

TDK DOLGOZAT

Kántor Kristóf Szilveszter

Debrecen
2013

Debreceni Egyetem
Informatikai Kar
Számítógéptudományi Tanszék

Generalized P colony automata, a nature motivated
computational model based on multiset processing

Témavezető

Dr. Vaszil György

Egyetemi Docens

Debreceni Egyetem, Informatikai Kar

Számítógéptudományi Tanszék

Szerző

Kántor Kristóf Szilveszter

Programtervező Informatikus

MSc, 2. évfolyam

Debrecen

2013

Contents

1	Overview	1
1.1	Motivations and Goals	1
1.2	Description of the Authors' Contribution	3
2	Introduction	4
2.1	The Need for Self-Organizing, Decentralized Systems	4
2.2	The Chemical Computational Model	5
2.3	Membrane Systems, P Colonies - Preliminary Definitions	6
3	On the Computational Power of Generalized P Colony Automata	11
3.1	Definitions	12
3.2	GenPCol Automata with Restricted Programs	18
3.3	GenPCol Automata Computing in the Non-Restricted Mode	22
4	Conclusion	27
4.1	Comparison with Similar Models	27
4.2	Future Work	29
5	Summary	30
	Acknowledgement	32

Chapter 1

Overview

1.1 Motivations and Goals

Nature-inspired models have recently gained momentum in composing, coordinating and adapting the “Internet of Services”, where a myriad of services and users are interacting autonomously, exhibiting properties that might assist with programming a global computing platform. The virtual communication world has been overtaken by applications and it is no longer operating the way we were using it in the last two decades. Instead of just surfing, we are accessing a service through application. Internet as a computing platform has become an ecosystem comprising a myriad of heterogeneous and autonomous services encapsulating IT components such as storage space, computing power, software or sensors, which is exploited the best, if according to users’ requests, one composes, coordinates and adapts such services dynamically, [9].

A class of computational models which might be adequate to describe the scenario outlined above is based on what might be called a chemical paradigm of computing. This chemical approach is justified by the observation that chemistry concepts, which express dynamics,

decentralization and parallelism, are similar to those appearing in this virtual ecosystem. Based on chemical laws, programs take place as connections between molecules that react with each other in an implicitly parallel, non-deterministic, autonomous and distributed way.

The aim of this research was to come up with a new computational model, that takes this parallel and decentralized chemical metaphor into account. To this aim, we take automata-like variants of membrane systems (P systems) and P colonies as our starting point, [16]. These systems are nature motivated, unconventional computational models inspired by the structure and the functioning of living cells. They provide a formalization of the above mentioned chemical paradigm of computing in general, and their automata-like variants are designed to emphasize and capture the interaction between the system and its environment. This way they provide a description of a system of interacting agents, processes, or any kind of computational devices forming the elements of the interaction model which seems to be appropriate to give a kind of basis for the formal description of the virtual ecosystem of Future Internet sketched above.

In this study, we introduce a generalized variant of P colony automata, provide results on their computational power, and compare them to other automata-like variants of the P systems model. Our results not only give a characterization of the limits of this type of chemical computational approach, but also help to classify these P system models, to understand their properties and the way they influence their computational/descriptive power.

The work is organized as follows: the second chapter gives some motivation and a short introduction to the chemical computing paradigm and membrane systems (P systems), P colonies in general. In the third chapter, we introduce the reader to generalized P colony automata and after the necessary definitions and some examples, we study the computational capabilities of three of their variants. We also compare our findings with

the corresponding results on related models, namely, the related variants of P automata. We will see that (separating the description of the system and its environment) generalized P colony automata have an essentially more simple description than other P system variants (the number of objects inside the system is bounded by a finite constant), but still, their computational power can in many cases be similar.

1.2 Description of the Authors' Contribution

Definitions, propositions and theorems with their corresponding proofs associated with the generalized P colony automata computational model appear first in this work, thus the model has been defined and first studied here. In addition to the present work, a paper based on the results of Chapter 3 has been accepted for publication in the Journal of Automata, Languages and Combinatorics, [12].

The ideas concerning the chemical computational paradigm and the results on P automata which are used in the last part of Chapter 4 are from [11, 9, 1] and from [8, 4], respectively.

Chapter 2

Introduction

The information found in this chapter is heavily based on [11, 9] and [1], that the reader should look into for a more in-depth understanding of the problems proposed here.

