DISSERTATION

P Automata

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

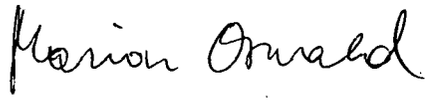
Ao. Univ.-Prof. Mag.rer.nat. Dipl.-Ing. Dr.techn. Rudolf Freund
E 185/2
Institut für Computersprachen

eingereicht an der technischen Universität Wien
Fakultät für technische Naturwissenschaften und Informatik

von

Dipl.-Ing. Marion Oswald
9003484
1040 Wien, Große Neugasse 44/11

Wien, am 23.11.2003



Kurzfassung

P Systeme sind unkonventionelle Berechnungsmodelle, welche von der Funktionsweise biologischer Zellen abstrahiert sind. Dabei werden die fundamentalen Eigenschaften von lebenden Zellen im theoretischen Modell erfasst: in einer Membranstruktur, wo die Membranen als Separatoren sowie Kommunikationskanäle fungieren, entwickelt sich eine Vielzahl von Objekten gemäß vorgegebener Evolutionsregeln. In der Basisklasse von P Systemen findet diese Entwicklung in einer non-deterministischen, maximal parallelen Weise statt. Seit 1998 wurden verschiedene Varianten dieses Modells untersucht und deren computationale Vollständigkeit bewiesen. Hier befassen wir uns jedoch nur mit akzeptierenden Varianten, welche 2002 unter dem Namen P Automaten bekannt wurden.

Nach formalen Definitionen und einem kurzen Literaturüberlick werden in dieser Arbeit verschiedene Varianten im Hinblick auf deren Akzeptierungsmächtigkeit erforscht. Wir beginnen mit rein kommunizierenden Systemen, bei welchen die Objekte nicht verändert werden, sondern nur die Membranen passieren, sogenannten analysierenden P Systemen mit Antiport Regeln, P Automaten mit Membrankanälen sowie akzeptierenden Varianten von P Systemen mit an die Membran gebundenen bedingten Kommunikationsregeln.

Darauf folgend werden katalytische sowie rein katalytische Systeme präsentiert. Von Interesse ist auch die Akzeptierung unendlicher Wörter, welche hier mittels P Automaten mit Membrankanälen untersucht wird. Schließlich wird der neue Begriff des $k-$Determinismus vorgestellt, welcher effizientere Simulationen von verschiedensten P Automaten auf herkömmlichen Computern ermöglichen soll. Eine kurze Zusammenfassung und Diskussion offener Probleme in den untersuchten Gebieten beschließen die vorliegende Arbeit.

Abstract

P systems are unconventional models of computation that are abstracted from cell functioning, capturing the fundamental properties of alive cells: In a membrane structure, where the membranes act as separators as well as communication channels, multisets of objects evolve according to prescribed evolution rules. In the basic class of P systems, the evolution takes place in a maximally parallel, non-deterministic way. Since their introduction in 1998, many variants have been investigated most of which have been proved to be computationally universal. Yet we here consider only the accepting variants, introduced as P automata in 2002.

After some formal definitions and a brief literature review, we investigate several variants of such P automata with respect to their recognizing power. We start by considering three purely communicating systems, where the objects are only moved across the membranes without being affected by the use of the rules: Analysing P systems with antiport rules, P automata with membrane channels and an accepting variant of P systems with conditional communication rules assigned to membranes.

We then shortly leave the area of purely communicating systems and present catalytic as well as purely catalytic variants of accepting P systems. Computations on infinite words by means of P automata with membrane channels are the subject of investigations in the sequel. Finally we introduce the new notion of $k-$determinism allowing for more efficient simulations of various kinds of accepting P systems on conventional computers. Some final remarks and open problems conclude this work.

Contents

1 Introduction **1**

2 Preliminary Definitions **3**
2.1 Formal Language Prerequisites 3
2.2 Register Machines 5
2.2.1 Definition 5
2.2.2 Results 6

3 P Automata - a Brief Literature Review **10**
3.1 The Original Model, a Variant and Improvements 11
3.2 Inducing an Infinite Hierarchy 12
3.3 Some Further Remarks 14

4 Analysing P Systems with Antiport Rules **15**
4.1 Definitions 16
4.2 Results 17
4.3 Conclusion 19

5 P Automata with Membrane Channels **20**
5.1 Definitions 21
5.2 Results 22
5.3 (Finite) P automata with Antiport Rules 26
5.4 Conclusion 29

6 P Automata with Conditional Communication Rules Assigned to Membranes **30**
6.1 PACCRAM - Definition 31
6.2 Results 33
6.3 Conclusion 36

7 Accepting P systems / P Automata with Catalysts **37**
7.1 Definitions 38
7.2 Register Machines and Counter Automata 38
7.3 The Standard Model of P Systems and Variants 40
7.4 Results 44
7.5 Conclusion 52

8 ω-P Automata with Communication Rules **53**
8.1 Preliminary Definitions 54
8.2 ω-Turing Machines 54
8.2.1 Variants of Acceptance 54
8.2.2 ω-Turing Machines - Definitions 55
8.2.3 ω-languages Accepted by ω-Turing Machines 56
8.2.4 Finite ω-automata 57
8.3 ω-P Automata 57
8.3.1 ω-P Automata with Antiport Rules 58
8.3.2 Finite ω-P Automata 61
8.4 Conclusion 63

9 On "Weak" Determinism in P Automata **64**
9.1 Preliminary Definition and Example 65
9.2 $k-$Determinism 66
9.3 Results 67
9.4 Conclusion 75

10 Final Remarks **76**

Acknowledgements **78**

Bibliography **78**

Curriculum Vitae **84**

Chapter 1

Introduction

In 1998, P systems were introduced by Gheorghe Păun [48] as a new model of unconventional computation inspired by biochemical reactions that take place in living cells.

Cells are the basic units for the structure and function of all organisms. They can be divided by membranes into a system of interconnected cavities and separate compartments in which chemical reactions take place. But membranes not only compartmentalize cells and regulate traffic between compartments, their functions also involve energy transformation and information processing, often interpreted as computing processes (see, e.g., [7] and the references there).

P systems are computing devices abstracted from cell functioning and are based on the notion of a membrane structure. A *membrane structure* consists of membranes hierarchically embedded in the outermost *skin membrane*; every membrane encloses a *region* possibly containing other membranes; the region outside the skin membrane is called *outer region* or *environment*. In the membranes, multisets of objects can be placed, which evolve according to given evolution rules. Depending on the model, these rules can be applied in parallel across all membranes or in a rather sequential manner. In any way, the rules to be applied are non-deterministically chosen by the system, hence if an object can evolve according to more than one evolution rule at the same time, any one is chosen.

A *configuration* can be illustrated by putting the objects and rules in the corresponding compartments of the membrane structure. In this way, a computing device is obtained in the following sense: Starting from an *initial configuration*, the system evolves by passing from one configuration to another one, thereby performing a *computation*. If the system halts, i.e., no rule can be applied anymore, the computation is called *successful*. Instead of going into more details here, we refer to [51] for motivations and examples

(also see [44]).

On the other hand, P systems can also be used as accepting devices. The notion of P Automata was introduced by Erzsébet Csuhaj-Varjú and Györgi Vaszil in [14] as purely communicating accepting P system with one-way communication, which will be shortly presented in a brief literature review in Chapter 3.

What follows is an investigation of various models of such accepting devices starting with some purely communicating variants: In Chapter 4 we present the work from [28] on analysing P systems with antiport rules, Chapter 5 is about P automata with membrane channels, that partly appeared in [46] and [33]. In Chapter 6 we investigate the accepting variant of P systems with conditional communication rules assigned to membranes as they were introduced in [29]. In Chapter 7 we shortly leave the area of purely communicating systems and present parts of the work from [31], dealing with (purely) catalytic variants of accepting P systems. We go back to P automata with membrane channels in Chapter 8, but this time, far away from biological motivation, considering computations on infinite words. We there present results obtained in [33]. In Chapter 9 we introduce the new notion of $k-$determinism allowing for more efficient simulations of various kinds of accepting P systems. Some final remarks and acknowledgements conclude this work.

But before starting with notions and notations from formal language theory, also presenting some results on register machines that have appeared in [26], we should like to remark the following: Other than for generating P systems of whatever type, P automata sometimes also appear under the names of accepting or analysing P systems in the literature. Therefore we here use the original terms under which the respective systems have been introduced, except for the accepting variant of P systems with conditional communication rules assigned to membranes, which will be called P automata with conditional communication rules in the following.

Chapter 2

Preliminary Definitions

In this Chapter, we first recall some general definitions from formal language theory. For more details on that subject, we refer to [18], [56], [57]. We then define register machines, a universal model of computation that will extensively be used in most proofs in subsequent Chapters. We conclude by giving some results on register machines. Further definitions that are only of local interest will be introduced when necessary.

2.1 Formal Language Prerequisites

The set of non-negative integers is denoted 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^*; the *empty string* is denoted by λ. The elements of V^* are called *words* or *strings*. The set of non empty strings over V, that is $V^* \setminus \{\lambda\}$, is denoted by V^+. Any subset of V^+ is called a *λ-free (string) language*.

A *multiset* (over an arbitrary set B) is a function $M : B \to \mathbf{N} \cup \{\infty\}$; $M(x)$ is the number of occurrences of $x \in B$ in the multiset M. The set $supp(M) = \{x \in B \mid M(x) > 0\}$ is called the *support of* M. Hence a usual set $S \subseteq B$ can be seen as a multiset where $M(x) = 1$ for $x \in S$, and $M(x) = 0$ otherwise. For two multisets $M_1, M_2 : B \to \mathbf{N}$, their union is defined by $(M_1 + M_2)(x) = M_1(x) + M_2(x)$, $x \in B$. Provided that $M_1(x) \geq M_2(x)$ for all $x \in B$, their difference is given by $(M_1 - M_2)(x) = M_1(x) - M_2(x)$. A multiset M with finite support is represented by a set of pairs $\langle x, M(x) \rangle$, for $x \in supp(M)$. An *empty multiset*, i.e., a multiset M with empty support, is denoted by $\emptyset$.

Finite multisets over B can also be represented as strings over B (e.g., the multiset of five objects a_1 and two objects a_2 is denoted by $\langle (a_1, 5) + (a_2, 2) \rangle$

and can be represented by strings like $a_1^5a_2^2$, $a_1^4a_2^2a_1$, and any of their permutations) or by the corresponding Parikh vectors (e.g., the multiset of five objects a_1 and two objects a_2 can be represented by the Parikh vector $(5,2)$). A subset M of the set of non-negative integers $\mathbf{N}$ can be represented as the formal language $L = \{a_1^k \mid k \in M\}$; by $\mid x \mid$ we denote the length of the word x over V as well as the number of elements in the multiset represented by x.

A *(string) grammar* is a quadruple $G = (V_N, V_T, P, A)$, where V_N and V_T are finite sets of *nonterminal* and *terminal symbols*, and $V_N \cap V_T = \emptyset$, P is a finite set of *productions* $\alpha \to \beta$ with $\alpha \in V^+$and $\beta \in V^*$, where $V = V^+ \cup V^*$, and $A \in V_N$ is the *axiom*. For $x, y \in V^*$ we say that y *is directly derivable* from x in G, denoted by $x \Rightarrow_G y$, if and only if for some $\alpha \to \beta$ in P and $u, v \in V^*$ we get $x = u\alpha v$ and $y = u\beta v$. Denoting the reflexive and transitive closure of the derivation relation $\Rightarrow_G$ by $\Rightarrow_G^*$, the *language generated by* G is $L(G) = \{w \in V_T^* \mid A \Rightarrow_G^* w\}$. A production $\alpha \to \beta$ is called *context-free* if $\alpha \in V_N$. If G contains only context-free rules it is called a *context-free grammar*. The family of recursively enumerable languages is denoted by RE.

We will also use the following more general notion of a grammar:

A *grammar* is a quadruple $G = (B, B_T, P, A)$, where B and B_T are sets of *objects* and *terminal objects*, respectively, with $B_T \subseteq B$, P is a finite set of *productions*, and $A \in B$ is the *axiom*. A production p in P in general is a partial recursive relation $\subseteq B \times B$, where we also demand that the domain of p is recursive (i.e., given $w \in B$ it is decidable if there exists some $v \in B$ with $(w, v) \in P$) and, moreover, that the range for every w is finite, i.e., for any $w \in B$, $card(\{v \in B \mid (w, v) \in p\}) < \infty$. As for string grammars above, the productions in P induce a derivation relation $\Rightarrow_B$ on the objects in B etc. The *language generated by* G is $L(G) = \{w \in B_T \mid A \Rightarrow_G^* w\}$.

For example, a string grammar (V_N, V_T, P, A) in this general notion is now written as $((V_N \cup V_T)^*, V_T^*, P, A)$.

A *finite automaton* (*FA* for short) is a quintuple $M = (Q, T_M, \delta, q_0, F)$ where Q is the finite set of *states*, T is the *input alphabet*, $\delta : Q \times T \to 2^Q$ is the *state transition function*, $q_0 \in Q$ is the *starting state* and $F \subseteq Q$ is the set of *final states*. A finite automaton is called *deterministic* if $card\,(\delta\,(q, a)) = 1$ for all $q \in Q$ and $a \in T$. The transition function δ can be extended in a natural way to a function $\delta : Q \times T^+ \to Q$. The language accepted by the DFA M is the set of all strings $w \in T^+$ that are accepted by M in such a way that $\delta\,(q_0, w) \in F$.

2.2 Register Machines

When considering multisets of symbols, a simple universal computational model are register machines (see [43] for some original definitions and [26], [58] for definitions like that we use here).

2.2.1 Definition

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 $f : \mathbf{N}^{\alpha} \rightarrow \mathbf{N}^{\beta}$; starting with an input vector $(n_1, ..., n_{\alpha}) \in \mathbf{N}^{\alpha}$ in registers 1 to α, M has computed $f(n_1, ..., n_{\alpha}) = (r_1, ..., r_{\beta})$ if it halts in the final label h with registers 1 to β containing r_1 to r_{β}. If the final label cannot be reached, $f(n_1, ..., n_{\alpha})$ remains undefined.

A deterministic n-register machine can also analyse an input $(n_1, ..., n_{\alpha}) \in \mathbf{N}^{\alpha}$ in registers 1 to α, 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 (β) register(s) and with all other registers being empty.

2.2.2 Results

The following theorem was already established in [26], based on results from [43] and [36]:

Theorem 1 *Let $L \subseteq \mathbf{N}^n$ be a recursively enumerable set of vectors of non-negative integers. Then L can be accepted by a deterministic m-register machine with $m \leq n+2$ registers.*

Proof (sketch). For the m-register machine, we consider the restricted model (see [43]) that uses the following labeled program instructions:

- $k : A(i)$ corresponds to $(A(i), k+1, k+1)$;
- $k : S(i,m)$ corresponds to $(S(i), m, k+1)$.

1. Start with the vector $(k_1, ..., k_n)$ in the first n registers.
2. Encode this vector $(k_1, ..., k_n)$ as $2^{k_1}3^{k_2}...p_n^{k_n}$ in register $n+1$:

   $l_1 : \quad A(n+1)$
   $l_1+1 : \quad S(n+2, l_1)$

   As long as register i, $1 \leq i \leq n$, is not empty, delete 1 from its contents and proceed with the according instruction $q_i + p_i$ to multiply the contents of register $n+1$ with p_i; when registers 1 to n are empty, proceed to instruction l_2:

   $l_1+2 : \quad S(1, q_1+p_1)$
   ... ...
   $l_1+i+1 : \quad S(i, q_i+p_i)$
   ... ...
   $l_1+n+1 : \quad S(n, q_n+p_n)$
   $l_1+n+2 : \quad A(n+1)$
   $l_1+n+3 : \quad S(n+1, l_2)$

   For each p_i we have the following sequence of instructions to multiply the contents of register $n+1$ with p_i, where $1 \leq i \leq n$:

q_i : $A(n+2)$
... ...
$q_i + p_i - 1$: $A(n+2)$
$q_i + p_i$: $S(n+1, q_i)$
$q_i + p_i + 1$: $S(n+2, l_1)$

Recopying to register $n+1$ is done by instructions l_1 and $l_1 + 1$.

3. Copy $2^{k_1}3^{k_2}...p_n^{k_n}$ into register 1 :

l_2 : $S(n+1, l_2+1)$
$l_2 + 1$: $A(1)$
$l_2 + 2$: $A(n+2)$
$l_2 + 3$: $S(n+1, l_2+1)$
$l_2 + 4$: $A(n+1)$
$l_2 + 5$: $S(n+2, l_2+4)$
$l_2 + 6$: $S(n+1, l_3)$

4. Simulate the computation of the original register machine M_L only in registers $n+1$ and $n+2$ (the start label for this simulation of M_L is l_3, the stop label of M_L has to be identified with label l_4):

- $k : (A(i), l)$ is simulated by:

  m_k : $S(n+1, m_k+1)$
  $m_k + 1$: $A(n+2)$
  ... ...
  $m_k + p_i$: $A(n+2)$
  $m_k + p_i + 1$: $S(n+1, m_k+1)$
  $m_k + p_i + 2$: $A(n+1)$
  $m_k + p_i + 3$: $S(n+2, m_k+p_i+2)$
  $m_k + p_i + 4$: $S(n+1, m_l)$

  Thus, adding 1 to register i in the original register machine is simulated by multiplying the contents of register $n+1$ with p_i.

- $k : (S(i), l, m)$ is simulated by:

  First, we subtract p_i from the contents of register $n+1$ as often as possible and each time add 1 in register $n+2$:

x_k : $S(n+1, x_k+1)$
x_k+1 : $S(n+1, x_k+4)$
x_k+2 : $A(n+1)$
x_k+3 : $S(n+1, x_k+4p_i+1)$
... ...
x_k+3j-2 : $S(n+1, x_k+3j+1)$
x_k+3j-1 : $A(n+1)$
x_k+3j : $S(n+1, x_k+4p_i+2-j)$
... ...
x_k+3p_i-5 : $S(n+1, x_k+3p_i-2)$
x_k+3p_i-4 : $A(n+1)$
x_k+3p_i-3 : $S(n+1, x_k+3p_i+3)$
x_k+3p_i-2 : $A(n+2)$
x_k+3p_i-1 : $S(n+1, x_k+1)$

If the contents of register $n+1$ was divisible by p_i without remainder (i.e., 1 could be subtracted form register i in the original register machine), we move the contents of register $n+2$ back into register $n+1$ and proceed with instruction x_l :

x_k+3p_i : $A(n+1)$
x_k+3p_i+1 : $S(n+2, x_k+3p_i)$
x_k+3p_i+2 : $S(n+1, x_l)$

If the contents of register $n+1$ was not divisible by p_i (i.e., register i in the original register machine was empty), we jump to an instruction between x_k+3p_i+3 and x_k+4p_i+1 to add the corresponding remainder and, if necessary (i.e., if register $n+2$ is not empty), we add the contents of register $n+2$ multiplied by p_i to the contents of register $n+1$, before finally jumping to instruction x_m:

x_k+3p_i+3 : $A(n+1)$
... ...
x_k+4p_i+1 : $A(n+1)$
x_k+4p_i+2 : $A(n+1)$
x_k+4p_i+3 : $S(n+2, x_k+4p_i+5)$
x_k+4p_i+4 : $S(n+1, x_m)$
x_k+4p_i+5 : $A(n+1)$
... ...
x_k+5p_i+4 : $A(n+1)$
x_k+5p_i+5 : $S(n+2, x_k+4p_i+5)$
x_k+5p_i+6 : $S(n+1, x_m)$

5. If the simulated register machine halts in the label we identify with l_4,

the first register contains the encoding $2^{k_1}3^{k_2}...p_n^{k_n}$ of the original input vector $(k_1, ..., k_n)$, whereas all other registers are zero.

Thus we have proved that L can be accepted by the $n+2$-register machine described above, which in addition halts in the final state with $2^{k_1}3^{k_2}...p_n^{k_n}$ in register 1 when started with the vector $(k_1, ..., k_n)$ in its first n registers. Moreover, it is worth mentioning that the actions of the original register machine can be simulated with only two registers in the new register machine when using the encoding described above (compare with [43]). □

From the above result we can immediately conclude the following:

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

Moreover, for sets of strings we have a similar result (also see [36]):

Proposition 3 *For any recursively enumerable set of strings L over the alphabet T with $card\,(T) = z-1$ 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.*

Chapter 3

P Automata - a Brief Literature Review

What about using P systems as *accepting* devices?

