177–192.
- [2] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin (1989).
- [3] J. Dassow and Gh. Păun, On the power of membrane computing, *Journal of Universal Computer Science* **5**, 2 (1999), 33–49 (<http://www.iicm.edu/jucs>).
- [4] R. Freund and M. Oswald, Generalized P systems with forbidding context, *Fundamenta Informaticae* **49**, 1-3 (2002), 81–102.

- [5] R. Freund and M. Oswald, P systems with activated/prohibited membrane channels, presented at WMC-CdeA2002, Curtea de Argeş, Romania (2002).
- [6] R. Freund and Gh. Păun, On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars, *Proc. Conf. Universal Machines and Computations*, Chişinău, 2001 (M. Margenstern and Y. Rogozhin, eds.), Springer-Verlag, Berlin (2001).
- [7] P. Frisco and H. J. Hoogeboom, Simulating counter automata by P systems with symport/antiport, in [14], 237–248.
- [8] M. L. Minsky, *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, USA (1967).
- [9] The P Systems Web Page, <http://psystems.disco.unimib.it>.
- [10] A. Păun and Gh. Păun, The Power of Communication: P Systems with Symport/Antiport, *New Generation Computing*, **20**, 3 (2002), 295–306.
- [11] Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences* **61**, 1 (2000), 108–143.
- [12] Gh. Păun, Computing with Membranes: An Introduction, *Bulletin EATCS* **67** (1999), 139–152.
- [13] Gh. Păun, *Membrane Computing - An Introduction*, Springer-Verlag, Berlin (2002).
- [14] Gh. Păun and C. Zandron (eds.), *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002)*, Curtea de Argeş, Romania (2002).
- [15] P. Sosik, P systems versus register machines: two universality proofs, in [14], 371–382.