2.1 The Need for Self-Organizing, Decentralized Systems

Since the early 1980s the Internet has grown into a worldwide open infrastructure for communication, commerce and information. Due to the rapid development of technology, the Internet has become the central part of our lives. Evidence about the statistics on Internet traffic growth, which show that Internet traffic grows about 50-60% annually, clearly confirms the statement that the Internet is the global network of the 21st century.

Behind these statistics are a whole mix of social and economic trends. Digital convergence has changed the business landscape for consumer electronics, media industries, information technologies and telecommunications forever. Cloud computing and software as a

service are emerging new paradigms that open up a myriad of opportunities for collaboration, innovation and efficiencies in business and organizations.

Although, in addition to just being an evolution, the Future Internet requires relentlessly new approaches, such as new architectures, new ways of managing data, new interfaces and new ways of integrating all the different Internet entities - people, devices, sensors, services and things. Since this will not only assure content and information, but also things and services, Service Oriented Architecture technologies are getting favored in the context of Future Internet where the role of service providers are taken up by diverse organizations and enterprises, while such services are exploited by users to fulfil their needs.

Nature inspired, unconventional computing models have recently gained momentum in expressing service coordination, for being autonomic, dynamic, decentralized and parallel.

2.2 The Chemical Computational Model

A chemical “machine” can be thought of as a symbolic chemical solution where data and computation are represented by molecules floating around and reactions between them. When certain molecules fulfill a reaction condition, they are replaced by the result of the reaction. Formally, this setup is represented by a multiset of symbols/objects representing the molecules, and multiset “rewriting” rules representing the reactions. The analogy can be carried over also to the execution model: the Brownian motion of the chemical molecules correspond to the parallel and nondeterministic computation of the chemical machine (see Table 2.1).

We can formulate some of the important characteristic properties of the chemical computational model as follows: parallel execution (when two reactions involve distinct elements, they can occur simultaneously), mutual exclusion (a molecule cannot take part in more

Abstract machine	Chemistry
Data	Molecule
Multiset	Solution
Parallelism/nondeterminism	Brownian motion
Computation	Reaction

Table 2.1: The elements of the abstract machine and the ingredients of a chemical solution.

than one reaction at the same time), atomic capture (either all ingredients of the reaction are present, or no reaction occurs).

Having parallelism as a property, the chemical computational model is capable to express algorithms that are often reflected sequentially. Gamma (General Abstract Model for Multiset Manipulation) was the first significant chemical computational model in terms of being parallel, where multiset transformations assured these algorithms to be reflected in parallel. The main characteristics of Gamma were decentralization and nondeterminism. Depending on actual conditions, the program, or in other words a set of atomic and declarative rules could be applied simultaneously and independently that modified the multiset.

2.3 Membrane Systems, P Colonies - Preliminary Definitions

Membrane systems, or P systems (after their inventor Gh. Păun) are distributed, chemistry inspired and parallel abstract models, that are motivated by molecular membranes alongside cells. They are composed of disjoint membranes, which are placed in a common environment, encapsulated by a skin membrane. These membranes function as delimiters, containing multisets of chemical objects and several evolution rules that function only on

objects that belong to the membrane, the evolution rule was defined to. Rules get applied in parallel, in addition to the property, that they can specify where objects should be sent after their application, which can be done easily by labelling membranes. Moreover, objects are able to pass through membranes: either a contained or a vessel membrane can receive the object that was sent out by an evolution rule.

Membrane systems were analysed as mathematical objects, where computer scientists started to compare their computational power to Turing machines and investigate their potential in solving computationally hard problems. See [16, 3] for more details.

P colonies are variants of very simple membrane systems, which in turn are popular and widely studied computational models based also on the chemical computing paradigm, see [13, 14].

P colonies consist of a collection of very simple computing agents which interact in a shared environment. The environment and the colony members (the computing agents) are both described by multisets of objects. These multisets are processed by the colony members using transformation rules (rewriting rules) and communication rules (rules for exchanging objects between the colony members and the environment). Several rules are grouped into programs, which execute the rules they contain in parallel. A computation consists of a sequence of computational steps during which the colony members execute their programs in parallel, until the system reaches a halting state (where no program can be applied).

Different variants of P colonies were shown to be able to generate any recursively enumerable language, that is, to have the same computational power as Turing machines. To see how P colonies, as concrete examples of the chemical computing paradigm can be employed to overcome certain shortcomings of other types of models, we will consider the classic example of the dining philosophers, but first we introduce the formal notion of a P colony.