This question was raised by Gheorghe Păun as Problem $Q32$ in [51], and maybe even before the book was printed, a first answer was given by Erzsébet Csuhaj-Varjú and György Vaszil in [14] by using purely communicating rules. This kind of rules was introduced in [47] by formalizing the way two chemicals cross the membrane in a collaborating manner: Either they pass through in the same direction (which is called symport) or in opposite directions (antiport). In systems that use only communication rules of this type, the objects are just moved within the regions without being affected by the application of the rules.

In this chapter we briefly introduce this first model in a rather informal way, also summarizing related results from [42] and improvements given in [24]. What follows is a brief presentation of the award-winning answer to another interesting question proposed in [50], which was provided by Oscar Ibarra in [39]. We conclude this Chapter by pointing out related literature.

3.1 The Original Model, a Variant and Improvements

The notion of a P automaton was introduced in [14] as a purely communicating accepting P system with one-way communication. That is, under some given conditions, a multiset of objects can be imported into membrane i only from the membrane immediately outside of i, which is the outer region (also called environment) in case of the skin membrane.

A *one-way P automaton* with n membranes is defined in [14] as a construct

$$\Gamma = (V, \mu, (w_1, P_1, F_1), ..., (w_n, P_n, F_n))$$

where

- V is an alphabet of objects,
- μ is a membrane structure consisting of n membranes (regions), with the membranes labelled in a one-to-one manner by natural numbers $1, .., n$,
- w_i, $1 \leq i \leq n$, are finite multisets over V representing the initial contents (state) of membrane i,
- P_i, $1 \leq i \leq n$, are finite sets of communication rules associated to region i, with the rules having the form $x : y \rightarrow in$, where x and y are finite multisets over V,
- F_i, $1 \leq i \leq n$, are finite sets of multisets over V, called the set of final states of region i.

The top-down communication is performed by conditional symport rules of the form indicated above. Such a rule, in [16] and [24] also written as $(y, in)\ |_x$, can be applied, provided region i contains the multiset x and the multiset given by y is present in the region immediately outside of i. Starting from an *initial configuration* (also called initial state), which consists of an $n-$tuple of multisets of objects initially present in the n regions of the system, a *computation* is performed by a sequential rule application: At each step, exactly one rule is applied in each membrane until either the system aborts (if there is at least one membrane where no rule can be applied) or it reaches a *final configuration*, where for every non-empty set of multisets F_i assigned to region i the contents of this region exactly coincides with an element of F_i.

The sequence of multisets of objects that enter the skin membrane can be considered as *input sequence*, corresponding to a word read from an input tape. The regions and objects represent both storage tapes and states of an automaton. An input sequence is *accepted* by a P automaton as described above, if it reaches a final state.

It was shown in [14] that any language which can be recognized by a two-counter machine can also be obtained as an $l-$projection of the language accepted by some one-way P automaton consisting of seven membranes.

A variant of the above system was investigated in [42], where both objects and states are considered: Rules are of the form $(qy, in)\ |_{px}$, where p as well as q are states and x, y are multisets of objects. One state-object is associated with each region. A multiset that enters the system during a computation is accepted, if the system halts in a final state. This type of systems equipped with only two membranes was proved to accept all recursively enumerable sets of natural numbers in [42].

In [24], the results from [14] and [42] were significantly improved: It was shown that one-way P automata can recognize all recursively enumerable languages in only two membranes, moreover reducing the size of promoters from 4 to 2 as well as the size of the moved multisets from 6 to 2, while the result from [42] could be extended to languages, even for systems with restricted forms of rules.

Moreover, a new mode of introducing the strings into the system was proposed, the so-called *initial mode*: In the first steps of a computation, the string x (consisting of symbols from a terminal alphabet) to be recognized is introduced into the system symbol by symbol. After this procedure, the computation may continue, eventually bringing other symbols in, but without allowing any symbol from the original string to leave the system. If the system halts, x is accepted. For the systems from [14] and [42] working in the initial mode, the same improved results hold true for languages over a one-letter alphabet.

3.2 Inducing an Infinite Hierarchy

Another interesting model was introduced in [39] as a restricted variant of communicating P systems (see [58]), called restricted communicating P systems, or RCPS for short.

Such an RCPS consists of an alphabet of objects V including a distinguished object o, and a membrane structure μ with $m \geq n$ membranes, n

of them being designated input membranes. Initially, the regions may contain multisets of objects, whereas the objects o may only be put into the input membranes. Moreover, each region has a set of evolution rules $u \to v$ associated with it, that can be of the following forms:

1. $a \to a_\tau$ or
2. $ab \to a_{\tau_1} b_{\tau_2}$ or
3. $ab \to a_{\tau_1} b_{\tau_2} c_{come}$,

where $a, b, c \in V$ and $\tau, \tau_1, \tau_2 \in \{here, out\} \cup \{in_j \mid 1 \leq j \leq n\}$. For an object being in membrane k, if it appears in an evolution rule with subscript *out* or in_j, respectively, then it is transported out to the surrounding region or into membrane j, respectively, provided that j is directly contained in (adjacent to) k. The subscript *here* means that the corresponding objects have to stay in the same place. Evolution rules of the third type can only be placed in the skin membrane, having the meaning that the object c is imported from the environment. In one transition step, all possible rules are applied in a maximally parallel way. In contrast to the systems introduced in [58], the environment of an RCPS initially does not contain any object at all. Only objects transferred out of the skin membrane during a computation can later be imported again. Hence, the objects present in the system and its environment are the same in each step of the computation.

An RCPS can now be seen as an acceptor of $n-$tuples of non-negative integers in the following way: Let $\alpha = (i_1, ..., i_n) \in \mathbf{N}^n$. The RCPS is said to accept α, if it eventually halts after being started with $(o^{i_1}, ..., o^{i_n})$ in the designated input membranes, but no objects o in other membranes. It is worth noting that the objects from $V \setminus \{o\}$ have fixed numbers and their distribution across the membranes is initially given, too, independent of α.

It is then shown that RCPSs as defined above are equivalent to two-way multihead finite automata over bounded languages. Moreover, an answer to another question raised by Gheorghe Păun in [50] (for which even a prize was offered, see [63]) is provided: The number of membranes in RCPSs induces an infinite hierarchy. That is, for every r, there is an $s > r$ and a unary language l accepted by an RCPS with s membranes that cannot be accepted by an RCPS with r membranes.

Other variants of RCPSs are investigated, showing that they form an infinite hierarchy with respect to the number of membranes, too. For more details, we refer to [39].

3.3 Some Further Remarks

Still using purely communicating rules only, variants of P automata as introduced in [14], [15] are investigated in [16], this time using symport and antiport rules, with and without promoters. Another model closely related to the original one, but using priorities among the rules, is studied in [10]. So-called evolution communication P automata (using both evolution and communication rules) are explored in [1]. Another approach is made in [40], [41], where the input symbols exist on a tape which is transferred across the membranes during the computation. Yet another class of P automata is introduced in [4], using the features of membrane dissolution and creation and applying these so-called active P automata to the parsing of (natural language) sentences into dependency trees.

For more variants and results we refer to the well maintained up-to-date bibliography of this area at [63].

Chapter 4

Analysing P Systems with Antiport Rules

In contrast to various other models of P systems, where the objects themselves can be transformed during a computation, here we consider purely communicating systems (as already done for the first time in [47], also see [35]), and, moreover, we use these systems for analysing an input sequence of terminal symbols (for a first variant of P automata see [14]).

We show that analysing P systems with only one membrane and antiport rules of radius $(1,2)$ and $(2,1)$, can already recognize any recursively enumerable language of strings; 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 [27] as well as in [38].

What follows has been published in [28], yet here we additionally consider analysing P systems with antiport rules that we call initial. We will show that these systems, where the multiset of terminal objects to be analysed is initially put into a specified membrane, can recognize any recursively enumerable set of (vectors of) non-negative integers.

4.1 Definitions

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

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

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 $1, ..., n$ in a one-to-one manner);
- $w_1, ..., w_n$ are multisets over V associated with the regions $1, ..., n$ of μ;
- $R_1, ..., R_n$ are finite sets of *antiport rules* associated with the regions (membranes) $1, ..., n$; an antiport rule is of the form $(x, out; y, in)$, 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 $(x, out; y, in)$ is defined as $(|x|, |y|)$.

Starting from the *initial configuration*, which consists of μ and $w_1, ..., w_n$, 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 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.)

On the other hand, we might also consider systems for accepting (vectors of) non-negative integers, where the multiset over T to be analysed is initially put into a specified membrane together with possibly some other symbols from V. Such a system will be called *initial analysing P system with antiport rules* in the following.

Remark 4 *We here do not demand the membrane where the multiset of terminal symbols is put initially to be an elementary membrane. But of course, we could also consider to use an additional elementary input membrane, as this is the case for generating P systems having an elementary output membrane.*

4.2 Results

Based on the proof techniques used, e.g., in [26], [36], we can immediately show the following results using Proposition 3.

Theorem 5 *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 radius $(|x|, |y|) \in \{(1,2), (2,1)\}$ only.*

Proof (sketch). According to Proposition 3, 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 $j : (A(i), k, l)$ can be 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 $(f'f'', out; f, in)$
  $(j'', out; j'''j'''', in)$
  $(j'''j'''', out; j'''', in)$
  $(j'j'''', out; l, in)$

  The condition of maximal parallelism guarantees that $(j'a_i, out; f, in)$ is applied in parallel with the rule $(j'', out; j'''j'''', in)$, 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 two steps for being used in the rule $(j'j'''', out; l, in)$ together with the symbol j'''' introduced by the rule $(j'''j'''', out; j'''', in)$.
- 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 $(q, out; q_a a, in)$. 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:

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

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

$q''_{a,i}$: $(A(1), q''_{a,i+1}, q''_{a,i+1})$ 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 3:

$(q, out; q'q'', in)$
$(q'q'', out; q_0, in)$

where q_0 corresponds to 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 $g_z(w)$. □

Observe that, in contrast to P systems with antiport rules as defined in [47], 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 $(q, out; q_a a, in)$.

Considering initial analysing P systems with antiport rules, we obtain a similar result for recursively enumerable multisets over T (and the corresponding sets of vectors of natural numbers, respectively):

Corollary 6 *Let $L \subseteq \mathbf{N}^k$, $k \geq 1$, be a recursively enumerable set of (vectors of) non-negative integers. Then L can be accepted by an initial analysing P system with antiport rules of radius $(1,2)$ or $(2,1)$, in only one membrane.*

Proof (sketch). Other than in the previous proof, we now do not have to take care of how to read the input, as it is already contained in the skin

membrane. Thus we can immediately start the $(k+2)$ −register machine from Proposition 2, where the multiset over T initially present represents the contents of the first k registers. We only have to show how we can simulate the instructions of the register machine by antiport rules of radius $(1,2)$ or $(2,1)$, which is done exactly in the same way as in the proof of Theorem 5. □

4.3 Conclusion

We have investigated (initial) analysing P systems with antiport rules which surprisingly already obtain their maximal recognizing power with the simplest membrane structure and rules with radius $(1,2)$ or $(2,1)$. According to the features of the 3-register machine constructed in Proposition 3 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 to the situation of an automaton accepting by a final state (also compare with the definition of acceptance by P automata as defined in [14]). 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 [47]).

Chapter 5

P Automata with Membrane Channels

In this Chapter (that has partly appeared in [46] and [33]), we give the definition of a P automaton with membrane channels and its specific variants, especially using only antiport rules, as well as a very restricted variant characterizing regular languages. Again, we consider the initial variant of these systems as well.

In the model presented here, objects can cross the membranes by passing through corresponding channels that have been opened by means of activators, unless the channel is blocked by a prohibitor. Applying an activating rule means that an activator multiset (or a single activator symbol) opens input and output channels for specific objects. In the following substep, each object can pass through the surrounding membrane provided there is no prohibitor active, which prevents the object from passing through the corresponding channel. Whereas P systems with activated / prohibited membrane channels used as computing and generating devices have been investigated in [27], we here consider such systems as accepting devices, using them for analysing an input sequence of terminal symbols, as done for the first time in [14]. According to usual notations in formal language theory we will call such devices P automata with membrane channels.

5.1 Definitions

A *P automaton with membrane channels* is a construct Π of the following form:

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

where

1. V is an alphabet of *objects*;
2. $T \subseteq V$ is the terminal alphabet;
3. μ is a *membrane structure* (with the membranes labelled by natural numbers $1, ..., n$ in a one-to-one manner);
4. $w_1, ..., w_n$ are multisets over V associated with the regions $1, ..., n$ of μ;
5. $R_1, ..., R_n$ are finite sets of *rules* associated with the compartments $1, ..., n$, which can be of the following forms:
   - *activating rules*: $\langle P; x, out; y, in \rangle$, where $x, y \in V^*$ and P is a finite multiset over V,
   - *prohibiting rules*: $\langle b, out; Q \rangle$ or $\langle b, in; Q \rangle$, where $b \in V$ and Q is a finite multiset over V.
6. F is a finite set of *final states.*

A *final state* is a function f assigning a finite multiset with each membrane region (see [14]); the empty set (in the following we shall use the special symbol Λ instead of $\emptyset$ for specifying this case) indicates that we do not care about the contents of the corresponding region, whereas for every non-empty multiset assigned to a membrane region the contents of this region must coincide with this finite multiset; in this case we say that the underlying configuration has reached the final state f.

Starting from the *initial configuration*, which consists of μ and $w_1, ..., w_n$, the system passes from one configuration to another one by non-deterministically in a maximally parallel way applying rules from R_i in the following sense: Let $x = x_1...x_m$ and $y = y_1...y_k$. An activating rule $\langle P; x, out; y, in \rangle$ means that by the activator multiset P an output channel for each symbol x_i, $1 \leq i \leq m$, is activated, and for each y_j, $1 \leq j \leq k$, an input channel is activated. In the following substep of a derivation (computation), each activated channel allows for the transport of one object x_i and

y_j, respectively, provided there is no prohibitor multiset Q active by a prohibiting rule $\langle x_i, out; Q\rangle$ or $\langle y_j, in; Q\rangle$, respectively (which means that the multiset Q can be found in the underlying compartment). The activating multisets P in the activating rules have to be chosen in a maximally parallel way.

A system that uses only activating rules is called a *P automaton with activated membrane channels* in the following.

A sequence of transitions is called a *computation*. For a multiset or a string w over an alphabet T, a computation usually is called *successful*, if and only if it halts (i.e., no rule can be applied anymore); yet following the idea of final states introduced in [14], in this Chapter we shall call a computation *successful*, if and only if it reaches a final state f from F. A multiset or a string w over an alphabet T is recognized by the P automaton with membrane channels 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.)

Again we will also consider *initial P automata with activated membrane channels* able to eventually accept (vectors of) non-negative integers, when being initially supplied with the multiset over T to be analysed in the skin membrane together with possibly some other symbols from V.

5.2 Results

The main result established in [43] is that the actions of a deterministic Turing machine can be simulated by a 2-register machine. Based on this result and the proof techniques used, e.g., in [26] and [36], we can immediately show the following results using Proposition 3.

Theorem 7 *Let $L \subseteq T^*$ be a recursively enumerable set. Then L can be recognized by a P automaton with membrane channels in only one membrane using only singleton activators and prohibitors.*

Proof (sketch). According to Proposition 3, 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:

1. An Add-instruction $j : (A(i), k, k)$ is simulated by the activating rule $\langle j; j, out; ka_i, in\rangle$.

2. A conditional Subtract-instruction $j : (S(i), k, l)$ is simulated by the following rules:

   $\langle j; ja_i, out; kf_j, in\rangle$ $\quad \langle f_j, in; a_i\rangle$
   $\langle j; j, out; j'j'', in\rangle$ $\quad \langle j'', in; a_i\rangle$
   $\langle j'; j'j'', out; lf_j, in\rangle$ $\quad \langle f_j, in; j''\rangle$
   $\langle f_j; f_j, out; f_j, in\rangle$

   The construction of the rules in the P automaton with membrane channels guarantees that rules sending out another object together with the activating symbol can only be used without introducing a failure symbol (trap symbol) if also this other object is present in the skin membrane.

3. The halting instruction $h : HALT$ is simulated by just taking the halting symbol h as final state.

Now let us start with the singleton q in the initial configuration. For every $a \in T$ we take $\langle q; q, out; q_a a, in\rangle$. 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:

instruction		**simulated by**	
$q_a :$	$(S(1), q_{a,1}, \hat{q}'_a)$	$\langle q_a; q_a a_1, out; \hat{q}'_a f_1, in\rangle$	$\langle f_{q_a}, in; a_1\rangle$
		$\langle q_a; q_a, out; q'_a q''_a, in\rangle$	$\langle q''_a, in; a_1\rangle$
		$\langle q'_a; q'_a q''_a, out; \hat{q}'_a f_{q_a}, in\rangle$	$\langle f_{q_a}, in; q''_a\rangle$
		$\langle f_{q_a}; f_{q_a}, out; f_{q_a}, in\rangle$	
$q_{a,i} :$	$(A(2), q_{a,i+1}, q_{a,i+1})$	$\langle q_{a,i}; q_{a,i}, out; q_{a,i+1} a_2, in\rangle$	for $1 \leq i < z$
$q_{a,z} :$	$(A(2), q_a, q_a)$	$\langle q_{a,z}; q_{a,z}, out; q_a a_2, in\rangle$	
$\hat{q}_a :$	$(S(2), \hat{q}_{a,1}, \hat{q}'_{a,1})$	$\langle \hat{q}_a; \hat{q}_a a_2, out; \hat{q}'_{a,1} f_{\hat{q}_a}, in\rangle$	$\langle f_{\hat{q}_a}, in; a_2\rangle$
		$\langle \hat{q}_a; \hat{q}_a, out; \hat{q}'_a \hat{q}''_a, in\rangle$	$\langle \hat{q}''_a, in; a_2\rangle$
		$\langle \hat{q}'_a; \hat{q}'_a \hat{q}''_a, out; \hat{q}'_{a,1} f_{\hat{q}_a}, in\rangle$	$\langle f_{\hat{q}_a}, in; \hat{q}''_a\rangle$
		$\langle f_{\hat{q}_a}; f_{\hat{q}_a}, out; f_{\hat{q}_a}, in\rangle$	
$\hat{q}_{a,1} :$	$(A(1), \hat{q}_a, \hat{q}_a)$	$\langle \hat{q}_{a,1}; \hat{q}_{a,1}, out; \hat{q}_a a_1, in\rangle$	

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

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

The input of the next terminal symbol starts with the activating rule $\langle q''_{a,k}; q''_{a,k} a, out; q, in\rangle$.

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 have to compute $2^{g_z(w)}$ before using the following activating rule to start the simulation of the 3-register machine indicated in Proposition 3:

$$\langle q; q, out; q_0, in \rangle$$

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

Obviously, the P automaton halts if and only if the register machine accepts the input $g_z(w)$. □

The string to be accepted is given by the sequence of terminal symbols a taken from the environment by activating rules of the form $\langle q; q, out; q_a a, in \rangle$.

If considering now the acceptance of sets of vectors of natural numbers by initial P automata with activated membrane channels, we obtain a similar result.

Corollary 8 *Let $L \subseteq \mathbf{N}^k$, $k \geq 1$, be a recursively enumerable set of (vectors of) non-negative integers. Then L can be accepted by an initial P automaton with activated membrane channels in only one membrane using only singleton activators and prohibitors.*

Proof (sketch). As already explained in the proof of Corollary 6, we only have to show how the instructions of the register machine from Proposition 2 are simulated, which is done as in the proof of Theorem 7. □

With respect to the size of the activator and prohibitor multisets the above results are already optimal. But if we here define the radius of an activating rule $\langle P; x, out; y, in \rangle$ as the pair of numbers $(|x|, |y|)$, then we still can improve the above result:

Theorem 9 *Let $L \subseteq T^*$ be a recursively enumerable set. Then L can be recognized by a P automaton with membrane channels in only one membrane using only singleton activators and prohibitors with the activating rules having radius $(1,2)$ or $(2,1)$.*

Proof. Now we only have to make sure that all activating rules have a radius of $(1,2)$ or $(2,1)$. The activating rule for the simulation of an Add-instruction from the proof of Theorem 7 already fulfills this condition. We only have to change the rules that are needed for the simulation of the Subtract-instruction to fit our needs:

A conditional Subtract-instruction $j : (S(i), k, l)$ is now simulated by the following rules:

$\langle j; j, out; j' f_j, in\rangle \qquad \langle f_j, in; a_i\rangle$
$\langle j'; j' a_i, out; k, in\rangle$
$\langle j; j, out; j'' j''', in\rangle \qquad \langle j''', in; a_i\rangle$
$\langle j''; j'', out; j'''' f_j, in\rangle \quad \langle f_j, in; j'''\rangle$
$\langle j''''; j''' j'''', out; l, in\rangle$
$\langle f_j; f_j, out; f_j' f_j'', in\rangle$
$\langle f_j'; f_j' f_j'', out; f_j, in\rangle$

Again we now start with the singleton q in the initial configuration. For every $a \in T$ we take the activating rule $\langle q; q, out; q_a a, in\rangle$. 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:

$q_a : \quad (S(1), q_{a,1}, q_a')$
$q_{a,i} : \quad (A(2), q_{a,i+1}, q_{a,i+1})$ for $1 \leq i < z$
$q_{a,z} : \quad (A(2), q_a, q_a)$
$q_a' : \quad (S(2), q_{a,1}', q_{a,1}'')$
$q_{a,1}' : \quad (A(1), q_a', q_a')$

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

$q_{a,i}'' : \quad (A(1), q_{a,i+1}'', q_{a,i+1}'')$ for $1 \leq i \leq k-1$

The input of the next terminal symbol starts with the activating rule $\langle q_{a,k}''; q_{a,k}'' a, out; q, in\rangle$.

If no further input symbols should be taken in, we have to compute $2^{g_z(w)}$ before using the following activating rules to start the simulation of the 3-register machine indicated in Proposition 3:

$$\langle q; q, out; q'' q''', in\rangle$$
$$\langle q''; q'' q''', out; q_0, in\rangle$$

where q_0 corresponds to 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 $g_z(w)$. □

We now establish a similar result to Corollary 8 for recursively enumerable multisets over T (and the corresponding sets of vectors of natural numbers, respectively):

Corollary 10 *Let $L \subseteq \mathbf{N}^k$, $k \geq 1$, be a recursively enumerable set of (vectors of) non-negative integers. Then L can be accepted by an initial P automaton with membrane channels in only one membrane using only singleton activators and prohibitors with the activating rules having radius $(1,2)$ or $(2,1)$.*

On the other hand, we can also get rid of the prohibiting rules by allowing the size of the activator multisets being greater than one:

Theorem 11 *Let $L \subseteq T^*$ be a recursively enumerable set. Then L can be recognized by a P automaton with activated membrane channels in only one membrane with the activating rules having radius $(1,2)$ or $(2,1)$.*

Proof. Again we use the same construction as in the proof of Theorem 7, only that now we have to replace the rules to simulate conditional Subtract-instruction by the following ones:

$\langle ja_i; ja_i, out; k, in\rangle$
$\langle j; j, out; j'j'', in\rangle$
$\langle j'a_i; j'a_i, out; f_j, in\rangle \quad \langle f_j; f_j, out; f'_j f''_j, in\rangle \; \langle f'_j; f'_j f''_j, out; f_j, in\rangle$
$\langle j''; j'', out; j'''j'''', in\rangle$
$\langle j'''; j'''j'''', out; j'''', in\rangle$
$\langle j''''; j'j'''', out; l, in\rangle$

The condition of maximal parallelism guarantees that $\langle j'a_i; j'a_i, out; f_j, in\rangle$ is applied in parallel with the rule $\langle j''; j'', out; j'''j'''', in\rangle$, which leads to a non-halting computation by the introduction of the failure symbol (trap symbol) f_j. Only if in the current configuration no symbol a_i is present in the skin membrane, the object j' can wait two more steps until being used in the rule $\langle j''''; j'j'''', out; l, in\rangle$ together with the symbol j''''. □

And of course again we can immediately conclude the following:

Corollary 12 *Let $L \subseteq \mathbf{N}^k$, $k \geq 1$, be a recursively enumerable set of (vectors of) non-negative integers. Then L can be accepted by an initial P automaton with activated membrane channels in only one membrane using only activating rules with radius $(1,2)$ or $(2,1)$.*

5.3 (Finite) P automata with Antiport Rules

In this last proof, we sometimes used rules of the form $\langle x; x, out; y, in\rangle$, where $x, y \in V^+$, thereby "forcing" all output channels to be used, i.e., for the application of such a rule, all objects x have to be present in the skin membrane and pass through the opened channels in the subsequent step. (Remember that input channels are always used, as due to our definition, the environment contains all objects in an arbitrary number.)

Hence, writing these special rules in a shorter way as $(x, out; y, in)$, we get antiport rules as explained in the previous Chapter. In the following, we will call this special variant of a P automaton with activated membrane

channels a *P automaton with antiport rules.* Here again, we define the *radius* of the rule $(x, out; y, in)$ as the pair of numbers $(|x|, |y|)$.

In Chapter 4, it was shown that analysing P systems with antiport rules of radius $(2,1)$ or $(1,2)$, consisting of only one membrane can already recognize any recursively enumerable language of multisets and strings, respectively; in fact, this was proved for halting computations, but obviously the same is true for computations accepting by final states, too:

Theorem 13 *Let $L \subseteq \Sigma^*$ be a recursively enumerable language. Then L can be accepted by a P automaton with antiport rules that consists of the simplest membrane structure and only uses rules of the form $(x, out; y, in)$ with the radius of the rules being $(2,1)$ or $(1,2)$.*

When considering P automata with antiport rules that consist of the simplest membrane structure, i.e., only the skin membrane, and use only rules of very specific forms, we obtain characterizations of regular languages which is shown in the following:

A *finite P automaton with antiport rules* is a P automaton with antiport rules

$$\Pi = (V, T, [_1]_1, w_1, R_1, F)$$

with only one membrane such that

1. the axiom w_1 is a non-terminal symbol (in the case of finite P automata with antiport rules simply called "state"), i.e., $w_1 \in V \setminus T$;
2. antiport rules in R_1 are of the forms $(q, out; pa, in)$ and $(pa, out; r, in)$, where a is a terminal symbol in T and p, q, r are non-terminal symbols (states) in $V \setminus T$;
3. for any rule $(q, out; pa, in)$ in R_1, the only other rules in R_1 containing p are of the form $(pa, out; r, in)$;
4. $F \subseteq V \setminus T$ (more precisely, each multiset in F consists of exactly one "state").

The following example shows the importance of the third condition given above for finite P automata, which guarantees that every terminal symbol taken into the skin membrane is immediately sent out again in the next step.

Example 14 *Consider the P automaton with antiport rules*

$$\Pi_1 = (\{a, b, p, q, r\}, \{a, b\}, [_1]_1, p, R_1, \{q\})$$

with R_1 containing the rules

$$(p, out; pa, in), (p, out; qa, in), (qa, out; r, in), (r, out; sb, in), (sb, out; q, in).$$

Then the rule $(p, out; pa, in)$ imports an arbitrary number n of symbols a, finally at least one symbol a is taken in by the rule $(p, out; qa, in)$. Every application of the sequence of the rules $(qa, out; r, in)$, $(r, out; sb, in)$, and $(sb, out; q, in)$ sends out one of the symbols a while at the same time "accepting" one symbol b by importing it and sending it out immediately in the succeeding step. Hence, the computation halts successfully in the final state q after having accepted the string $a^n b^n$; hence, $L(\Pi_1) = \{a^n b^n \mid n \geq 1\}$, which is a well-known linear, but non-regular language.

Theorem 15 *Let $L \subseteq T^+$. Then L is regular if and only if L is accepted by a finite P automaton.*

Proof. Let L be a regular (string) language accepted by the FA $M = (Q, T, \delta, q_0, F_M)$. Then we can construct a finite P automaton

$$\Pi = (V, T_\Pi, [_1]_1, w_1, R_1, F)$$

that accepts L with

1. $V = Q \cup \{(q, a, r) \mid r \in \delta(q, a)$ for some $q, r \in Q, a \in T\} \cup T$;
2. $T_\Pi = T$;
3. $w_1 = q_0$;
4. $R_1 = \{(p, out; (q, a, r)\, a, in), ((q, a, r)\, a, out; r, in) \mid$
   $p, q, r \in Q, a \in T$ and $r \in \delta(q, a)\}$;
5. $F = F_M$.

Every step in M using the transition (q, a, r) is simulated by two steps in Π using the rules $(p, out; (q, a, r)\, a, in), ((q, a, r)\, a, out; r, in)$ via the intermediate configuration where the skin membrane contains $(q, a, r)\, a$.

On the other hand, if the regular (string) language L is recognized by a finite P automaton $\Pi = (V, T, [_1]_1, w_1, R_1, F)$, then it is also accepted by the finite automaton $M = (Q, T_M, \delta, q_0, F_M)$ with

1. $Q = V \setminus T$;
2. $T_M = T$;

3. $q_0 = w_1$;

4. $\delta = \{((q,a), \{r \in Q \mid (q, out; pa, in), (pa, out; r, in) \in R_1$ for some $p \in Q\}) \mid q \in Q, a \in T\}$;

5. $F_M = F$.

Every application of a sequence $(q, out; pa, in), (pa, out; r, in)$ of rules in R_1 is simulated by only one transition (q, a, r) in M; as the intermediate contents pa of the skin membrane cannot appear as final state, in a successful computation of M the application of the rule $(q, out; pa, in)$ must be followed by the application of a rule $(pa, out; r, in)$ for some $r \in Q$, hence the transitions constructed in δ correctly simulate the rules in R_1. □

5.4 Conclusion

We have investigated some variants of P automata with membrane channels and initial P automata with membrane channels which already obtain their maximal recognizing power with the simplest membrane structure. We have shown that by using singleton activators and prohibitors, the radius of the activating rules can be reduced to $(1, 2)$ or $(2, 1)$, whereas when using activating rules only, the size of the activating multisets has to be slightly increased.

A very restricted variant using only special activating rules (in this case antiport rules) allows for the characterization of regular languages.

Chapter 6

P Automata with Conditional Communication Rules Assigned to Membranes

In contrast to all models of P systems and P automata investigated so far, where the evolution rules are placed within a region, in this Chapter we consider P automata where the purely communicating rules are directly assigned to the membranes (PACCRAMs for short): Depending on promoting and inhibiting multisets inside and outside the membrane, one multiset of objects has to leave the region surrounded by the membrane and another multiset of objects has to enter this region coming from the region outside the membrane (for the skin membrane, this is the environment, which we assume to contain all objects in arbitrarily many copies).

Whereas in [29], the universal computational power of P systems with conditional communication rules with only one membrane and singleton multisets used as promoters and inhibitors as well as singleton objects transported through the skin membrane was shown by means of graph-controlled grammars, we here investigate the accepting variant.

After giving the definition of PACCRAMs, we point out the similarities and differences between the model presented here and related ones with respect to the idea that the rules are directly assigned to the membranes.

We show that PACCRAMs already obtain their maximal recognizing power with only one membrane and the size of the promoting and inhibiting multisets as well as the size of the multisets of objects transported through the skin membrane not exceeding 1.

Note that other than in the original paper ([29]), we here use the notion of P automata with conditional communication rules assigned to membranes.

6.1 PACCRAM - Definition

A *P automaton with conditional communication rules assigned to membranes* (a *PACCRAM* for short) is a construct Π of the following form:

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

where

- V is an alphabet of *objects*;
- $T \subseteq V$ is the alphabet of *terminal objects;*
- μ is a *membrane structure* (with the membranes labelled by natural numbers $1, ..., n$ in a one-to-one manner);
- $w_1, ..., w_n$ are multisets over V associated with the regions $1, ..., n$ of μ;
- $R_1, ..., R_n$ are finite sets of *rules* associated with the membranes $1, ..., n$, which are of the form

  $$(P_{in}, Q_{in}; P_{out}, Q_{out}; y, in; x, out)$$

  where x, y and P_{in}, Q_{in}, P_{out}, Q_{out} are finite multisets over V.

  In a more depictive way, a rule

  $$(P_{in}, Q_{in}; P_{out}, Q_{out}; y, in; x, out)$$

  can be written in the following form:

  $$\begin{array}{rl} P_{out} & P_{in} \\ y \longrightarrow & \\ & \longleftarrow x \\ Q_{out} & Q_{in} \end{array}$$

  Moreover, if $P_{out} = Q_{out} = \lambda$, then the rule

  $$\begin{array}{rl} & P_{in} \\ y \longrightarrow & \\ & \longleftarrow x \\ & Q_{in} \end{array}$$

  will be represented by $(P_{in}, Q_{in}; y, in; x, out)$ thus omitting the multisets P_{out}, Q_{out}; this is the only form of rules relevant for the skin membrane, as we assume the environment to contain all objects in arbitrarily many copies (therefore, the condition given by a multiset P_{out} is always fulfilled and the condition given by a multiset Q_{out} can only be fulfilled for the empty multiset).

- F is a finite set of *final states.*

Starting from the *initial configuration*, which consists of μ and $w_1, ..., w_n$, the system passes from one configuration to another one by non-deterministically choosing one rule from some R_i and applying it in the following sense (observe that here we consider a sequential model of applying the rules (as, e.g., done in [22], [21], [25], [45]) instead of choosing rules in a maximally parallel way as it is often required in P systems):

Let $x = x_1...x_m$ and $y = y_1...y_k$. A rule $(P_{in}, Q_{in}; P_{out}, Q_{out}; y, in; x, out)$ means that in the presence of the promoting multisets P_{in} and P_{out} inside and outside the membrane and provided that the inhibiting multisets Q_{in} and Q_{out} are not present inside respectively outside the membrane, the objects x_i, $1 \leq i \leq m$, are sent out and the objects y_j, $1 \leq j \leq k$, are taken into the membrane; the objects forming the promoting multisets P_{in} and P_{out}, respectively, cannot be part of the multisets x and y, respectively.

Again we call a computation *successful*, if and only if it reaches a final state f from F. A multiset or a string w over an alphabet T is recognized by the PACCRAM 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.)

To obtain a model for accepting sets of vectors of non-negative integers, which here will be called initial PACCRAM, we again do not consider an input sequence, but in the beginning put the multiset of terminal symbols to be analysed into a specified region.

We should like to mention that rather similar ideas like those used in PACCRAMs can be found in [5], where communication rules of the form $xx'[_iyy' \rightarrow xy'[_ix'y$ for x, x', y, $y' \in V^*$, $1 \leq i \leq n$, are used, allowing for the transfer of the objects x' and y' across the membrane provided the multisets x and y are present outside and inside, respectively, of membrane i. But in contrast to the systems used in this paper, in [5] these rules are applied in a non-deterministic, maximally parallel manner. Moreover, the skin membrane cannot interact with the environment, and the objects sent out of the system are lost.

Another model resembling the ideas of our new model is that of P systems with promoters/inhibitors introduced in [6], where rules of the forms $u \rightarrow v|_a$ and $u \rightarrow v|_{\neg a}$ are used where the presence respectively absence of the object a in the corresponding region controls the applicability of the rules.

In contrast to P systems with antiport rules as defined in [47], we need not specify the environment, because we assume every symbol to appear in an unlimited number there.

Comparing the notions introduced for P automata with membrane channels as defined in Chapter 5 with the definitions of PACCRAMs we should like to emphasize once more that the objects forming the promoting multisets P_{in} and P_{out} inside and outside the membrane cannot be transported through the membrane when the corresponding rule is applied.

To give a first impression of how a PACCRAM works, we consider the following example (compare with Example 14):

Example 16 *Consider the PACCRAM*

$$\Pi_2 = (\{a, b, p, q, r\}, \{a, b\}, [_1]_1, p, R_1, \{q\})$$

where

$$R_1 = \left\{ pa \longrightarrow \Big| \longleftarrow p \,,\; qa \longrightarrow \Big| \longleftarrow p \right\} \cup$$
$$\left\{ r \longrightarrow \Big| \longleftarrow qa \,,\; sb \longrightarrow \Big| \longleftarrow r \,,\; q \longrightarrow \Big| \longleftarrow sb \right\}.$$

Then the rule $(\lambda, \lambda; pa, in; p, out)$ *imports an arbitrary number* n *of symbols* a*, finally at least one symbol* a *is taken in by the rule* $(\lambda, \lambda; qa, in; p, out)$*. Every application of the sequence of* $(\lambda, \lambda; r, in; qa, out)$*,* $(\lambda, \lambda; sb, in; r, out)$*, and* $(\lambda, \lambda; q, in; sb, out)$ *sends out one of the symbols* a *while at the same time "accepting" one symbol* b *by importing it and sending it out immediately in the succeeding step. Hence, the computation halts successfully in the final state* q *after having accepted the string* $a^n b^n$*; thus,* $L(\Pi_2) = \{a^n b^n \mid n \geq 1\}$.

6.2 Results

We use the same proof idea as in previous Chapters to show the main result for PACCRAMs:

Theorem 17 *Let $L \subseteq T^*$ be a recursively enumerable string language. Then L can be accepted by a PACCRAM Π in only one membrane using only singleton promoting and inhibiting multisets as well as singleton objects transported through the skin membrane.*

Proof (Sketch.) According to Proposition 3, 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:

1. To simulate an Add-instruction $j : (A(i), k, k)$ we need the following rules:

$$j' \longrightarrow \left| \begin{array}{l} \\ \longleftarrow j \end{array} \right. , \quad j'' \longrightarrow \left| \begin{array}{l} j' \\ \\ j'' \end{array} \right. , \quad a_i \longrightarrow \left| \begin{array}{l} j'' \\ \longleftarrow j' \end{array} \right. , \quad k \longrightarrow \left| \begin{array}{l} \\ \longleftarrow j'' \\ j' \end{array} \right. .$$

2. For the simulation of a conditional Subtract-instruction $j : (S(i), k, l)$ we include the following rules:

$$l \longrightarrow \left| \begin{array}{l} \\ \longleftarrow j \\ a_i \end{array} \right. \quad \text{as well as}$$

$$k' \longrightarrow \left| \begin{array}{l} a_i \\ \longleftarrow j \end{array} \right. , \quad k'' \longrightarrow \left| \begin{array}{l} k' \\ \longleftarrow a_i \\ k'' \end{array} \right. , \quad \left| \begin{array}{l} k'' \\ \longleftarrow k' \end{array} \right. , \quad k \longrightarrow \left| \begin{array}{l} \\ \longleftarrow k'' \\ k' \end{array} \right. .$$

3. The halting instruction $h : HALT$ is simulated by just taking the halting symbol h as final state.

Let us start with the singleton q in the initial configuration. For every $a \in T$ we take the rules

$$q' \longrightarrow \left| \begin{array}{l} \\ \longleftarrow q \end{array} \right. , \quad q'' \longrightarrow \left| \begin{array}{l} q' \\ \\ q'' \end{array} \right. , \quad a \longrightarrow \left| \begin{array}{l} q'' \\ \longleftarrow q' \end{array} \right. , \quad q_a \longrightarrow \left| \begin{array}{l} \\ \longleftarrow q'' \\ q' \end{array} \right. .$$

Now we assume that 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:

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

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

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

The input of the next terminal symbol starts with the membrane rule

$$q \longrightarrow \Big| \longleftarrow q''_{a,k} .$$

Obviously, the instructions of the subprogram above can be translated into rules assigned to membranes as already elaborated at the beginning of the proof.