Let V be a finite alphabet, let the set of all words over V be denoted by V^* , and let ε be the empty word. We denote the number of occurrences of a symbol $a \in V$ in w by $|w|_a$.

If the set of non-negative integers is denoted by $\mathbb{N}$, then a multiset over a set V is a mapping $M : V \rightarrow \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M . The support of M is the set $\text{supp}(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, $\text{supp}(M) = \emptyset$. We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ε will represent the empty multiset which is also denoted by $\emptyset$.

We say that $a \in M$ if $M(a) \geq 1$, and the cardinality of M , $\text{card}(M)$ is defined as $\text{card}(M) = \sum_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \rightarrow \mathbb{N}$, $M_1 \subseteq M_2$ holds, if for all $a \in V$, $M_1(a) \leq M_2(a)$. The union of M_1 and M_2 is defined as $(M_1 \cup M_2) : V \rightarrow \mathbb{N}$ with $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$ for all $a \in V$, the difference is defined for $M_2 \subseteq M_1$ as $(M_1 - M_2) : V \rightarrow \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$.

Definition 1 A *P colony* of capacity k and with n cells, $k, n \geq 1$, is a construct $\Pi = (V, e, w_E, (w_1, P_1), \dots, (w_n, P_n), F)$ where V is an *alphabet*, the alphabet of the automaton, its elements are called *objects*; $e \in V$ is the *environmental object* of the automaton; $w_E \in (V - \{e\})^*$ is a string representing the multiset of objects different from e which is found in the environment initially; $(w_i, P_i), 1 \leq i \leq n$, is the *i-th cell*; and F is a set of *accepting configurations* of the system.

For each cell, (w_i, P_i) , $1 \leq i \leq n$, w_i is a multiset over V , it determines the initial contents of the cell, and its cardinality $|w_i| = k$ is called the *capacity* of the system; P_i is a set of *programs*, where every program is formed from k rules of the following types:

- *rewriting rules* of the form $a \rightarrow b$, where $a, b \in V$, or
- *communication rules* of the form $c \leftrightarrow d$, where $c, d \in V$.

The *computation* of a P Colony automaton starts in the initial configuration, and the

configurations are changed by the cells with the application of some of their programs. The programs either change the objects inside the cells (with rewriting rules) or exchange them for other objects with the environment (with communication rules). When a program is used, the rules are applied to the k objects inside the cell simultaneously, thus, the program produces k new objects in one computational step, they are either results of rewriting operations or brought inside the cell from the environment by communication rules.

A *configuration* of a P colony is an $(n + 1)$ -tuple $(u_E, u_1, \dots, u_n)$, where $u_E \in (V - \{e\})^*$ represent the multiset of objects different from e in the environment, and $u_i \in V^*$, $1 \leq i \leq n$, represents the contents of i -th cell. The *initial configuration* is given by $(w_E, w_1, \dots, w_n)$, the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as halting configurations, that is, configurations where no program can be applied in any of the cells.

In the dining philosopher problem, several (say n) philosophers sit at a round table, each having a plate of spaghetti in front of him, and there are forks between each plate (thus, if we have n philosophers and n plates, then there are n forks on the table). A philosopher can eat the spaghetti with two forks, thus, if one of them starts to eat, he has to pick up the forks on both sides of his plate which means that none of his neighbors will be able to eat at until he decides to stop. The problem is to determine a strategy for each philosopher to follow in such a way, that each of them gets to eat the spaghetti sooner or later, that is, where no deadlock occurs. (A deadlock can occur, for example, if the instructions to the philosophers contain to take any available fork, and wait for the other one until it becomes free. In this case, in a situation when each philosopher already has one of the forks in the hand, they keep on waiting for the other one to become available forever.)

Example 1 To represent this scenario, let us construct a cell for each philosopher, let us

have a symbol f_i for each fork $1 \leq i \leq n$, and let us use the symbols E_i and T_i , $1 \leq i \leq n$, representing the fact that the i th philosopher is eating or thinking, respectively.