In case no further input symbols should be taken in, we have to compute $2^{g_z(w)}$ before starting the simulation of the 3-register machine indicated in Proposition 3. Obviously, the P automaton halts in the final state h if and only if the register machine accepts the input $g_z(w)$. □

A similar result also holds true for initial PACCRAMs accepting recursively enumerable set of (vectors of) non-negative integers:

Corollary 18 *Let $L \subseteq \mathbf{N}^k$, $k \geq 1$, be a recursively enumerable set of (vectors of) non-negative integers. Then L can be accepted by a PACCRAM Π in only one membrane using only singleton promoting and inhibiting multisets as well as singleton objects transported through the skin membrane.*

Remark 19 *The size of the multisets used in the rules of the PACCRAMs can be considered as a complexity measure for this type of P automata. With respect to this complexity measure, the PACCRAMs constructed in the proofs of this Section are already optimal, as we only had to use multisets of size 1, i.e., in the rules $(P_{in}, Q_{in}; y, in; x, out)$ the multisets P_{in}, Q_{in}, y, x either are empty or consist of exactly one symbol.*

6.3 Conclusion

We have investigated (initial) P automata with conditional communication rules assigned to membranes (PACCRAMs), and we have shown that already with the simplest membrane structure and only singleton promoters and inhibitors as well as singleton objects transported through the skin membrane, PACCRAMs can be used as accepting devices for recursively enumerable string languages, whereas initial PACCRAMs allow for accepting recursively enumerable sets of (vectors of) non-negative integers. The results obtained in this Chapter are already optimal with respect to the number of membranes as well as with respect to the size of the multisets used as promoters, inhibitors, and strings transported across a membrane (see Remark 19).

The new idea of assigning the evolution rules directly to the membranes of a membrane system is also used in [23] together with splicing rules and cutting/recombination rules (these rules involve several strings and are often used in the area of DNA computing, see [52]). There again the interplay between the objects inside the skin membrane as well as in the environment through the skin membrane allows for universal computational systems with only one membrane.

Chapter 7

Accepting P systems / P Automata with Catalysts

In the original paper in [48] introducing membrane systems (P systems) as a symbol manipulating model, catalysts as well as priority relations on the rules were used to prove them to be computationally universal; in [60] it was shown that a priority relation on the rules is not necessary to obtain this universality result. In [30] the number of catalysts was reduced by one for the variants of P systems with two membranes considered there; moreover, the number of catalysts could even be reduced by one more when considering computations reaching some finitely specified final states as in the model of P automata introduced in [15] instead of halting computations. In [31] it is shown that even two catalysts are already sufficient for all these variants.

Other than in [31], we here only consider the accepting variants, accepting by halting computations as well as by final states. Moreover, we investigate all these systems having catalytic rules only.

In the following Section, after some prerequisites we describe a special variant of counter automata that we use for proving our results about P automata accepting string languages. Then we define the specific variants of P automata considered in this Chapter. In the further parts of the Chapter we show how we can reduce the number of catalysts in P systems with specific stopping conditions by using new proof techniques for simulating register machines. Using well-known results for two-counter automata we prove our universality results for P automata accepting string languages.

7.1 Definitions

By $N^{\alpha}RE$ we denote the family of recursively enumerable sets of α-vectors $(y_1, ..., y_\alpha)$ of non-negative integers. Two sets of α-vectors are considered to be equal if they only differ at most by the zero-vector $(0, ..., 0)$.

Let $m \geq 2$ and let k, l be two positive integers not greater than m; then we define:

$$l \ominus_m k := \begin{cases} l - k & \text{for } l > k \\ l - k + m & \text{for } l \leq k \end{cases}$$

7.2 Register Machines and Counter Automata

The results proved in [26] (see also Chapter 2) as well as in [36] and [37] immediately lead us to the following results which differ from the original ones mainly by the fact that the result of a computation is stored in registers that must not be decremented:

Proposition 20 *For any partial recursive function $f : \mathbf{N}^{\alpha} \rightarrow \mathbf{N}^{\beta}$ there exists a deterministic $(\alpha + 2 + \beta)$-register machine M computing f in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbf{N}^{\alpha}$ in registers 1 to α, M has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label h with registers $\alpha + 3$ to $\alpha + 2 + \beta$ containing r_1 to r_β, and all other registers being empty; if the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined.*

The following Corollary is an immediate consequence of the preceding proposition (by taking $\beta = 0$):

Corollary 21 *For any recursively enumerable set $L \subseteq \mathbf{N}^{\alpha}$ of vectors of non-negative integers there exists a deterministic $(\alpha + 2)$-register machine M accepting L in such a way that M halts with all registers being empty if and only if M starts with some $(n_1, ..., n_\alpha) \in L$ in registers 1 to α and the registers $\alpha + 1$ to $\alpha + 2$ being empty.*

For dealing with strings, we introduce the following (quite restricted) model of an n-counter automaton, which can be seen as an n-register machine having an additional input tape for the input string to be analysed; we assume this input tape to contain the letters of the input string in the correct order followed by an arbitrary (infinite) number of blank symbols B:

An *n-counter automaton* is a construct

$$M = (n, \Sigma_B, P, i, h)$$

where

- n is the number of registers,
- Σ is the input alphabet and B is a special symbol not in Σ, $\Sigma = \{b_t \mid 1 \leq t \leq s\}$, $s \geq 1$, $\Sigma_B = \Sigma \cup \{B\}$,
- P is a set of labelled instructions of the form $j : (op(r), k, l)$, where $op(r)$ is an operation on register r of M, j, k, l are labels from the set $Lab(M)$ (which numbers the instructions of the program of M represented by P), or of the form $j : (read, k_1, ..., k_s, l)$, where $j, k_1, ..., k_s, l$ are labels from the set $Lab(M)$,
- i is the initial label, and
- h is the final label.

As an n-register machine, an n-counter automaton is capable of the following instructions on its registers:

$(A(r), k, l)$ Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variant we demand $k = l$.

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

Moreover, we again have the $Halt$-operation:

$Halt$: Stop the machine. This additional instruction can only be assigned to the final label h.

In addition, we now also have operations on the input tape which allow for reading the (letters of the) input string:

$j : (read, k_1, ..., k_s, l)$: Read the symbol under the read-only head of the input tape and move the read-only head of the input tape one position to the right; if this symbol equals b_t, $1 \leq t \leq s$, then go to instruction k_t, otherwise (if the read symbol equals B) proceed to instruction l.

Given a string $w \in \Sigma^+$, we say that M accepts w if and only if M, when started with w on its input tape and all counters being empty, finally halts in the final label. Without loss of generality, we may assume that all counters are empty when M halts; moreover, we also assume that after reading the first blank symbol B on the input tape, M never accesses the input tape any more (because then only some more blank symbols would be read). The string language $L \subseteq \Sigma^+$ accepted by M is the set of all strings $w \in \Sigma^+$ accepted by M.

We know from the results proved in [43] that two counters are sufficient to simulate the actions of a Turing machine, hence, the following result can easily be derived even for the special model of n-counter automata defined above:

Proposition 22 *For any recursively enumerable λ-free string language $L \subseteq \Sigma^+$ there exists a (deterministic) 2-counter automaton M accepting L.*

7.3 The Standard Model of P Systems and Variants

The standard type of membrane systems (P systems) has been studied in many papers and several monographs; we refer to [8], [19], [48], [49], [53] and [51] for motivation and examples. In the definition of the P system below we omit some ingredients (like priority relations on the rules) not needed in the following.

A *P system (of degree d, $d \geq 1$)* is a construct

$$\Pi = (V, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o),$$

where:

(i) V is an alphabet; its elements are called *objects*;

(ii) $C \subseteq V$ is a set of *catalysts*;

(iii) μ is a membrane structure consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, $1 \leq i \leq d$);

(iv) w_i, $1 \leq i \leq d$, are strings over V associated with the regions $1, 2, \ldots, d$ of μ; they represent multisets of objects present in the regions of μ (the multiplicity of a symbol in a region is given by the number of occurrences of this symbol in the string corresponding to that region);

(v) R_i, $1 \leq i \leq d$, are finite sets of *evolution rules* over V associated with the regions $1, 2, \ldots, d$ of μ; these evolution rules are of the forms $a \rightarrow v$ or $ca \rightarrow cv$, where c is a catalyst, a is an object from $V \setminus C$, and v is a string from $((V \setminus C) \times \{here, out, in\})^*$;

(vi) i_0 is a number between 1 and d and it specifies the *input* membrane of Π.

The membrane structure and the multisets represented by w_i, $1 \leq i \leq d$, in Π constitute the *initial configuration* of the system. A transition between configurations is governed by the application of the evolution rules which is done in parallel: all objects, from all membranes, which *can be* the subject of local evolution rules *have to* evolve simultaneously.

The application of a rule $u \rightarrow v$ in a region containing a multiset M results in subtracting from M the multiset identified by u, and then in adding the multiset identified by v. The objects can eventually be transported through membranes due to targets *in* and *out* (we usually omit the target *here*). We refer to [8] and [51] for further details and examples. According to [17], the P system Π is called *catalytic,* if every evolution rule involves a catalyst.

The system continues parallel steps until there remain no applicable rules in any region of Π; then the system halts.

Here we only consider accepting P systems where the multiset to be analysed is put into region i_0 together with w_{i_0} and accepted by a halting computation. The classes of all sets of α-vectors $(y_1, ..., y_\alpha)$ of non-negative integers accepted in that way by halting computations in P systems of this type with at most d membranes and the set of catalysts containing at most m elements are denoted by

$$N^\alpha OP_{acc}(d, cat_m, halt).$$

If we specify a set of terminal objects Σ, and only take into account the terminal symbols in the specified membrane i_0, we obtain *extended (accepting) P systems* of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_0),$$

and the corresponding class

$$N^\alpha EP_{acc}(d, cat_m, halt).$$

In [15] accepting P systems were introduced as P automata using finite states as accepting conditions, i.e., instead of the halting condition an input

is accepted if the P system reaches a configuration where the contents of (specified) membranes coincides with the multisets given by a finite state. In more detail, for a P system as defined above a final state over V is of the form $(f_1, ..., f_d)$ where each f_i, $1 \leq i \leq d$, either is a final multiset over V or (a special symbol denoted by) Λ; then the P system accepts its input (given in i_0) by this final state if during the computation a configuration is reached such that the contents of every membrane i with $f_i \neq \Lambda$ coincides with f_i. The special symbol Λ indicates that we do not care about the contents of membrane i if $f_i = \Lambda$. Hence, a P system accepting by final states is a construct of the form

$$\Pi = (V, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o, F),$$

where $V, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o$ are defined as above and F is a finite set of final states over V. The class of all sets of α-vectors $(y_1, ..., y_\alpha)$ of non-negative integers accepted in P systems with at most d membranes and the set of catalysts containing at most m elements by computations reaching a final state is denoted by

$$N^\alpha OP_{acc}(d, cat_m, \textit{final state}).$$

If we again specify a set of terminal objects Σ and only take into account the terminal symbols in the specified membrane i_0, we obtain *extended P (accepting by final states)* of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_0, F),$$

and the corresponding class

$$N^\alpha EP_{acc}(d, cat_m, \textit{final state}).$$

If in the variants of accepting P systems defined above only catalytic rules are used, we add the superscript *cat* thus obtaining the classes

$$N^\alpha OP^{cat}_{acc}(d, cat_m, Y), \; Y \in \{halt, \textit{final state}\}$$

and

$$N^\alpha_0 EP^{cat}_{acc}(d, cat_m, Y), Y \in \{halt, \textit{final state}\}.$$

Remark 23 *Note that due to the notations used in the preceding Chapters the above systems would be called initial.*

We now also consider accepting P systems analysing a sequence of letters taken from the environment during a halting computation or during a computation stopping by reaching a final state:

A *P automaton (of degree d, $d \geq 1$) with final states* is a construct of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, F),$$

where $V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, F$ are defined as above, only the catalytic rules in the skin membrane may also contain the target indication *come* for terminal symbols, i.e., they are of the form $ca \rightarrow cv$, where c is a catalyst, a is an object from $V \setminus C$, and v is a string from

$$(((V \setminus C) \times \{here, out, in\}) \cup (\Sigma \times \{come\}))^*.$$

The target indication *come* (compare with the P systems introduced in [59]) for a terminal symbol b_t means that such a symbol is taken in from the environment (in some sense like reading it from an external input tape).

We now consider the sequence of terminal symbols taken in from the environment during a computation having reached a final state as the accepted string(s). Then the class of all languages $\subseteq \Sigma^+$ accepted by P automata with final states having at most d membranes and the set of catalysts containing at most m elements is denoted by

$$P_{acc}(d, cat_m, final\ state).$$

If we again consider halting computations instead of computations reaching a final state, we obtain a *P automaton (of degree d, $d \geq 1$)* as a construct of the form

$$\Pi = (V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d),$$

where $V, \Sigma, C, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d$ are defined as above, we only omit the set of final states. Then the class of all languages $\subseteq \Sigma^+$ accepted (in halting computations) by such P automata having d membranes and the set of catalysts containing at most m elements is denoted by

$$P_{acc}(d, cat_m, halt).$$

If in the variants of P automata defined above only catalytic rules are used, we add the superscript *cat* thus obtaining the classes

$$P^{cat}_{acc}(d, cat_m, Y),\ Y \in \{halt, final\ state\}.$$

All the (catalytic) P automata taking their input from the environment as defined above can also be considered as devices for accepting sets of (vectors of) non-negative integers by interpreting the strings as representations of these (vectors of) non-negative integers; in that way we obtain the classes

$$N^{\alpha}IP_{acc}(d, cat_m, Y),\ Y \in \{halt, final\ state\}$$

and

$$N^{\alpha}IP_{acc}^{cat}(d, cat_m, Y),\ Y \in \{halt, final\ state\}.$$

7.4 Results

Although for P automata we have the minimal number of only one membrane, the number of catalysts depends on the number α of components of the vector of non-negative integers to be analysed.

For a register machine M with m registers, $m \geq 1$, let P be the program for M with n instructions $i_1, i_2, \ldots, i_n$ accepting $L \in N^{\alpha}RE$. Informally, each register a is represented by objects o_a playing the rôles of counter elements. The value of register a at each moment corresponds to the number of symbols o_a in the system.

There are also special objects p_j, $1 \leq j \leq n$; they play the rôle of program labels and their marked variants guide the simulation of the instruction labelled by p_j within the P automaton. The presence of the marked variants $p_j^{\langle h,1\rangle}$, $1 \leq h \leq m$, of the object p_j - for each catalyst there has to be such a marked variant to keep it busy - starts the sequence of operations corresponding to the instruction j. In contrast to the proofs given in [60] and then in [30] we now need only one catalyst for each register, because we use the concept of "paired catalysts": Together with the catalyst c_a associated with register a we also associate ("pair") another catalyst (we shall take $c_{a\ominus_m 1}$) which together with c_a will do the correct simulation of an instruction $j : (S(a), k, l) \in P$ in four steps; the remaining catalysts $c_{a\ominus_m h}$ with $2 \leq h < m$ are occupied by the marked variants of p_j, $p_j^{\langle h,l\rangle}$, $1 \leq l \leq 4$, during these four steps, and the $p_j^{\langle h,4\rangle}$ are eliminated in the fourth step, before in the next step the new multiset $p_k^{\langle 1,1\rangle}...p_k^{\langle m,1\rangle}$ or $p_l^{\langle 1,1\rangle}...p_l^{\langle m,1\rangle}$ of (marked) program labels appears. The simulation of an instruction $j : (A(a), k, k) \in P$ needs only one step. Finally, if the multiset $p_n^{\langle 1,1\rangle}...p_n^{\langle m,1\rangle}$ representing the final label n appears, these objects are also eliminated in one step, where after the computation halts if and only if it has been successful, i.e., no trap symbol # is present (after having been generated during the simulation of some subtract-instruction).

Theorem 24 *$N^{\alpha}RE = N^{\alpha}OP_{acc}(d, cat_{\alpha+2}, halt)$ for $d \geq 1$.*

Proof. Consider a (deterministic) register machine M as defined above with m registers (from the result stated in Corollary 21 we know that $m = \alpha + 2$ is sufficient). Now let P be a program which accepts a set $L \in N^{\alpha}RE$ such that the initial instruction has the label 1 and the halting instruction has the label n. The input values $x_1, ..., x_{\alpha}$ are expected to be in the first α registers. Moreover, without loss of generality, we may assume that at the beginning of a computation all the registers except eventually the registers 1 to α contain zero.

We construct the P system

$$\Pi = (V, C, [_1]_1, w, 1, R)$$

where

$$\begin{aligned}
V &= \{\#\} \cup \{c_i, c_i', c_i'' \mid 1 \leq i \leq m\} \cup \{o_k \mid 1 \leq k \leq m\} \cup \\
&\quad \left\{p_n^{\langle h,1\rangle} \mid 1 \leq h \leq m\right\} \cup \left\{p_j^{\langle h,1\rangle} \mid 1 \leq h \leq m,\ j : (A(a), k, k) \in P\right\} \cup \\
&\quad \left\{p_j^{\langle h,1\rangle} \mid 1 \leq h \leq m,\ j : (S(a), k, l) \in P\right\} \cup \\
&\quad \left\{p_j^{\langle h,l\rangle} \mid 2 \leq h < m,\ 1 \leq l \leq 4,\ j : (S(a), k, l) \in P\right\} \cup \\
&\quad \{p_j', p_j'', \bar{p}_j, \bar{p}_j', \bar{p}_j'', \hat{p}_j, \hat{p}_j', \hat{p}_j'' \mid j : (S(a), k, l) \in P\}, \\
C &= \{c_i \mid 1 \leq i \leq m\}, \\
w &= c_1 \ldots c_m p_1^{\langle 1,1\rangle} \ldots p_1^{\langle m,1\rangle}, \\
R &= \{x \to \# \mid x \in V \backslash \\
&\quad (C \cup \{o_k \mid 1 \leq k \leq m'\} \cup \{\bar{p}_j', \hat{p}_j' \mid j : (S(a), k, l) \in P\})\} \cup \\
&\quad \left\{c_{m \ominus_m h} p_n^{\langle h,1\rangle} \to c_{m \ominus_m h} \mid 1 \leq h \leq m\right\} \cup \\
&\quad \left\{c_{m \ominus_m h} p_j^{\langle h,1\rangle} \to c_{m \ominus_m h} \mid 1 \leq h < m,\ 1 \leq a \leq m,\right. \\
&\quad \left. j : (A(a), k, k) \in P\right\} \cup \\
&\quad \left\{c_m p_j^{\langle m,1\rangle} \to c_m p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle} o_a \mid 1 \leq a \leq m,\ j : (A(a), k, k) \in P\right\} \cup \\
&\quad \left\{c_{a \ominus_m h} p_j^{\langle h,l\rangle} \to c_{a \ominus_m h} p_j^{\langle h,l+1\rangle} \mid 2 \leq h < m,\ 1 \leq a \leq m,\right. \\
&\quad \left. 1 \leq l \leq 3,\ j : (S(a), k, l) \in P\right\} \cup \\
&\quad \{c_{a \ominus_m h} p_j^{\langle h,4\rangle} \to c_{a \ominus_m h} \mid 2 \leq h < m,\ 1 \leq a \leq m, \\
&\quad j : (S(a), k, l) \in P\} \cup
\end{aligned}$$

$$\Big\{c_a p_j^{\langle m,1\rangle} \to c_a \hat{p}_j \hat{p}'_j, c_a p_j^{\langle m,1\rangle} \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j,$$
$$c_a o_a \to c_a c'_a, c_a c'_a \to c_a c''_a, c_{a\ominus_m 1} c''_a \to c_{a\ominus_m 1},$$
$$c_a \hat{p}'_j \to c_a \#, c_{a\ominus_m 1} \hat{p}'_j \to c_{a\ominus_m 1} \hat{p}''_j, c_a \hat{p}''_j \to c_a p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle},$$
$$c_a \bar{p}_j \to c_a, c_{a\ominus_m 1} \bar{p}''_j \to c_{a\ominus_m 1} p''_j, c_{a\ominus_m 1} p''_j \to c_{a\ominus_m 1} p'_j,$$
$$c_a p'_j \to c_a p_l^{\langle 1,1\rangle} \ldots p_l^{\langle m,1\rangle} \mid 1 \le a \le m,\ j : (S(a), k, l) \in P\Big\} \cup$$
$$\Big\{c_{a\ominus_m 1} y \to c_{a\ominus_m 1} \mid y \in \Big\{p_j^{\langle 1,1\rangle}, \hat{p}_j, \bar{p}'_j\Big\}, 1 \le a \le m,$$
$$j : (S(a), k, l) \in P\Big\}.$$

Then for an arbitrary $(x_1, \ldots, x_\alpha) \in \mathbf{N}^\alpha$ the axiom of the corresponding system $\Pi_{(x_1,\ldots,x_\alpha)}$ is

$$c_1 \ldots c_m p_1^{\langle 1,1\rangle} \ldots p_1^{\langle m,1\rangle} o_1^{x_1} \ldots o_\alpha^{x_\alpha}.$$

The contents of register a, $1 \le a \le m$, is represented by the sum of the number of symbols o_a and conditional decrementing actions on this register are guarded by the pair of catalysts c_a and $c_{a\ominus_m 1}$.

The set of rules R depends on the instructions of P; the halting instruction as well as each add-instruction is simulated in one step, whereas each subtract-instruction is simulated in four steps; in more detail, the simulation works as follows:

1. Every simulation of a rule starts with the program labels $p_1^{\langle 1,1\rangle}, \ldots, p_1^{\langle m,1\rangle}$. The halting instruction eliminates the final labels $p_n^{\langle 1,1\rangle}, \ldots, p_n^{\langle m,1\rangle}$ by using the rules $c_{m\ominus_m h} p_n^{\langle h,1\rangle} \to c_{m\ominus_m h}$, $1 \le h \le m$; if the input is accepted, then only the catalysts remain in the skin membrane.

2. Each add-instruction $j : (A(a), k, k) \in P$, $1 \le a \le m$, is simulated in one step by using the catalytic rules $c_{m\ominus_m h} p_j^{\langle h,1\rangle} \to c_{m\ominus_m h}$, $1 \le h < m$, as well as $c_m p_j^{\langle m,1\rangle} \to c_m p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle} o_a$. Observe that by definition $a \ominus_m m = a$ for all a with $1 \le a \le m$.

3. Each subtract-instruction $j : (S(a), k, l) \in P$ is simulated in four steps. We have to distinguish between two cases depending on the contents of register a; in both cases the catalysts $c_{a\ominus_m h}$, $2 \le h < m$, are busy with the objects $p_j^{\langle h,l\rangle}$, $1 \le l \le 4$; the objects $p_j^{\langle h,4\rangle}$ finally are eliminated in the fourth step. The main part of the simulation is accomplished by the catalyst c_a and its "paired companion" $c_{a\ominus_m 1}$, which is also shown in the following table (where the rules in the brackets $(\langle c_a \hat{p}'_j \to c_a \#\rangle$

as well as $\langle c_a o_a \to c_a c'_a \rangle$) are those which should not be applied at that stage of the simulation; their application (only the application of the rule $\langle c_a \hat{p}'_j \to c_a \# \rangle$ may even be forced due to maximal parallelism) leads to the introduction of the failure symbol # (directly or one step later) and therefore to a non-halting computation):

simulation of the subtract-instruction $j : (S(a), k, l)$ if	
a. register a is not empty	**b.** register a is empty
$c_a p_j^{\langle m,1 \rangle} \to c_a \hat{p}_j \hat{p}'_j$ $c_{a \ominus_m 1} p_j^{\langle 1,1 \rangle} \to c_{a \ominus_m 1}$	$c_a p_j^{\langle m,1 \rangle} \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$ $c_{a \ominus_m 1} p_j^{\langle 1,1 \rangle} \to c_{a \ominus_m 1}$
$c_a o_a \to c_a c'_a$ $\langle c_a \hat{p}'_j \to c_a \# \rangle$ $c_{a \ominus_m 1} \hat{p}_j \to c_{a \ominus_m 1}$	$c_a \bar{p}_j \to c_a$ $c_{a \ominus_m 1} \bar{p}''_j \to c_{a \ominus_m 1} p''_j$
$c_a c'_a \to c_a c''_a$ $c_{a \ominus_m 1} \hat{p}'_j \to c_{a \ominus_m 1} \hat{p}''_j$	 $\langle c_a o_a \to c_a c'_a \rangle$ $c_{a \ominus_m 1} p''_j \to c_{a \ominus_m 1} p'_j$
$c_a \hat{p}''_j \to c_a p_k^{\langle 1,1 \rangle} \ldots p_k^{\langle m,1 \rangle}$ $c_{a \ominus_m 1} c''_a \to c_{a \ominus_m 1}$	$c_a p'_j \to c_a p_l^{\langle 1,1 \rangle} \ldots p_l^{\langle m,1 \rangle}$ $c_{a \ominus_m 1} \bar{p}'_j \to c_{a \ominus_m 1}$

(a) We non-deterministically assume that the contents of register a is not empty; we start with the rules $c_a p_j^{\langle m,1 \rangle} \to c_a \hat{p}_j \hat{p}'_j$ together with $c_{a \ominus_m 1} p_j^{\langle 1,1 \rangle} \to c_{a \ominus_m 1}$. In the second step, the number of symbols o_a is decremented by using the rule $c_a o_a \to c_a c'_a$; if in contrast to our choice, no such symbol o_a is present (i.e., the contents of the register represented by the number of symbols o_a is empty), then by the enforced application of the rule $c_a \hat{p}'_j \to c_a \#$ the trap symbol # is introduced, which causes a non-halting computation due to the rule $\# \to \#$. If $\hat{p}'_j$ could wait until being used in the third step by the rule $c_{a \ominus_m 1} \hat{p}'_j \to c_{a \ominus_m 1} \hat{p}''_j$, then the simulation will be successful: In the second step, $c_{a \ominus_m 1}$ is used in the rule $c_{a \ominus_m 1} \hat{p}_j \to c_{a \ominus_m 1}$, and in the third step c_a is used in the rule $c_a c'_a \to c_a c''_a$. We finish with the application of the rules $c_a \hat{p}''_j \to c_a p_k^{\langle 1,1 \rangle} \ldots p_k^{\langle m,1 \rangle}$ and $c_{a \ominus_m 1} c''_a \to c_{a \ominus_m 1}$.

(b) For the other case, we non-deterministically assume that the contents of register a is empty; we start with the two rules $c_a p_j^{\langle m,1 \rangle} \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$ and $c_{a \ominus_m 1} p_j^{\langle 1,1 \rangle} \to c_{a \ominus_m 1}$. In the second step, we are forced to use the two rules $c_a \bar{p}_j \to c_a$ and $c_{a \ominus_m 1} \bar{p}''_j \to c_{a \ominus_m 1} p''_j$ in order not to introduce the trap symbol #. In the third step, we only use $c_{a \ominus_m 1} p''_j \to c_{a \ominus_m 1} p'_j$ and finish with applying the two rules $c_a p'_j \to c_a p_l^{\langle 1,1 \rangle} \ldots p_l^{\langle m,1 \rangle}$ and $c_{a \ominus_m 1} \bar{p}'_j \to c_{a \ominus_m 1}$ in the fourth step. In

the third step the catalyst c_a is not used if our non-deterministic choice has been correct, i.e., if there is no symbol o_a present in the skin membrane; otherwise, the rule $c_a o_a \rightarrow c_a c'_a$ has to be applied in the third step, but in this case both c'_a and p'_j would need the catalyst c_a in the fourth step of the simulation in order not to be sent to the trap symbol #.

Any other behavior of the system as the one described above for the correct simulation of the instructions of P by the rules in R leads to the appearance of the trap object # within the system, hence, the system never halts.

(We should like to mention that at any time c_a can be used in the rule $c_a o_a \rightarrow c_a c'_a$, but carried out at the wrong time, the application of this rule will immediately cause the introduction of the trap symbol #.)

It follows from the description given above that after each simulation of an instruction the number of objects o_a equals the contents of register a, $1 \leq a \leq m$. Hence, after having simulated the instruction $Halt$ and halting the system, the only objects remaining within the system are the m catalysts in the skin membrane; according to the result about register machines stated in Proposition 21, $m = \alpha + 2$ and therefore $\alpha + 2$ catalysts are enough. □

For accepting P systems with final states, we can immediately take over the construction given in the preceding proof:

Corollary 25 *$N^\alpha RE = N^\alpha OP_{acc}(d, cat_{\alpha+2}, final\ state)$ for $d \geq 1$.*

Proof. The only difference to the P system constructed in Theorem 24 is that we have to define the final state for successful computations, which simply is the contents of the skin membrane at the end of a halting computation, i.e., $c_1 \ldots c_m$. Hence, taking $F = \{(c_1 \ldots c_m)\}$ we obtain the P system with final states Π' is $(\Pi, \{(c_1 \ldots c_m)\})$, where Π is the P system constructed in the proof of Theorem 24. □

In catalytic systems we only need one more catalyst for the rules handling the trap symbol #:

Corollary 26 *For every $d \geq 1$ we have*

$$\begin{aligned} N^\alpha RE &= N^\alpha OP^{cat}_{acc}(d, cat_{\alpha+3}, halt) \\ &= N^\alpha OP^{cat}_{acc}(d, cat_{\alpha+3}, final\ state) \end{aligned}$$

Proof. Consider the

1. (halting) catalytic accepting P system

$$\Pi^{cH} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1]_1, w, R_C, 1)$$

2. as well as the catalytic accepting P system with final states

$$\Pi^{cF} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1]_1, w, R_C, 1, F),$$

respectively, with at most $\alpha + 3$ catalysts and with the objects $o_a \in V$ satisfying the following conditions: For any arbitrary $(x_1, ..., x_\alpha) \in \mathbf{N}^\alpha$, denote

1. $\Pi^{cH}_{(x_1,...,x_\alpha)} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, R_C, 1)$ and
2. $\Pi^{cF}_{(x_1,...,x_\alpha)} = (V \cup \{c_0\}, C \cup \{c_0\}, [_1]_1, wo_1^{x_1}...o_\alpha^{x_\alpha}, R_C, 1, F)$,

respectively. The system

1. $\Pi^{cH}_{(x_1,...,x_\alpha)}$ halts,
2. $\Pi^{cF}_{(x_1,...,x_\alpha)}$ reaches a final state,

respectively, if and only if $(x_1, ..., x_\alpha)$ is accepted and

1. in the halting computation or
2. in the final state

respectively, in the skin membrane only the catalysts remain.

Then the rules in R_C are obtained from the rules in R constructed in the proof of Theorem 24 by just replacing the rules in

$$\left\{x \to \# \mid x \in V \setminus \left(C \cup \left\{\vec{p}_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\} \cup \{o_k \mid 1 \leq k \leq m\}\right)\right\}$$

with the rules in

$$\left\{c_0 x \to c_0 \# \mid x \in V \setminus \left(C \cup \left\{\vec{p}_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\} \cup \{o_k \mid 1 \leq k \leq m\}\right)\right\}$$

using the additional catalyst c_0. □

The proofs of the following results immediately follow from preceding proofs:

Theorem 27 *For every $d \geq 1$, we have*

$$\begin{aligned} N^{\alpha}RE &= N^{\alpha}XP_{acc}(d, cat_{\alpha+2}, Y) \\ &= N^{\alpha}XP_{acc}^{cat}(d, cat_{\alpha+3}, Y) \end{aligned}$$

for every $X \in \{E, I\}$ *and* $Y \in \{halt, final\ state\}$.

For the simplest case of $\alpha = 1$, therefore the maximal number of catalysts needed for accepting languages from $N^{\alpha}RE$ by P automata is 3 and by catalytic P automata is 4.

Theorem 28 *For every* $d \geq 1$, *we have*

$$\begin{aligned} RE &= P_{acc}(d, cat_2, halt) \\ &= P_{acc}(d, cat_2, final\ state). \end{aligned}$$

Proof. We first prove the inclusion $RE \subseteq P_{acc}(1, cat_2, halt)$. In the same way as in the proof of Theorem 24 we simulated the operations on the two registers allowing for decrementation we now simulate the operations on the two counters of the 2-counter automaton from Proposition 22:

$$M = (n, \Sigma_B, P, i, h),$$

where $\Sigma = \{b_t \mid 1 \leq t \leq s\}$, $s \geq 1$. Hence, let us define

$$\Pi = (V, \Sigma, \{c_1, c_2\}, [_1]_1, w, R, \emptyset)$$

where R is constructed in a similar way as R' in the proof of Theorem 24, except that now we also have to consider instructions $j : (read, k_1, ..., k_s, l)$, which are simulated (in a non-deterministic way) by the following rules:

$$\begin{aligned} &c_1\tilde{p}_j \to c_1,\ c_1 b_t \to c_1 \\ &c_2 p_j \to c_2 p'_{j,t}(b_t, come),\ c_2 p'_{j,t} \to c_2 p_{k_t}\tilde{p}_{k_t},\ 1 \leq t \leq s, \\ &c_2 p_j \to c_2 p_l \tilde{p}_l. \end{aligned}$$

In sum, we obtain the following P automaton Π:

$$\begin{aligned} V &= \{\#\} \cup \{c_1, c'_1, c''_1, c_2, c'_2, c''_2\} \cup \{o_1, o_2\} \cup \\ &\quad \{p_j, \tilde{p}_j, p''_j, \bar{p}_j, \bar{p}'_j, \bar{p}''_j, \hat{p}_j, \hat{p}'_j, \hat{p}''_j, p'_{j,t} \mid j : (S(a), k, l) \in P\} \cup \\ &\quad \{p_j, \tilde{p}_j \mid j : (A(a), k, l) \in P\} \cup \Sigma \cup \{B\}, \\ C &= \{c_1, c_2\}, \\ w &= c_1 c_2 p_1 \tilde{p}_1, \end{aligned}$$

$$
\begin{aligned}
R \;=\; & \left\{x \to \# \mid x \in V \setminus \left(C \cup \{o_1, o_2\} \cup \left\{\vec{p}'_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\}\right)\right\} \cup \\
& \{c_1 p_n \to c_1, c_2 \tilde{p}_n \to c_2\} \cup \\
& \{c_1 \tilde{p}_j \to c_1 \mid j : (A(a), k, l) \in P\} \cup \\
& \{c_2 p_j \to c_2 p_k \tilde{p}_k o_a, c_2 p_j \to c_2 p_l \tilde{p}_l o_a \mid \\
& a \in \{1,2\},\ j : (A(a), k, l) \in P\} \cup \\
& \{c_a p_j \to c_a \hat{p}_j \hat{p}'_j, c_a p_j \to c_a \vec{p}_j \vec{p}'_j \vec{p}''_j, \\
& c_a o_a \to c_a c'_a, c_a c'_a \to c_a c''_a, c_{3-a} c''_a \to c_{3-a}, \\
& c_a \hat{p}'_j \to c_a \#, c_{3-a} \hat{p}'_j \to c_{3-a} \hat{p}''_j, c_a \hat{p}''_j \to c_a p_k \tilde{p}_k, \\
& c_a \vec{p}_j \to c_a, c_{3-a} \vec{p}''_j \to c_{3-a} p''_j, c_{3-a} p''_j \to c_{3-a} p'_j, \\
& c_a p'_j \to c_a p_l \tilde{p}_l \mid a \in \{1,2\},\ j : (S(a), k, l) \in P\} \cup \\
& \{c_{3-a} y \to c_{3-a} \mid y \in \{\tilde{p}_j, \hat{p}_j, \vec{p}'_j\}, a \in \{1,2\}, \\
& j : (S(a), k, l) \in P\} \cup \\
& \{c_1 \tilde{p}_j \to c_1 \mid j : (read, k_1, ..., k_s, l) \in P\} \cup \\
& \{c_2 p_j \to c_2 p'_{j,t}(b_t, come),\ c_2 p'_{j,t} \to c_2 p_{k_t} \tilde{p}_{k_t},\ c_1 b_t \to c_1 \mid \\
& j : (read, k_1, ..., k_s, l) \in P,\ 1 \le t \le s,\} \cup \\
& \{c_2 p_j \to c_2 p_l \tilde{p}_l \mid j : (read, k_1, ..., k_s, l) \in P\}.
\end{aligned}
$$

For the corresponding P automata with final states we use the final state $c_1 c_2$, which immediately proves $RE \subseteq P_{acc}(1, cat_2, final\ state)$. □

For the catalytic variants we need one more catalyst (compare with Corollary 26):

Corollary 29 *For every $d \geq 1$, we have*

$$
\begin{aligned}
RE &= P^{cat}_{acc}(d, cat_3, halt) \\
&= P^{cat}_{acc}(d, cat_3, final\ state).
\end{aligned}
$$

Proof. As in the proof of Corollary 26 we replace the rules in

$$\left\{x \to \# \mid x \in V \setminus \left(C \cup \{o_1, o_2\} \cup \left\{\vec{p}'_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\}\right)\right\}$$

in the sets of rules constructed in Theorem 28 with the rules in

$$\left\{c_0 x \to c_0 \# \mid x \in V \setminus \left(C \cup \{o_1, o_2\} \cup \left\{\vec{p}'_j, \hat{p}'_j \mid j : (S(a), k, l) \in P\right\}\right)\right\}$$

using the additional catalyst c_0. □

7.5 Conclusion

We have considered accepting P systems where the multiset to be analysed is put into a designated input region together with the axiom. In extended (accepting) P systems we specified a terminal alphabet and only considered the terminal symbols contained in the input membrane (in effect this means that we ignored the catalysts, which of course can never be eliminated). Although for accepting P systems of the above types we have the minimal number of only one membrane, the number of catalysts depends on the number α of components of the vector of non-negative integers to be analysed.

We then investigated P automata accepting string languages; we proved that every recursively enumerable string language can be accepted by these systems (by halting or by final state) with two catalysts in only one membrane.

For the purely catalytic variants of all these systems, one more catalyst is necessary.

Chapter 8

ω-P Automata with Communication Rules

In this Chapter we consider ω-P automata based on the model of P systems with membrane channels, especially for the special variant using only antiport rules of specific types. The main problem we face when dealing with ω-words is the fact that usually in P systems successful computations are assumed to be the halting computations, whereas failing computations are made non-halting by introducing a failure symbol together with rules allowing for infinite computations. On the other hand, computations analysing infinite words have to be infinite and, in contrast to the case of finite words, failing computations have to stop. We shall show that for any well-known variant of acceptance mode for ω-Turing machines we can effectively construct an ω-P automaton simulating the computations of the ω-Turing machine.

After some preliminary definitions we give a short introduction to ω-words and ω-Turing machines, especially focussing on the different acceptance modes to be found in the literature. In the following Section we introduce ω-P automata (with membrane channels or with antiport rules only) and then prove our main result showing that for any well-known variant of acceptance mode for ω-Turing machines we can effectively construct an ω-P automaton with two membranes simulating the computations of the ω-Turing machine; moreover, ω-P automata of a very restricted form (with only one membrane) exactly characterize the family of ω-regular languages. A short summary of results and an outlook on future research topics conclude the Chapter which was presented in [33] (also see [32], [34]).

8.1 Preliminary Definitions

Given a word $w \in T^+$, a sequence $q_0 q_1 ... q_n$ is called a *run* of M on w if and only if $w = a_1 ... a_n$, $a_i \in T$, $1 \leq i \leq n$, and $q_i \in \delta(q_{i-1}, a_i)$; the run $q_0 q_1 ... q_n$ on w is called *successful* if and only if $q_n \in F$. The (λ-free) language accepted by M is the set of all strings $w \in T^+$ which allow for a successful run of M on w.

Remark 30 *Based on the results established in [43], we know that the actions of a Turing machine can be simulated by a register machine in only two registers using a z-ary representation (where $z+1$ is the cardinality of the tape alphabet) of the left- and right-hand side of the Turing tape with respect to the current position of the read/write-head on the working tape of the Turing machine. Using a prime number encoding in the two registers, even all necessary operations for the simulation of a Turing machine can be simulated by a register machine with only two registers. For the purposes of this Chapter, we need a more "relaxed" representation of the actions and the contents of the working tape of a Turing machine: We only store the contents of the left- and right-hand side of the working tape with respect to the current position of the read/write-head and simulate the actions on the working tape in these two registers; on the other hand, the current state of the Turing machine is stored in a separate additional register using a unary encoding.*

8.2 ω-Turing Machines

We consider the space X^ω of infinite strings (ω-words) on a finite alphabet of cardinality ≥ 2. For $w \in X^*$ and $b \in X^\omega$ let $w \cdot b$ be their *concatenation.* This concatenation product extends in an obvious way to subsets $W \subseteq X^*$ and $B \subseteq X^\omega$. Subsets of X^ω are called ω-languages. For an ω-word ξ and every $n \in \mathbf{N}$, ξ/n denotes the prefix of ξ of length n.

8.2.1 Variants of Acceptance

In the models found in most papers in the literature (e.g., see the recent surveys [20] or [61]), the acceptance of ω-languages by Turing machines is determined by the behaviour of the Turing machines on the input tape as well as by specific final state conditions well-known from the acceptance of ω-languages by finite automata.