Let $\Pi = (V, e, w_E, (w_1, P_1), \dots, (w_n, P_n), F)$ be a P colony of capacity $k = 3$, where $V = \{E_i, T_i, f_i, f_{n+1} \mid 1 \leq i \leq n\}$, $w_E = \{T_i, f_i \mid 1 \leq i \leq n\}$, and the sets of programs are defined as

$$\begin{aligned} P_i = & \{eat_i : \langle e \leftrightarrow f_i, e \leftrightarrow f_{i+1 \bmod n}, E_i \leftrightarrow T_i \rangle, \\ & think_i : \langle f_i \leftrightarrow e, f_{i+1 \bmod n} \leftrightarrow e, T_i \leftrightarrow E_i \rangle, \\ & keepeat_i : \langle f_i \rightarrow f_i, f_{i+1 \bmod n} \rightarrow f_{i+1 \bmod n}, T_i \rightarrow T_i \rangle, \\ & keepthink_i : \langle e \rightarrow e, e \rightarrow e, E_i \rightarrow E_i \rangle\} \end{aligned}$$

for $1 \leq i \leq n$.

Initially the environment contains n forks (the symbols f_i , $1 \leq i \leq n$), and the symbols T_i , $1 \leq i \leq n$, representing the fact the all philosophers are thinking, and the i th cell contains the 3 element multiset of symbols e, e, E_i .

The application of the program $keepthink_i$ in the i th cell does not change this situation, the application of eat_i , however, takes the two forks near the plates of the i th philosopher from the environment, and also exchanges the symbol T_i to E_i .

Note that the atomic capture property of the model (either all the reactants are present, or no reaction occurs) expresses the non-blocking property of the above description.

Chapter 3

On the Computational Power of Generalized P Colony Automata

As membrane systems, and P colonies in particular, are computational models working with multisets as their basic “data structure”, they are able to characterize sets of non-negative integers, or integer vectors in a straightforward manner. P colony automata were introduced in [2] with the aim of P colonies capable of describing also string languages. In order to do this, so called tape rules were introduced which beside executing the multiset transformation (as in P colonies) also trigger the reading of an input symbol from an input “tape”. Several computational modes of these automata were studied, each of them requiring that all tape rules which are applied in a computational step should request the reading of the same symbol from the tape. In this section, we relax this requirement, and allow the simultaneous application of tape programs which require the reading of different input symbols, by saying that instead of single symbols, multisets of symbols are read during each step of the computation.

3.1 Definitions

Now we define the notion of a generalized P colony automaton (genPCol automaton in short).

Definition 2 A *genPCol automaton* of capacity k and with n cells, $k, n \geq 1$, is a construct $\Pi = (V, e, w_E, (w_1, P_1), \dots, (w_n, P_n), F)$ where $V, e \in V$, $w_E \in (V - \{e\})^*$, $(w_i, P_i), 1 \leq i \leq n$, and F are defined as in the case of P colonies above.

The sets P_i of *programs*, is formed, however, from k rules of the following types:

- *tape rules* of the form $a \xrightarrow{T} b$, or $a \xleftrightarrow{T} b$, called rewriting tape rules and communication tape rules, respectively; or
- *nontape rules* of the form $a \rightarrow b$, or $c \leftrightarrow d$, called rewriting (nontape) rules and communication (nontape) rules, respectively, where $a, b, c, d \in V$.

A genPCol automaton works similarly to a P colony, but during the computation, the automaton processes an input word. A prefix (possibly consisting more than one symbols) of the yet non-read part of the input word is read during a configuration change: the processed prefix of the input corresponds to the multiset of symbols introduced by the tape rules of the system. This process is defined more precisely as follows.

A *configuration* of genPCol automaton is an $(n+2)$ -tuple $(u; u_E, u_1, \dots, u_n)$, where $u \in V^*$ is the unprocessed (unread) part of the input string, $u_E \in (V - \{e\})^*$ represent the multiset of objects different from e in the environment, and $u_i \in V^*$, $1 \leq i \leq n$, represents the contents of i -th cell. The *initial configuration* is given by $(w; w_E, w_1, \dots, w_n)$, the input word to be processed by the system and the initial contents of the environment and the cells. The elements of the set F of *accepting configurations* are given as configurations of the form $(\varepsilon; v_E, v_1, \dots, v_n)$, where

- $v_E \subseteq (V - \{e\})^*$ represents a multiset of objects different from e being in the

environment, and each

- $v_i \in V^*$, $1 \leq i \leq n$, is the contents of the i -th cell.