For Turing machines accepting infinite strings (ω-words), in literature different *variants of acceptance* can be found:

Type 1 The approach described in [61] and [62] does not take into consideration the behaviour of the Turing machine on its input tape. Acceptance is based solely on the infinite sequence of internal states the machine runs through during its infinite computation. Thus the machine may base its decision on a finite part of the whole infinite input.

Type 2 For X-automata Engelfriet and Hoogeboom (see [20]) require that, in addition to the fulfillment of certain conditions on the infinite sequence of internal states in order to accept an input, the machine has to read the whole infinite input tape. Thus, besides blocking as for Type 1, machines have a further possibility to reject inputs.

Type 3 The most complicated type of acceptance for Turing machines was introduced by Cohen and Gold (see [12] and [13]). In addition to Type 2 they require that the machine scans every cell of the input tape only finitely many times; this behaviour is termed as having a *complete non-oscillating run.*

8.2.2 ω-Turing Machines - Definitions

In order to be in accordance with the X-automata of Engelfriet and Hoogeboom we consider Turing machines $M = (X, \Gamma, Q, q_0, P)$ with a separate input tape on which the read-only-head moves only to right, a working tape, X as its input alphabet, Γ as its worktape alphabet, Q the finite set of internal states, q_0 the initial state, and the relation

$$P \subseteq Q \times X \times \Gamma \times Q \times \{0, +1\} \times \Gamma \times \{-1, 0, +1\}$$

defining the next configuration.

Here $(q, x_0, x_1; p, y_0, y_1, y_2) \in P$ means that if M is in state $q \in Q$, reads $x_0 \in X$ on its input tape and $x_1 \in \Gamma$ on its worktape, M changes its state to $p \in Q$, moves its head on the input tape to the right if $y_0 = +1$ or if $y_0 = 0$ does not move the head, and for $y_1 \in \Gamma$ and $y_2 \in \{-1, 0, +1\}$ the machine M writes y_1 instead of x_1 in its worktape and moves the head on this tape to the left, if $y_2 = -1$, to the right, if $y_2 = +1$, or does not move it, if $y_2 = 0$.

Unless stated otherwise, in the sequel we shall assume that our accepting devices be fully defined, i.e., for every situation (q, x_0, x_1) in $Z \times X^2$ the transition relation R has to contain at least one (exactly one, if the device is deterministic) move $(q, x_0, x_1; p, y_0, y_1, y_2)$.

For some sequence $x \in X^\omega$, let x be the input of the Turing machine M. We call a sequence $z \in Q^\omega$ of states a *run* of M on x if z is the sequence of states the Turing machine runs through in its (some of its, if the machine is non-deterministic) computation(s) with input x.

8.2.3 ω-languages Accepted by ω-Turing Machines

We say that an input sequence $x \in X^\omega$ is accepted by M according to condition (mode) C if there is a run z of M on x such that z satisfies C. In the sequel we shall consider the following conditions using the notation of Engelfriet and Hoogeboom:

Let $\alpha : Q^\omega \to 2^Q$ be a mapping which assigns to every ω-word $\zeta \in Q^\omega$ a subset $Q' \subseteq Q$, and let $R \subseteq 2^Q \times 2^Q$ be a relation between subsets of Q. We say that a pair (M, Y) where $Y \subseteq 2^Q$ accepts an ω-word $x \in X^\omega$ if and only if

$$\exists Q' \exists z \left(Q' \in Y \wedge z \text{ is a run of } M \text{ on } x \wedge \left(\alpha\left(z\right), Q'\right) \in R\right).$$

If Y consists of only one subset of Q, i.e., $Y = \{F\}$ for some $F \subseteq Q$, then we usually write (M, F) instead of $(M, \{F\})$.

For an ω-word $z \in Q^\omega$ let

$$ran\left(z\right) := \{v \mid v \in Q \wedge \exists i \left(i \in N \setminus \{0\} \wedge z\left(i\right) = v\right)\}$$

be the *range* of z (considered as a mapping $z : N \setminus \{0\} \to Q$), that is, the set of all letters occurring in z, and let

$$inf\left(z\right) := \left\{v \mid v \in Q \wedge z^{-1}\left(v\right) \text{ is infinite}\right\}$$

be the *infinity set* of z, that is, the set of all letters occurring infinitely often in z. As relations R we shall use $=$, $\subseteq$ and $\sqcap$ ($Z' \sqcap Z''$ means $Z' \cap Z'' \neq \emptyset$).

We obtain the six types of acceptance presented in the following table:

(α, R)	type of acceptance	meaning
$(ran, \sqcap)$	1-acceptance	at least once
$(ran, \subseteq)$	1′-acceptance	everywhere
$(ran, =)$		
$(inf, \sqcap)$	2-acceptance	infinitely often
$(inf, \subseteq)$	2′-acceptance	almost everywhere
$(inf, =)$	3-acceptance	

Theorem 31 *(e.g., see [20]) For all $\alpha \in \{ran, inf\}$ and all $R \in \{\subseteq, \sqcap, =\}$ the class of ω-languages accepted according to type 2 in the (α, R)-mode by non-deterministic ω-Turing machines collapses and coincides with the class of Σ_1^1-definable ω-languages over X.*

An ω-language F is referred to as Σ_1^1-definable provided

$$F = \{\xi \mid \exists \eta \, (\eta \in X^\omega \wedge \forall n \exists m \, ((n, \eta/m, \xi/m) \in M_F))\}$$

for some recursive relation $M_F \subseteq N \times X^* \times X^*$.

8.2.4 Finite ω-automata

A *regular ω-language* is a finite union of ω-languages of the form UV^ω, where U and V are regular languages.

A finite ω-automaton is an ω-Turing machine using only the input tape. The class of ω-languages 3-accepted by deterministic finite ω-automata coincides with the class of ω-regular languages; ω-languages 2-accepted by non-deterministic finite ω-automata give another characterization of the family of regular ω-languages, too. In fact, such a (non-deterministic) finite ω-automaton can be described as (non-deterministic) finite automaton $M = (Q, T_M, \delta, q_0, F)$.

Given an ω-word $\xi \in T_M^\omega$, $\xi = a_1 a_2 ...$, $a_i \in T_M$ for all $i \geq 1$, a run of M on ξ is an infinite sequence $s \in Q^\omega$, $s = q_0 q_1 q_2 ...$, of states such that $q_i \in \delta \, (q_{i-1}, a_i)$ for all $i \geq 1$; the run s is called successful (in the sense of 2-acceptance) if $inf \, (s) \cap F \neq \emptyset$. The ω-language of M is the set of all $\xi \in T_M^\omega$ which allow for a successful run of M on ξ.

8.3 ω-P Automata

In the case of ω-words, we not only have to take care of the (now infinite) sequence of terminal symbols taken from the environment, yet we also have to check the acceptance condition defined via the final states (which is done using the second membrane and turns out to be much more complicated than using the halting condition in the case of string languages).

An ω-P automaton is a construct

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

as defined in Section 5.1, but now used for analysing infinite sequences of terminal symbols and accepting these ω-words according to specific accepting conditions as defined in Subsection 8.2.3 with respect to a given set $Y \subseteq 2^Q$ of sets of final states F.

8.3.1 ω-P Automata with Antiport Rules

An ω-P automaton with antiport rules is an ω-P automaton as specified above but using only antiport rules (compare with the definitions in Subsection 5.3).

The main result of this Chapter is established in the following Theorem and its proof is based on the observations described in Remark 30:

Theorem 32 *Let $L \subseteq \Sigma^\omega$ be an ω-language accepted by an ω-Turing machine in the acceptance mode (α, R), for some $\alpha \in \{ran, inf\}$ and some $R \in \{\subseteq, \sqcap, =\}$.*

Then we can effectively construct an ω-P automaton with antiport rules in two membranes that simulates the actions of the Turing machine and accepts L in the same acceptance mode (α, R) and only uses rules of the form $(x, out; y, in)$ with the radius of the rules only being $(2,1)$ or $(1,2)$.

Proof (sketch). Let $L \subseteq \Sigma^\omega$ be an ω-language accepted by an ω-Turing machine

$$M = (\Sigma, \Gamma, Q, q_0, P)$$

together with $Y \subseteq 2^Q$ in the acceptance mode (α, R). We now elaborate the main ideas for constructing an ω-P automaton

$$\Pi = (V, \Sigma, [_1[_2]_2]_1, q_0', fg, R_1, R_2, Y')$$

with antiport rules in two membranes that simulates the actions of the Turing machine and accepts L in the same acceptance mode (α, R) and only uses rules of the form $(x, out; y, in)$ with the radius of the rules only being $(2,1)$ or $(1,2)$.

In fact, we use the register machine representation of the ω-Turing machine M as described in Remark 30, i.e., the actions of M are simulated by a register machine R_M in only two registers using a z-ary representation (where $z+1$ is the cardinality of the tape alphabet) of the left- and right-hand side of the Turing tape with respect to the current position of the read/write-head on the working tape of M; the current state of M is stored in a separate additional register using a unary encoding. The more complex steps of M can be simulated by Π in several substeps:

Reading a new symbol $b \in \Sigma$ on the input tape of M can be simulated by rules of the form $(q, out; (q, b, r)\, b, in)$ and $((q, b, r)\, b, out; r, in)$ in the ω-P automaton Π, where $q, r \in V \setminus T$. Modifying the left- and right-hand side of the Turing tape as well as changing the current state of M is simulated by actions of the register machine R_M. The actions of the simulating register machine R_M itself can easily be simulated by rules of the following types in region 1 (i.e., by rules in R_1), the contents of register i being represented by the corresponding number of symbols a_i in region 1 of Π:

- An Add-instruction $j : (A(i), k, l)$ is simulated by the two 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:

  $(ja_i, out; k, in)$
  $(j, out; j'j'', in)$
  $(j'a_i, out; \#, in)$
  $\left(j'', out; \hat{j}\hat{j}', in\right)$
  $\left(\hat{j}\hat{j}', out; \hat{j}'', in\right)$
  $\left(j'\hat{j}'', out; l, in\right)$

  In the case where the decrementation of register i is possible, we simply use the rule $(ja_i, out; k, in)$. In the other case, the rules $(j, out; j'j'', in)$, $(j'', out; \hat{j}\hat{j}', in)$, $(\hat{j}\hat{j}', out; \hat{j}'', in)$, and $(j'\hat{j}'', out; l, in)$ should be used; the condition of maximal parallelism guarantees that the antiport rule $(j'a_i, out; \#, in)$ is applied in parallel with $(j'', out; \hat{j}\hat{j}', in)$, if a symbol a_i is present although we have assumed the contrary, which leads to a halting computation, because then the symbol j' needed in the rule $(j'\hat{j}'', out; l, in)$ is missing. Only if in the current configuration no symbol a_i is present in the skin membrane, the object j' can wait the two steps where we apply the rules $(j'', out; \hat{j}\hat{j}', in)$ and $(\hat{j}\hat{j}', out; \hat{j}'', in)$ for being used in the rule $(j'\hat{j}'', out; l, in)$ together with the symbol $\hat{j}''$ appearing after these two intermediate steps.

- Observe that as we consider only infinite computations, we shall never reach the halt instruction of R_M.

Whenever the ω-P automaton Π has simulated one step of the ω-Turing machine M in the way described above, before continuing with the simulation of the register machine instruction labelled by r, Π starts an intermediate procedure for exposing an object that represents the current state of M in region 2 by importing an object $\bar{r}$ instead of the object r, which then will only appear at the end of this intermediate procedure:

We start with the rule $\left(\bar{r}, out; \bar{r}^{(0)}\bar{r}', in\right)$ in region 1 and then check the current state of M represented in the register machine simulation by the number of symbols a_3, i.e., we apply the rules $\left(\bar{r}^{(i-1)}a_3, out; \bar{r}^{(i)}, in\right)$, $1 \leq i \leq m$ (where m is the number of states in M), until we have sent out all symbols a_3 and may continue with the rule $\left(\bar{r}^{(n)}, out; \bar{h}\,[n]\,, in\right)$. Then in region 2 the rule $(fg, out; [n]\,, in)$ has to be used taking in the object $[n]$,

which represents the current state of M; hence, in region 2 now only the object $[n]$ representing a state of M is present. Then the correct way to continue (in region 1) is to use only the rule $(g\bar{h}, out; \bar{h}', in)$; if we also have to use the rule $(fa_3, out; \#, in)$ (indicating that we have used the rule $(\bar{r}^{(n)}, out; \bar{h}[n], in)$ too early and thus chosen an incorrect value for the state) then the symbol f will be missing in some further steps to continue the computation in Π. After having used the rule $(\bar{h}', out; h\bar{h}'', in)$ in region 1, in region 2 the rule $([n], out; fh, in)$ can be used (provided that the symbol f is still present in region 1). Then we continue in region 1 with regaining the number of symbols a_3 representing state $[n]$ by using the rules $(\bar{r}'[n], out; \tilde{r}^{(n)}, in)$ and $(\tilde{r}^{(i)}, out; \tilde{r}^{(i-1)}a_3, in)$, $1 \leq i \leq m$, as well as $(\tilde{r}^{(0)}\bar{h}'', out; \tilde{r}, in)$, $(\tilde{r}, out; \tilde{r}'\tilde{h}'', in)$, and $(\tilde{h}'', out; \tilde{h}\tilde{h}', in)$. Now by using the rules $(h, out; \tilde{h}\tilde{h}', in)$ in region 2, $(\tilde{r}'h, out; \tilde{r}'', in)$, $(\tilde{r}'', out; \bar{r}''g, in)$ in region 1 and $(\tilde{h}\tilde{h}', out; g, in)$ again in region 2, the original contents fg in region 2 is regained. Finally, by applying the rules $(\bar{r}''\tilde{h}', out; \bar{\bar{r}}, in)$ and $(\bar{\bar{r}}\tilde{h}, out; r, in)$ in region 1, we get the object r representing the label of the register machine where we have to continue our simulation of M.

In sum, for this special subprocedures for expressing the state $[n]$ in region 2 we have the following rules in membrane 1 and membrane 2:

rules in R_1	rules in R_2
$(\bar{r}, out; \bar{r}^{(0)}\bar{r}', in)$	
$(\bar{r}^{(i-1)}a_3, out; \bar{r}^{(i)}, in)$, $1 \leq i \leq m$	
$(\bar{r}^{(n)}, out; \bar{h}[n], in)$, $1 \leq n \leq m$	
	$(fg, out; [n], in)$, $1 \leq n \leq m$
$(g\bar{h}, out; \bar{h}', in)$, $(fa_3, out; \#, in)$	
$(\bar{h}', out; h\bar{h}'', in)$	
	$([n], out; fh, in)$, $1 \leq n \leq m$
$(\bar{r}'[n], out; \tilde{r}^{(n)}, in)$	
$(\tilde{r}^{(i)}, out; \tilde{r}^{(i-1)}a_3, in)$, $1 \leq i \leq m$	
$(\tilde{r}^{(0)}\bar{h}'', out; \tilde{r}, in)$	
$(\tilde{r}, out; \tilde{r}'\tilde{h}'', in)$	
$(\tilde{h}'', out; \tilde{h}\tilde{h}', in)$	

$$\left(h, out; \tilde{h}\tilde{h}', in\right)$$

$$(\tilde{r}'h, out; \tilde{r}'', in)$$
$$(\tilde{r}'', out; \bar{r}''g, in)$$

$$\left(\tilde{h}\tilde{h}', out; g, in\right)$$

$$\left(\bar{r}''\tilde{h}', out; \bar{\bar{r}}, in\right)$$
$$\left(\bar{\bar{r}}\tilde{h}, out; r, in\right)$$

In order to obtain the corresponding set of final states F' in Y' for the ω-P automaton with antiport rules Π from a given set $F \in Y$ of final states for M, we observe that in region 2 of the ω-P automaton Π only the multisets fg, fh, $f\tilde{h}\tilde{h}'$, and the symbols $[n]$ for some states of M may appear. Hence, for the acceptance modes $(ran, \sqcap)$ and $(inf, \sqcap)$ we may simply take $F' = \{(\Lambda, l) \mid l \in F\}$; for the other acceptance modes we also have to take into account the "constants" fg, fh, and $f\tilde{h}\tilde{h}'$, i.e., we have to take $F' = \left\{(\Lambda, l) \mid l \in F \cup \left\{fg, fh, f\tilde{h}\tilde{h}'\right\}\right\}$. These constructions for the sets of final states depending on the accepting mode conclude the proof. □

8.3.2 Finite ω-P Automata

As in the case of P automata, very special restrictions on ω-P automata yield the concept of finite ω-P automata (compare with the definition given in Subsection 5.3):

A *finite ω-P automaton* is an ω-P automaton

$$\Pi = (V, T, [_1]_1, w_1, R_1, F)$$

with only one membrane such that

1. the axiom w_1 is a non-terminal symbol (in the case of finite ω-P automata simply called "state"), i.e., $w_1 \in V \setminus T$;

2. the rules in R_1 are of the forms $(q, out; pa, in)$ and $(pa, out; r, in)$, where a is a terminal symbol in T and p, q, r are non-terminal symbols (states) in $V \setminus T$;

3. for any rule $(q, out; pa, in)$ in R_1, the only other rules in R_1 containing p are of the form $(pa, out; r, in)$;

4. $F \subseteq V \setminus T$.

Finite ω-P automata yield a characterization of regular ω-languages:

Theorem 33 *Let $L \subseteq \Sigma^\omega$ be an ω-language. Then L is ω-regular if and only if L can be accepted in the 2-acceptance mode by a finite ω-P automaton.*

Proof. Consider the same constructions as in the proof of Theorem 15; then the interpretation of the finite automata and the finite P automata considered there as finite ω-automata and finite ω-P automata, respectively, already yield the desired results. □

As for finite P automata (see Example 14), more relaxed conditions on the forms of the rules in finite ω-P automata would yield ω-languages which are not ω-regular:

Example 34 *Consider the ω-P automaton*

$$\Pi_2 = (\{a, b, p, q, r\}, \{a, b\}, [_1]_1, p, R_2, \{p\})$$

with R_2 containing the rules $(p, out; pa, in)$, $(p, out; qa, in)$, $(qa, out; r, in)$, $(r, out; sb, in)$, $(sb, out; q, in)$, and $(sb, out; p, in)$. The ω-P automaton Π_2 differs from the P automaton Π_1 from Example 14 only by the additional rule $(sb, out; p, in)$, which is necessary to restart a computation with "state" p and in that way allows for infinite computations, and by taking p as the final state instead of q. Yet the ω-language 2-accepted by Π_2 now is of a more complicated structure than the string language accepted by the P automaton Π_1: The condition that the state p has to be reached infinitely often only guarantees that infinitely often the number of symbols a and the number of symbols b analysed so far is just the same, but it does not guarantee that this condition has to be fulfilled whenever we change again back to "state" p from "states" q and s, respectively. Hence, we do not obtain $L(\Pi_1)^\omega$, where $L(\Pi_1) = \{a^n b^n \mid n \geq 1\}$, but instead the ω-language 2-accepted by Π_2 is L_2^ω, where

$$L_2 = \left\{ a^{n_1} b^{m_2} \ldots a^{n_k} b^{m_k} \mid k \geq 1, \quad \sum_{i=1}^{k} n_i = \sum_{i=1}^{k} m_i, \right.$$
$$\left. \text{and} \quad \sum_{i=1}^{l} n_i \geq \sum_{i=1}^{l} m_i \quad \text{for all } l < k \right\};$$

obviously, L_2^ω is not an ω-regular language.

8.4 Conclusion

We have investigated one specific model of P automata (based on communication rules) allowing for the simulation of the actions of Turing machines on infinite words. The way we effectively constructed the corresponding ω-P automaton from a given ω-Turing machine for each of the well-known modes of acceptance would also work for deterministic variants of ω-Turing machines/ω-P automata as well as for ω-Turing machines describing functions on ω-languages. With respect to the number of membranes, the results elaborated in this Chapter are already optimal.

As there exist a lot of other models of P systems, it may be an interesting task to use these models as basis for other variants of ω-P automata. The main challenge seems to lie in the fact that many proof ideas in the literature rely on the use of a trap symbol to make computations non-halting, whereas in the case of ω-P automata failing computations should stop instead.

Chapter 9

On "Weak" Determinism in P Automata

Quite a few variants of P systems have already been implemented (e.g., see [3], [11]). But so far, only generating P systems have been taken into account, dealing with their inherent non-determinism, which is hard to implement on deterministic machines, i.e., conventional computers. Moreover, usual algorithms (like, e.g., backtracking) to find possible solutions which lead to halting computations are not very efficient. On the other hand, when thinking of implementing accepting P systems, we can define a kind of "weak" determinism leading to more efficient computations, that is, under certain conditions, we can limit the number of possible intermediate configurations to examine until the simulation of a computation in a P automaton eventually terminates successfully. In this Chapter we will show how this can be done for accepting P systems and P automata as considered in previous Chapters without taking into account the infinite case, i.e., $\omega-$P automata. As a computation in a P automaton can also be seen as a hierarchy of choices to be made, it can be represented in a more pictorial way by a tree. We start this Chapter by giving the definition of a computation tree that will be used in the following and continue by illustrating this with an example. We will then define the new term of $k-$determinism, which is followed by a syntactic study of the systems investigated in previous Chapters under the point of view of their $k-$deterministic behaviour. Some final remarks and open problems conclude this Chapter.

9.1 Preliminary Definition and Example

To represent a computation in a P automaton we will now, following the definition given in [51], use the notion of a computation tree:

The *computation tree* of a P automaton is a rooted labelled maximal tree, where the root node of the tree corresponds to the initial configuration of the system. The children of a node are configurations that follow in a one-step transition. Nodes are labelled by configurations and edges are labelled by multisets of applicable rules. We say that a computation halts if it represents a finite branch in the computation tree.

Example 35 *Consider the P automaton from Example 14*

$$\Pi_1 = (\{a, b, p, q, r\}, \{a, b\}, [_1]_1, p, R_1, \{q\})$$

with R_1 *containing the rules*
$1 : (p, out; pa, in)$, $2 : (p, out; qa, in)$, $3 : (qa, out; r, in)$, $4 : (r, out; sb, in)$, $5 : (sb, out; q, in)$.
(For sake of simplicity we have labelled the rules by natural numbers and will use the number of a rule to mark the corresponding edge.)

Then we can build the following computation tree:

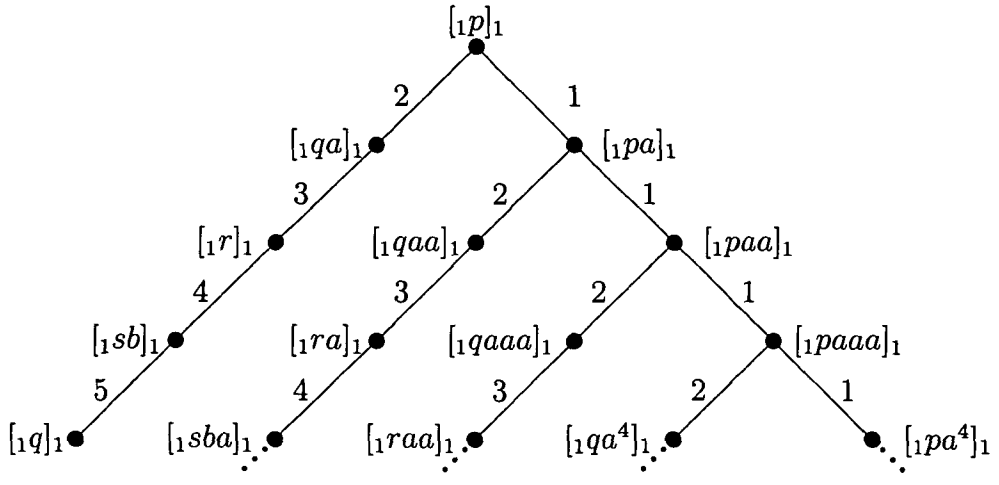


Figure 9.1: Computation tree of the P automaton Π_1.

We start with the initial configuration containing one object p, thus labelling the root node with $[_1p]_1$. *Then, in the first step, the system can already choose between the following two rules to be applied:*

1. *Applying rule* 2, *i.e.,* $(p, out; qa, in)$, *leads to the configuration* $[_1 qa]_1$, *from where rules* 3, 4, 5 *have to be applied consecutively until reaching the final state* $[_1 q]_1$. *This can also be seen in the computation tree in figure 9.1 by following the leftmost branch from the root to the leave node* $[_1 q]_1$.

2. *The application of* $(p, out; pa, in)$ *on the other hand yields a configuration* $[_1 pa]_1$, *where again the system can choose to apply*

   (a) *rule* 2, *hence starting to send out the symbols a and importing the same number of symbols b (this time using the sequence of rules* 3, 4, 5 *twice until reaching the final state* $[_1 q]_1$*),*

   (b) *rule* 1 *again, in order to import an additional symbol a, hence following the rightmost branch of the computation tree shown in figure 9.1 and so on.*

Hence, the computation halts successfully in the final state q after having accepted the string $a^n b^n$, $n \geq 1$. *Speaking in terms more related to the tree structure, the system has accepted the string* $a^n b^n$ *whenever a leave node is reached.*

9.2 $k-$Determinism

To be more efficient in possible implementations of any type of accepting P systems or P automata, we do not want to expand the complete computation tree during the simulation, but rather "look ahead" in the computation tree as little as possible to be able to exclude the paths which would lead to an infinite loop and choose the path which leads to a continuation of possible acceptance.

Thus, we say that an accepting P system or P automaton has a *level of look-ahead* k, or shorter, is *k-deterministic* if the following condition holds:

> For every run on an initial configuration, if at any moment going at most k steps further for any arbitrary choice of productions to be applied, it can be decided (i.e., syntactically checked) which might be the only reasonable continuation that possibly may lead to successful acceptance.

Remark 36 *In all proofs for analysing / accepting P systems / P automata with initial input considered in this thesis (without taking into account the*

infinite case here), this condition holds: We only used simulations of deterministic register machines, hence, in the case of acceptance there is only one path through the computation tree which is finite. But of course, at a certain point, we have to deal with non-determinism, i.e., in the conditional subtract instructions. Whenever such an instruction has to be simulated, the system has to guess non-deterministically, if there is a corresponding symbol present or not and thus can act accordingly. Yet it is always guaranteed that if the wrong choice has been made, a trap symbol is introduced after a certain number of steps, hence the condition is fulfilled.

Remark 37 *Note that for the system from Example 35, the condition stated above does not hold, because in each step, a non-deterministic choice concerning the input sequence has to be made. Hence, there is no way to syntactically check which might be the only reasonable continuation that possibly may lead to successful acceptance.*

Broadening the above definition to terms specific for implementations, we define a *$k-$deterministic procedure* as a recursive procedure that has to consider at most k further steps down the computation tree yielding the maximal set of productions which is uniquely determined to be applied in the next step for the computation to continue successfully. This production set can also be empty, which is the case when the system halts.

9.3 Results

What follows is a syntactic investigation of the systems considered in this work with respect to their k-deterministic behaviour.

Remark 38 *As already pointed out in Remark 37, we have to deal with non-determinism when considering systems that accept an input sequence of terminal symbols, as there is always a choice to be made. The only chance to get rid of this inherent non-determinism is to assume that for each run on the respective system, the input stream sequence, i.e., the sequence of terminal symbols, is deterministically given in the environment so that the choice which rule has to be applied depends on the actual terminal symbol. Only with this assumption, the results in this section would also hold true for the respective non-initial systems.*

We start with initial analysing P systems as defined in Chapter 4:

Theorem 39 *For every recursively enumerable set of vectors of natural numbers there exists a* $2-$*deterministic initial analysing P system with antiport rules with radius* $(2,1)$ *or* $(1,2)$.

Proof. Going into the details of the proof of Corollary 6, it can be determined, following Remark 36, that the only choices that have to be made are in the simulation of a conditional subtract instruction:

$1 : (ja_i, out; k, in)$
$2 : (j, out; j'j'', in)$
$3 : (j'a_i, out; f, in)$ $\quad (f, out; f'f'', in)$ and $(f'f'', out; f, in)$
$4 : (j'', out; j'''j'''', in)$
$5 : (j'''j'''', out; j'''', in)$
$6 : (j'j'''', out; l, in)$

The only situation where a trap symbol f could be introduced into the system is, when there is at least one symbol a_i present, but instead of $(ja_i, out; k, in)$, the rule $(j, out; j'j'', in)$ is chosen. Then, in the next step, the rule $(j'a_i, out; f, in)$ (together with $(j'', out; j'''j'''', in)$) has to be applied, bringing in the symbol f which prevents the system from halting by using $(f, out; f'f'', in)$ and $(f'f'', out; f, in)$ without ever coming to an end.

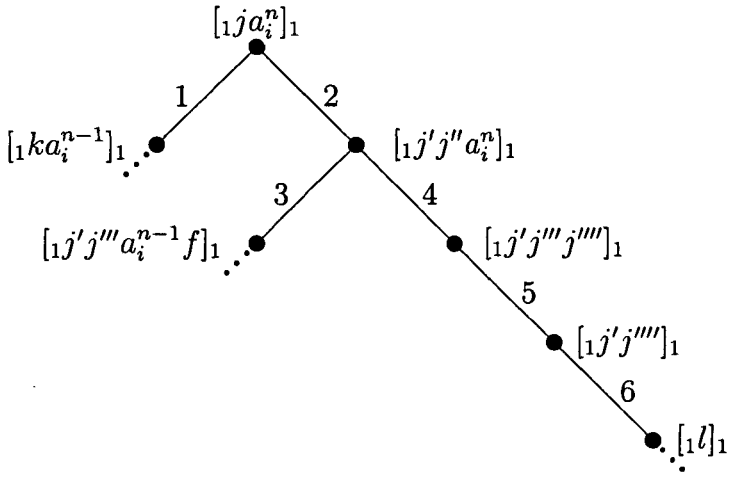


Figure 9.2: Computation subtree of the initial analysing P system.

This situation is also shown as a "computation subtree" in Figure 9.2, where no other symbols are taken into account except for the ones described here, hence the root node only represents the configuration containing the objects j and a_i^n. Whenever the $2-$deterministic procedure reaches the root node of the subtree from Figure 9.2, there are two possibilities:

1. $n = 0$, i.e., no symbol a_i is present. Then there is no choice, only the edge labelled by 2 can be followed, which means applying the rule $(j, out; j'j'', in)$ and yielding the configuration $[_1 j'j'']_1$. From that point, again no choice is to be made, the only possibility to proceed is going down the edge labelled by 4 (applying $(j'', out; j'''j'''', in)$) and so on.

2. $n > 0$: In this case, already in the root node there are two possibilities:

   (a) Applying $(ja_i, out; k, in)$ (going down edge 1) and continue the computation successfully, or

   (b) applying $(j, out; j'j'', in)$ (edge 2). In the sequel, edge 3 has to be followed, meaning the application of the rules $(j'a_i, out; f, in)$ and $(j'', out; j'''j'''', in)$ (as the system works in a maximally parallel way), which introduces the trap symbol f, hence leading to a non-halting computation.

From the explanations given above, as well as from the tree in Figure 9.2, it can easily be determined that initial analysing P systems with antiport rules with radius $(2, 1)$ or $(1, 2)$ have a level of look-ahead 2. □

If we now consider initial P automata with membrane channels, again we deal with systems that have a level of look-ahead 2 :

Theorem 40 *For every recursively enumerable set of vectors of natural numbers there exists a 2−deterministic initial P automaton with membrane channels.*

Proof. Looking at the proof of Corollary 8, it can easily be seen that there is only one crucial point in the register machine simulation, i.e., when simulating the conditional subtract instruction by the following rules:

$\langle j; ja_i, out; kf_j, in\rangle$ $\langle f_j, in; a_i\rangle$
$\langle j; j, out; j'j'', in\rangle$ $\langle j'', in; a_i\rangle$
$\langle j'; j'j'', out; lf_j, in\rangle$ $\langle f_j, in; j''\rangle$
$\langle f_j; f_j, out; f_j, in\rangle$

Being in any configuration that contains the objects j as well as a_i^n, then, depending on n, we have to distinguish the following cases:

1. $n = 0$ allows for two possible next configurations:

   (a) Applying $\langle j; j, out; j'j'', in\rangle$ sends out the object j and brings into the system j' and j''. Then the application of $\langle j'; j'j'', out; lf_j, in\rangle$ sends out j' and j'' while importing l. (f_j is not introduced to the system because of the presence of j'');

(b) If, on the other hand, $\langle j; ja_i, out; kf_j, in\rangle$ is applied, then a trap symbol is introduced after one step only, as the inhibitor a_i is not present to prevent f_j from coming in by $\langle f_j, in; a_i\rangle$. This branch does not have to be followed further, because after one step only it is already clear that under the given circumstances the wrong choice was made.

The computation subtree of this situation is shown in Figure 9.3, where for sake of simplicity, the configurations are limited in the sense that they only contain the objects mentioned above.

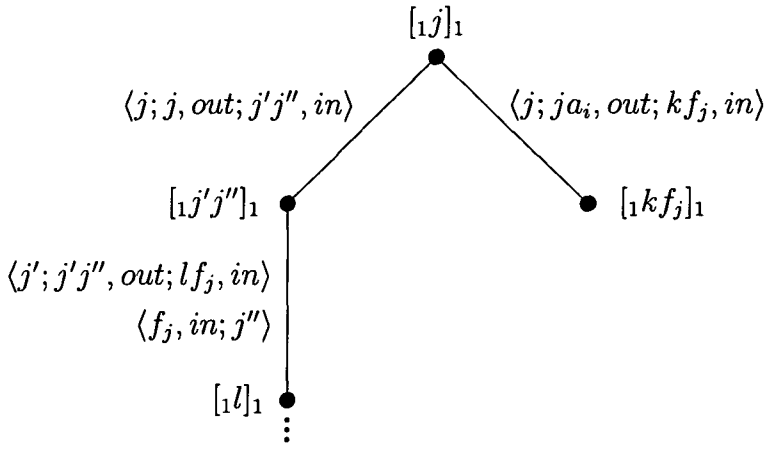


Figure 9.3: Computation subtree for the case $n = 0$.

2. $n > 0$ allows for the following next configurations:

   (a) the application of $\langle j; ja_i, out; kf_j, in\rangle$ lets the system continue successfully, because there is at least one object a_i which can act as prohibitor for f_j to come in;

   (b) but finally, if $\langle j; j, out; j'j'', in\rangle$ is applied while there is at least one a_i in the system, then j'' cannot enter, and so in the next step, when applying $\langle j'; j'j'', out; lf_j, in\rangle$, f_j has to be imported.

   This situation is now shown as a computation subtree in Figure 9.4, where again the configurations only contain the relevant objects.

Hence, as at least in one case there are two transitions to be made until the trap symbol occurs, we can conclude that initial P automata with activated/prohibited membrane channels have a level of look-ahead 2. □

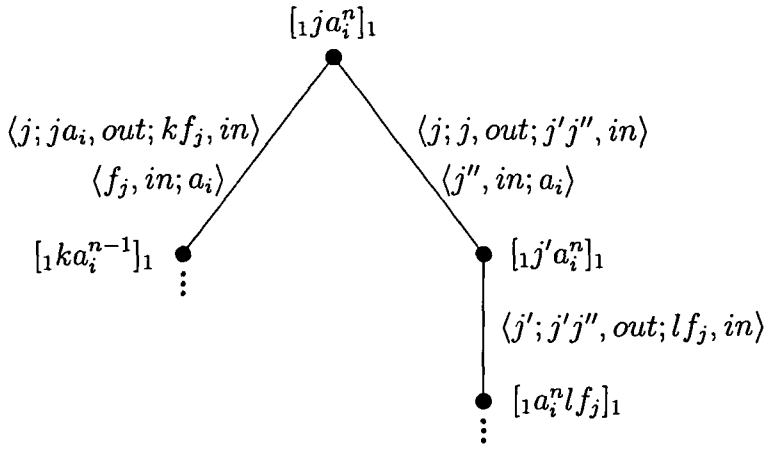


Figure 9.4: Computation subtree for the case $n > 0$

Looking at the working mode of P automata with membrane channels, where not for all the channels opened an object for passing through in either direction has to be present, we could also think of another variant. If during the computation of a P automaton with membrane channels a rule can be applied only if *all* opened channels are used in the succeeding step, we will call such a system a *restricted P automaton with membrane channels.*

Theorem 41 *Let $L \subseteq T^*$ be a recursively enumerable set. Then L can be recognized by a restricted P automaton with membrane channels in only one membrane using only singleton activators and prohibitors.*

Proof. We use exactly the same construction as in the proof of Theorem 7 except for the simulation of the conditional subtract instruction. Because of the restricted working mode, an instruction $j : (S(i), k, l)$ is now simulated by :

$\langle j; j a_i, out; k, in\rangle$

$\langle j; j, out; l, in\rangle \qquad \langle l, in; a_i\rangle$

This simulation is straightforward: The rule $\langle j; j a_i, out; k, in\rangle$ can only be used if an object a_i is present, hence no prohibitor is needed here. On the other hand, $\langle j; j, out; l, in\rangle$ can only be applied if no object a_i is present. Observe that this rule cannot be applied, if there is an object a_i in the system, because in this case the object l would be prohibited to enter, which in consequence makes the use of this rule impossible. □

The same result holds true for vectors of non-negative integers when considering initial restricted P automata with membrane channels:

Corollary 42 *Let $L \subseteq \mathbf{N}^k$, $k \geq 1$, be a recursively enumerable set of (vectors of) non-negative integers. Then L can be accepted by an initial restricted P automaton with membrane channels in only one membrane using only singleton promoting and inhibiting multisets as well as singleton objects transported through the skin membrane.*

Hence, using this restriction on the working mode of an initial P automaton with membrane channels, we get a deterministic system:

Theorem 43 *For every recursively enumerable set of vectors of natural numbers there exists a deterministic initial restricted P automaton with membrane channels.*

Proof. According to the definition of an initial restricted P automaton with membrane channels given above, at no point there is any choice to be made. That is, whenever an object acts as a prohibitor, the corresponding activating rule cannot be applied, as there is at least one object prevented from passing through an opened channel. □

Looking now at initial P automata with conditional communication rules assigned to membranes, the following result can be obtained:

Theorem 44 *For every recursively enumerable set of vectors of natural numbers there exits a deterministic initial P automaton with conditional communication rules assigned to membranes consisting of only one membrane using only singleton promoting and inhibiting multisets as well as singleton objects transported through the skin membrane.*

Proof. Going into the details of the proof of Corollary 18, it becomes clear that (because of the inhibiting multisets) there is no point where a non-deterministic choice has to be made, hence, the system is deterministic. □

Looking at accepting P systems with catalysts, things get a bit more complex, that is, they have a higher level of look-ahead as the ones investigated so far (observe that these systems are already initial by construction):

Theorem 45 *For every recursively enumerable set of vectors of natural numbers there exits a $4-$deterministic accepting P system with catalysts.*

Proof. We start by reproducing the relevant part of the proof of Theorem 24, i.e., the simulation of the instruction $j : (S(i), k, l)$:

simulation of the subtract-instruction $j : (S(a), k, l)$ if	
a. register a is not empty	**b.** register a is empty
$c_a p_j^{\langle m,1\rangle} \to c_a \hat{p}_j \hat{p}'_j$ $c_{a\ominus_m 1} p_j^{\langle 1,1\rangle} \to c_{a\ominus_m 1}$	$c_a p_j^{\langle m,1\rangle} \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$ $c_{a\ominus_m 1} p_j^{\langle 1,1\rangle} \to c_{a\ominus_m 1}$
$c_a o_a \to c_a c'_a$ $\langle c_a \hat{p}'_j \to c_a \# \rangle$ $c_{a\ominus_m 1} \hat{p}_j \to c_{a\ominus_m 1}$	$c_a \bar{p}_j \to c_a$ $c_{a\ominus_m 1} \bar{p}''_j \to c_{a\ominus_m 1} p''_j$
$c_a c'_a \to c_a c''_a$ $c_{a\ominus_m 1} \hat{p}'_j \to c_{a\ominus_m 1} \hat{p}''_j$	$\langle c_a o_a \to c_a c'_a \rangle$ $c_{a\ominus_m 1} p''_j \to c_{a\ominus_m 1} p'_j$
$c_a \hat{p}''_j \to c_a p_k^{\langle 1,1\rangle} \ldots p_k^{\langle m,1\rangle}$ $c_{a\ominus_m 1} c''_a \to c_{a\ominus_m 1}$	$c_a p'_j \to c_a p_l^{\langle 1,1\rangle} \ldots p_l^{\langle m,1\rangle}$ $c_{a\ominus_m 1} \bar{p}'_j \to c_{a\ominus_m 1}$

This time, we only give a schematic representation of the computation subtree of the above situation in Figure 9.5. That is, we omit any objects that might also be part of the respective configuration, but are not mentioned in the explanation. We also omit to include the catalysts c_a and $c_{a\ominus_m 1}$, respectively, as they are contained in all configurations. Moreover, we represent all misleading configurations by # only. (Going into the details of the proof of Theorem 24, we can find out that using the wrong rules at any moment leads to the introduction of a trap symbol after at most 2 steps, which is not shown here, either. Rather, such a case is represented as one edge only ending in a node labelled by #. As we claim the considered P automata to be 4−deterministic, this is of no big relevance for this proof, but helps to keep things more clear.)