To describe the computation process formally, for any rule r we define the following multisets. Let $X \in \{T, \varepsilon\}$, and if $r = a \xrightarrow{X} b$, or $r = a \xleftrightarrow{X} b$, then let $left(r) = a$, $right(r) = b$. Let us extend this notation also for programs. For $\alpha \in \{left, right\}$ and for any program p , let $\alpha(p) = \bigcup_{r \in p} \alpha(r)$ where the union denotes multiset union (as defined above), and for a rule r and program $p = \langle r_1, \dots, r_k \rangle$, the notation $r \in p$ denotes the fact that $r = r_j$ for some j , $1 \leq j \leq k$. Moreover, for any tape program p we also define $read(p)$ as the multiset of symbols (different from e) on the right side of rewriting tape rules and on the left side of communication tape rules, that is, $read(p) = \bigcup_{r \in p, r=a \xrightarrow{T} b, b \neq e} right(r) \cup \bigcup_{r \in p, r=a \xleftrightarrow{T} b, a \neq e} left(r)$. We also denote by $export(p)$ and $import(p)$ the multisets $export(p) = \bigcup_{r \in p, r=a \xleftrightarrow{X} b, a \neq e} a$ and $import(p) = \bigcup_{r \in p, r=a \xleftrightarrow{X} b} b$, and by $create(p)$ the multiset $create(p) = \bigcup_{r \in p, r=a \xrightarrow{X} b} b$. So by $export(p)$ and $import(p)$, that were defined for communication rules of a given p program, we indicate the objects that are sent out to the environment and brought inside the cell, respectively. Whereas $create(p)$ is the multiset of symbols produced by the rewriting rules of program p .

Thus, $left(r)$ and $right(r)$ are the multisets consisting of the symbol on the left or right side of the rule r . For a program p , $left(p)$ and $right(p)$ are the collection (multiset) of symbols on the left or right sides of the rules in the program p . Finally, $read(p)$ is the collection (multiset) of symbols (different from e) on the right side of rewriting tape rules or the left side of communication tape rules, $export(p)$ is the multiset of symbols (different from e) exported to the environment from the cells by the program p , and $import(p)$ is the multiset of symbols brought in the cell from the environment by rule p .

Let the programs of each P_i be labelled in a one-to-one manner by labels from the set $lab(P_i)$, $1 \leq i \leq n$, $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$. In the following, for the sake of brevity, if no confusion arises, we designate programs and their labels with the same letters, thus,

for a label $p \in \text{lab}(P_i)$, we also write $p \in P_i$.

Let $c = (u; u_E, u_1, \dots, u_n)$ be a configuration of a genPCol automaton Π , and let $U_E = u_E \cup \{e, e, \dots\}$, thus, the multiset of objects found in the environment (together with the infinite number of e s which are always present). We call a *set of programs*, P_c , *applicable in configuration c* , if the following conditions hold.

- At most one program is selected for each cell, that is, if $p, p' \in P_c, p \neq p'$ and $p \in P_i, p' \in P_j$, then $i \neq j$;
- the selected programs are applicable in the cells (the left sides of the rules contain the same symbols that are present in the cell), that is, for each $p \in P_c$, if $p \in P_i$ then $\text{left}(p) = u_i$;
- the symbols which are brought inside the cells by the programs are present in the environment, that is, $\bigcup_{p \in P_c} \text{import}(p) \subseteq U_E$;
- P_c is maximal, that is, if any other program is added to it, then some of the above conditions are not satisfied.

A configuration $c = (u; u_E, u_1, \dots, u_n)$, $a \in V$, is *changed* to a configuration $c' = (u'; u'_E, u'_1, \dots, u'_n)$ by applying the set P_c of applicable programs if the following properties hold:

- If there is a $p \in P_c$ such that $p \in P_i$, then $u'_i = \text{create}(p) \cup \text{import}(p)$, otherwise $u'_i = u_i$, $1 \leq i \leq n$;
- $U'_E = U_E - \bigcup_{p \in P_c} \text{import}(p) \cup \bigcup_{p \in P_c} \text{export}(p)$ (where U'_E again denotes $u'_E \cup \{e, e, \dots\}$ with an infinite number of e s); and
- if $\bigcup_{p \in P_c} \text{read}(p) = v$, then $u = \text{perm}(v)u'$ where $\text{perm}(v)$ denotes a string representing the multiset composed of the symbols of v , or in other words, $\text{perm}(v)$ is a string obtained by a permutation of the symbols of string v .

This means that the symbols which are “read” by the tape programs in one computational step, are read from the prefix of the input in the same computational step of the automaton. Thus, more than one symbols are read from the input in each step, those which are read by the tape programs applied in that step. Note that several different strings might be read during the same computational step, since different permutations of a series of symbols correspond to the same multiset.