We start with the configuration containing - among other objects - the two catalysts c_a and $c_{a\ominus_m 1}$, as well as the objects $p_j^{\langle m,1\rangle}$ and $p_j^{\langle 1,1\rangle}$ (symbolised as the root node in Figure 9.5). Already here, we have two choices to proceed. Of course, there are more possibilities to proceed, but all other ones would lead to the introduction of the trap symbol and therefore to an infinite loop, which is shown in Figure 9.5 by the black vertical edge that leads to a node labelled by #. So in the following, we will only consider the "constructive" cases:

1. Let us first assume that there is at least one object o_a present (following the dotted branch), and therefore choosing the rules $c_a p_j^{\langle m,1\rangle} \to c_a \hat{p}_j \hat{p}'_j$ as well as $c_{a\ominus_m 1} p_j^{\langle 1,1\rangle} \to c_{a\ominus_m 1}$. Then the next configuration contains the objects $\hat{p}_j$ and $\hat{p}'_j$. Consuming $\hat{p}_j$ by $c_{a\ominus_m 1} \hat{p}_j \to c_{a\ominus_m 1}$ is a good choice here, as otherwise the trap symbol can immediately enter. $\hat{p}'_j$ on the other hand, is the only object that could wait until being consumed without causing troubles. So if an object o_a is currently present, then it

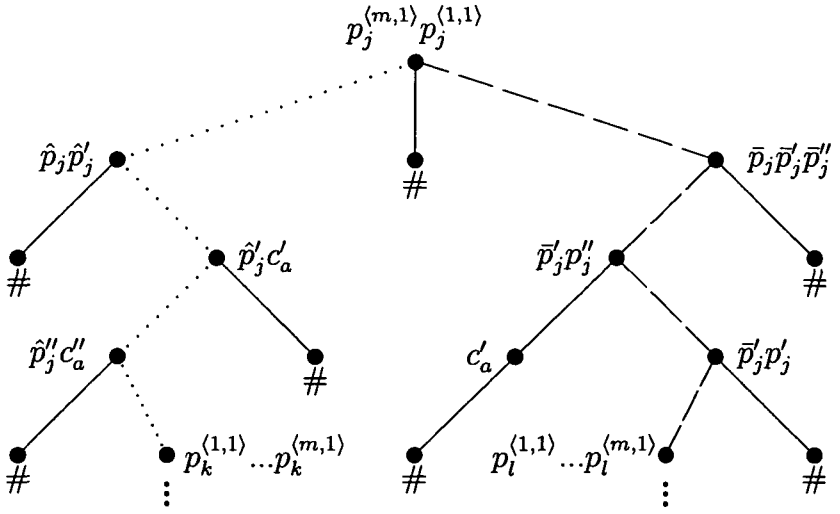


Figure 9.5: Schematic representaion of the computation subtree.

should be consumed by $c_a o_a \to c_a c'_a$. Only if our assumption was wrong (there is no object o_a), the catalyst c_a - due to maximal parallelism - has to be used together with $\hat{p}'_j$ to introduce the object #, which in Figure 9.5 means following the plain edge. So in this case, we already know after two steps if the non-deterministic guess concerning the presence of objects o_a was correct. Only if this is the case, we follow the dotted edges to a configuration containing $p_k^{\langle 1,1\rangle}...p_k^{\langle m,1\rangle}$ enabling the system to proceed with the simulation of the next instruction. (Note that from the node labelled by $\hat{p}'_j c'_a$, if taking the "wrong" rules, it would in fact need two further steps until the trap symbol appears.)

2. On the other hand, we can non-deterministically assume that there are no objects o_a present (i.e., register a is assumed to be empty). In this case, we follow the dashed edge from Figure 9.5 by first applying in parallel $c_a p_j^{\langle m,1\rangle} \to c_a \bar{p}_j \bar{p}'_j \bar{p}''_j$ and $c_{a\ominus_m 1} p_j^{\langle 1,1\rangle} \to c_{a\ominus_m 1}$. The only suitable continuation at this stage is using the rules $c_a \bar{p}_j \to c_a$ and $c_{a\ominus_m 1}\bar{p}''_j \to c_{a\ominus_m 1} p''_j$ leading to a configuration that contains $\bar{p}''_j$ and p''_j. Here we have reached a crucial point: Only if there really was no object o_a present we can follow the dashed edges further down, hence proceeding successfully. But if there was an object o_a contained in the system at that stage, then it has to be consumed (again due to maximal parallelism) by the rule $c_a o_a \to c_a c'_a$ and subsequently introducing the symbol #. So having made the wrong guess in this case is only obvious

after four steps. (Note that again from the configuration containing $\bar{p}'_j$ and p'_j it would take two further steps if making the wrong choice.)

It follows from the explanations given above that at most four steps are needed until the trap symbol # finally appears after having made a wrong non-deterministic decision. Consequently, P automata with catalysts have a level of look-ahead 4. □

9.4 Conclusion

In this Chapter we have introduced the new concept of $k-$determinism concerning P automata. We also have investigated some previously considered variants with respect to their k-deterministic behaviour.

We should like to point out that we only made a pure syntactic analysis. But it seems to be worth noting that even in the syntactic studies of the $k-$deterministic behaviour of various systems, k could sometimes be reduced by introducing other features, for instance priority relations among the rules, which on the other hand are not necessary to obtain computationally complete systems [60].

It seems to be clear whatsoever that when including semantic information, we could get better results, too. That is, when thinking of a possible implementation (or rather simulation) of any of the systems investigated, the current configuration of the system is known in every single step. Consequently, whenever the simulation reaches a point where a non-deterministic choice had to be made in the syntactic studies above, the situation now becomes different: As it can be determined whether the object to be possibly subtracted form the system is present or not, a non-deterministic guess in this situation becomes obsolete, thereby most of the time decreasing the number of possible edges to "look ahead" in the computation tree.

We should like to conclude this Chapter by pointing out that there are still some open questions. Having in mind the efficiency of possible implementations of P automata, we here have only considered how to limit the depth of the computation tree. But what about the breadth? Can we a priori determine the number of possible configurations that a k-deterministic procedure has to take into account? When for example thinking of purely catalytic P automata, the answer seems to be definitely positive. But this remains an interesting topic for future research.

Chapter 10

Final Remarks

We have made a survey of P automata, and after a brief literature review, we have investigated several purely communicating variants of P automata:

In Chapter 4, we focused on (initial) analysing P systems, showing that already one membrane suffices for these systems to obtain their maximal recognizing power with rules of radius $(1,2)$ or $(2,1)$.

For the related model of (initial) P automata with membrane channels some similar results were shown in Chapter 5. When using activating as well as prohibiting rules with radius $(1,2)$ or $(2,1)$, we only need singleton activators and prohibitors, whereas when omitting the prohibiting rules, the size of the activator multiset has to be increased to be able to recognize any recursively enumerable language of strings and multisets, respectively. We also showed that a very restricted variant of P automata with membrane channels using only special activating rules (in this case antiport rules) allows for the characterization of regular languages.

Another purely communicating variant was investigated in Chapter 6. There we could show that (initial) P automata with conditional communication rules assigned to membranes (PACCRAMs) only need one membrane and singleton promoters and inhibitors as well as singleton objects transported through the skin membrane to accept recursively enumerable string languages (and recursively enumerable sets of (vectors of) non-negative integers, respectively). With respect to the number of membranes as well as with respect to the size of the multisets used as promoters, inhibitors, and strings transported across a membrane these results are already optimal.

We then focused on the number of catalysts necessary to obtain systems with maximal recognizing power: In Chapter 7 we showed that for the respective initial variants, the number of catalysts depends on the number α of components of the vector of non-negative integers to be analysed, hence,

$\alpha + 2$ catalysts are necessary.

We then proved that P automata (with catalysts) can accept every recursively enumerable string language (by halting or by final state) with two catalysts in only one membrane. For the purely catalytic variants of all these systems, one catalyst more is necessary.

In Chapter 8, we considered one specific model of P automata (based on communication rules) allowing for the simulation of the actions of Turing machines on infinite words. With respect to the number of membranes, the results elaborated there are already optimal.

Finally we introduced the new notion of $k-$determinism. In Chapter 9 we investigated most of the initial variants mentioned before with respect to their $k-$deterministic behaviour.

We have already pointed out some open problems or topics that deserve further attention in the respective Chapters. But, of course, many more things remain to be done. The investigations in the last Chapter have been of a more theoretic type, but will hopefully also be useful in practice. In fact, a simulation under this point of view is already in progress.

Another point concerning this topic is the question whether the $k-$deterministic behaviour of some systems could be improved (obviously for $k > 0$). We have seen that in some proofs in previous Chapters, we already obtained a maximal recognizing power with minimal ingredients. For example, in the case of initial analysing P systems and initial P automata with membrane channels, only one membrane is needed with the rules having a minimal radius. And yet it turned out that they exhibit a level of look-ahead 2. So it would be interesting to see if a reduction of k is possible in this case. But, as for now, this remains for future research.

Acknowledgements

I gratefully acknowledge the enormous patience, competence and support of my supervisor Rudolf Freund and I want to express my deep respect and gratitude to Gheorghe Păun. I am also indebted to Erzsébet Csuhaj-Varjú, Ludwig Staiger and Györgi Vaszil for fruitful discussions on many topics presented here. Moreover I would like to thank all my co-authors not mentioned before.

The presentations of [23], [26], [27], [29], [30], [33] at international workshops and meetings were supported by MolCoNet project IST-2001-32008.

Bibliography

[1] A. Alhazov: Minimizing Evolution-Communication P Systems and EC P Automata. In [9], 23–31.

[2] A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Preproceedings of Workshop on Membrane Computing, WMC-2003*, Tarragona, July 17-22, 2003, Rovira i Virgili Univ., Tech. Rep. No. 28, Tarragona, 2003.

[3] F. Arroyo, C. Luengo, A.V. Baranda, L.F. de Mingo: A Software Simulation of Transition P Systems in Haskell. In [54], 19–32.

[4] G. Bel-Enguix, R. Gramatovici: Active P Automata and Natural Language Processing. In [2], 61–71.

[5] F. Bernardini, V. Manca: P Systems with Boundary Rules. In [55], 97–102, and [54], 107–118.

[6] P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg: Membrane Systems with Promoters/Inhibitors. *Acta Informatica* 38, 10 (2002), 695–720.

[7] D. Bray: Protein Molecules as Computational Elements in Living Cells. *Nature* 376 (1995), 307–312.

[8] C.S. Calude, Gh. Păun: *Computing with Cells and Atoms.* Taylor & Francis, London (2001).

[9] M. Cavaliere, C. Martín-Vide, Gh. Păun (Eds.): *Brainstorming Week on Membrane Computing*, Rovira i Virgili Univ., Tech. Rep. No. 26, Tarragona, 2003.

[10] L. Cienciala, L. Ciencialova: P Automata with Priorities. In [2], 161–168.

[11] G. Ciobanu, D. Paraschiv: Membrane Software. A P System Simulator. *Fundamenta Informaticae* 49, 1-3 (2002), 61–66.

[12] R.S. Cohen, A.Y. Gold: ω-Computations on Turing Machines. *Theoret. Comput. Sci.* 6 (1978), 1–23.

[13] R.S. Cohen, A.Y. Gold: On the Complexity of ω-Type Turing Acceptors. *Theoret. Comput. Sci.* 10 (1980), 249–272.

[14] E. Csuhaj-Varjú, G. Vaszil: P Automata. In [55], 177–192.

[15] E. Csuhaj-Varjú, G. Vaszil: P Automata or Purely Communicating Accepting P Systems. In [54], 219–233.

[16] E. Csuhaj-Varjú, Gy. Vaszil: New Results and Research Directions Concerning P Automata, Accepting P Systems with Communication Only. In [9], 171–179.

[17] Z. Dang, O. Egecioglu, O.H. Ibarra, G. Saxena: Characterizations of Catalytic Membrane Computing Systems. *To appear.*

[18] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin (1989).

[19] J. Dassow, Gh. Păun: On the Power of Membrane Computing, *Journal of Universal Computer Science* **5**, 2 (1999), 33–49 (`http://www.iicm.edu/jucs`).

[20] J. Engelfriet, H.J. Hoogeboom: X-Automata on ω-Words. *Theoret. Comput. Sci.* 110,1 (1993) 1–51.

[21] R. Freund, Generalized P Systems: *Fundamentals of Computation Theory, FCT'99,* Iaşi, 1999, (Ciobanu, G., Păun, Gh. Eds.), LNCS 1684, Springer-Verlag, Berlin, 1999, 281–292.

[22] R. Freund: Sequential P Systems. *Workshop on Multiset Processing,* Curtea de Argeş, Romania, 2000, and Theorietag 2000 (Freund, R., Ed.), TU Wien, 2000, 177–183.

[23] R. Freund, F. Freund, M. Margenstern, M. Oswald, Yu. Rogozhin, S. Verlan: P Systems with Cutting/Recombination Rules Assigned to Membranes. In [2], 241–251.

[24] R. Freund, C. Martin-Vide, A. Obtulowicz, Gh. Păun: On Three Classes of Automata-like P Systems. *Proc. 7th Int. Conf. DLT2003* (Z. Ésik, Z. Fülöp, Eds.), Szeged, Hungary, 2003, LNCS 2710, Springer-Verlag, 2003, 292–303.

[25] R. Freund, M. Oswald: Variants of GP Systems. *Preproceedings of Workshop on Membrane Computing* (C. Martín-Vide, Gh. Păun, Eds.), Curtea de Argeş, Romania, 2001, Rovira i Virgili Univ., Tech. Rep. No. 17, Tarragona, 2001, 77–88.

[26] R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**, 1-3 (2002), 81–102.

[27] R. Freund, M. Oswald, P Systems with Activated/Prohibited Membrane Channels. In [54], 261–268.

[28] R. Freund, M. Oswald: A Short Note on Analysing P Systems with Antiport Rules. *Bulletin EATCS*, **78** (2002), 231–236.

[29] R. Freund, M. Oswald: P Systems with Conditional Communication Rules Assigned to Membranes. In [2], 231–240.

[30] R. Freund, M. Oswald, P. Sosík: Reducing the Number of Catalysts Needed in Computationally Universal Systems without Priorities. In. E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszil, Eds.: Fifth International Workshop Descriptional Complexity of Formal Systems. Budapest, Hungary, July 2003. MTA SZTAKI, Budapest (2003), 102–113.

[31] R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *To appear.*

[32] R. Freund, M. Oswald, L. Staiger: P Automata on Finite and Infinite Words. *Proc. second year MolCoNet meeting,* Budapest, 2002.

[33] R. Freund, M. Oswald, L. Staiger: ω-P Automata with Communication Rules. In [2], 252–265.

[34] R. Freund, M. Oswald, L. Staiger: P Automaten und $\omega-$P Automaten. *13. Theorietag "Automaten und Formale Sprachen"* (M. Holzer, Ed.), TU München, 2003, 35–47.

[35] R. Freund, A. Păun: Membrane Systems with Symport/Antiport: Universality Results. In [54], 270–287.

[36] R. Freund, Gh. Păun: On the Number of Non-Terminal Symbols in Graph-Controlled, Programmed and Matrix Grammars. *Proc. MCU 2001* (M. Margenstern, Yu. Rogozhin, Eds.), Chişinău, 2001, LNCS 2055, Springer-Verlag, Berlin (2001), 214–225.

[37] R. Freund, C. Martín-Vide, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *To appear in TCS.*

[38] P. Frisco, H. J. Hoogeboom: Simulating Counter Automata by P Systems with Symport/Antiport. In [55], 237–248, and [54], 288–301.

[39] O.H. Ibarra: The Number of Membranes Matters. In [2], 273–285.

[40] K. Krithivasan: P Automata with Tapes. In [9], 216–225.

[41] K. Krithivasan, S.V. Varma: On Minimising Finite State P Automata. Submitted, 2003.

[42] M. Madhu, K. Krithivasan: On a Class of P Automata. submitted, 2002.

[43] M. L. Minsky: *Berechnung: Endliche und Unendliche Maschinen.* Verlag Berliner Union GmbH. Stuttgart, Verlag W. Kohlhammer GmbH. Stuttgart (1971). Original English volume: M. L. Minsky: *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, New Jersey, USA (1967).

[44] M. Oswald: *Molecular Computations with P Systems*, diploma thesis, Techn. Univ. Wien, 2001.

[45] M. Oswald: Verallgemeinerte P-Systeme mit Verbotenem Kontext. *12. Theorietag "Automaten und Formale Sprachen"* (R. Mazala, L. Staiger, R. Winter, Eds.), Wittenberg, 2002, 40–41.

[46] M. Oswald, R. Freund: P Automata with Membrane Channels. *Proceedings of the eighth Int. Symp. on Artificial Life and Robotics* (M. Sugisaka, H. Tanaka, Eds.), Beppu, Japan, 2003, 275–278.

[47] A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. *New Generation Computing*, **20**, 3 (2002), 295–306.

[48] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61**, 1 (2000), 108–143.

[49] Gh. Păun: Computing with Membranes: An Introduction. *Bulletin EATCS* **67** (1999), 139–152.

[50] Gh. Păun, Computing with Membranes (P Systems): Twenty Six Research Topics. CDMTCS TR 119, Univ. of Auckland, 2000 (`www.cs.auckland.ac.nz/CDMTCS`).

[51] Gh. Păun: *Membrane Computing - An Introduction*. Springer-Verlag, Berlin (2002).

[52] Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer-Verlag, Berlin (1998).

[53] Gh. Păun, G. Rozenberg, A. Salomaa: Membrane Computing with External Output. *Fundamenta Informaticae* **41** (3), 2000, 259–266, and TUCS Research Report No. 218, 1998 (http://www.tucs.fi).

[54] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Membrane Computing*. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002. LNCS 2597, Springer, Berlin, 2003.

[55] Gh. Păun, C. Zandron (Eds.): *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002)*, Curtea de Argeş, Romania, 2002.

[56] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*. 3 Volumes, Springer-Verlag, Berlin (1997).

[57] A. Salomaa: *Formal Languages*. Academic Press, New York (1973).

[58] P. Sosík: P Systems Versus Register Machines: Two Universality Proofs. In [55], 371–382.

[59] P. Sosík: The Power of Catalysts and Priorities in Membrane Systems. *Grammars* **6**, 1 (2003), 13–24.

[60] P. Sosík, R. Freund: P Systems Without Priorities are Computationally Universal. In [54], 400–409.

[61] L. Staiger: ω-Languages. In [56], Vol. 3, 339–387.

[62] K. Wagner, L. Staiger: Recursive ω-Languages. M. Karpiński (Ed.): *Fundamentals of Computation Theory*. LNCS 56, Springer-Verlag, Berlin, 1977, 532–537.

[63] The P Systems Web Page. `http://psystems.disco.unimib.it`

Curriculum Vitae

Marion Oswald received her master degree in computer science from the Vienna University of Technology, Austria, in 2001. During her studies she obtained some practical experience working as assistant of the head of department in a technical highschool in Vienna, and later joining Philips Speech Processing, Vienna. She currently is working as student assistant at the Institute for Computer Languages, Theory and Logic Group, Vienna University of Technology, Austria. Her research interests include but are not limited to artificial life as well as models and systems for biological computing, in which fields she is author or co-author of more than ten scientific papers.