The *language accepted by a genPCol automaton* Π as above is defined as the set of strings which can be read during a successful computation:

$$L(\Pi) = \{w \in V^* | (w; w_E, w_1, \dots, w_n) \text{ can be} \\ \text{transformed by } \Pi \text{ into } (\varepsilon; v_E, v_1, \dots, v_n) \in F\}.$$

The general idea behind the above definitions is that instead of the different computational modes used in [2], we have a system with programs and we apply the programs in the maximally parallel way, as usual in P colonies, that is, in each moment, every agent must non-deterministically choose and apply one of its applicable programs, until no agent can apply any of its programs anymore. Then we look at those rules which were tape rules (in the applied set of programs) and collect all the symbols that they “read”: this multiset (of the collected symbols) should be read from the input, thus, a prefix of the not yet read part of the input string is read (which contains exactly these symbols in some order).

Now we repeat an example from [2] in order to demonstrate the differences between genPCol automata and the variant of the notion considered there.

Example 2 Let $\Pi = (\{a, b, c\}, e, w_E, (w_1, P_1), (w_2, P_2), (w_3, P_3), F)$ be a genPCol au-

tomaton, where the sets of programs are defined as

$$\begin{array}{ccc}
P_1 & P_2 & P_3 \\
\hline
p_1 : \langle e \xrightarrow{T} b; e \leftrightarrow a \rangle, & p_A : \langle e \xleftrightarrow{T} a; e \leftrightarrow a \rangle, & p_I : \langle e \xrightarrow{T} a; e \leftrightarrow a \rangle, \\
p_2 : \langle e \xrightarrow{T} a; e \rightarrow a \rangle, & p_B : \langle e \xleftrightarrow{T} b; e \leftrightarrow a \rangle, & p_{II} : \langle e \xrightarrow{T} a; e \rightarrow b \rangle, \\
p_3 : \langle e \leftrightarrow a; e \leftrightarrow e \rangle, & p_C : \langle e \rightarrow a; e \leftrightarrow e \rangle, & p_{III} : \langle e \rightarrow a; e \leftrightarrow e \rangle,
\end{array}$$

and $c = (w; aaccc, ee, ee, ee)$ is the current configuration of Π .

The set of programs $\{p_1, p_A, p_{II}\}$ is not applicable because there are not enough a s in the environment. The set of programs $\{p_1, p_I\}$ is not maximal, since p_C could be added to it. The set of programs $\{p_1, p_C, p_I\}$ is maximal, so these programs could be applied in the current configurations of Π , and during their application the multiset consisting of one a and one b is read from the input, thus, $w = abw'$ or $w = baw'$ must hold for some w' , and then the resulting configuration is $c' = (w'; ccc, ba, ae, aa)$.

The next example (also similar to the one from [2]) demonstrates that all regular languages can be accepted also by genPCol automata.

Example 3 Let $L \subseteq \Sigma^*$ be a regular language, and let $M = (\Sigma, Q, \delta, q_0, F)$ be a finite automaton with $L(M) = L$, with alphabet Σ , set of states Q , initial state q_0 , set of final states F , and transition function $\delta : \Sigma \times Q \rightarrow Q$.

Consider the genPCol automaton $\Pi = (\Sigma \cup \{q_I\} \cup Q, e, w_E, (w, P), F')$ of capacity two, where $q_I \notin \Sigma \cup Q$. It is not difficult to see that Π simulates the computations of M , that is, $L(\Pi) = L(M)$ with initial environment $w_E = \bigcup_{a \in \Sigma} \{a, a\}$, initial cell contents $w = eq_I$, set of rules

$$\begin{aligned}
P = & \{ \langle e \xleftrightarrow{T} a, q_I \rightarrow q_0 \rangle \mid a \in \Sigma \} \cup \\
& \{ \langle x \xleftrightarrow{T} a, q \rightarrow q' \rangle \mid \text{for all } x \in \Sigma \text{ such that } \delta(x, q) = q' \text{ for some } q, q' \in Q \},
\end{aligned}$$

and set of final configurations

$$F' = \{(\varepsilon; w_E - \{x\}, xq_f) \mid \text{for all } x \in \Sigma, \text{ and } q_f \in F\}.$$

The next example demonstrates how non-context-free languages can be accepted by genPCol automata.

Example 4 Let $\Pi = (\{a, b, c\}, e, \emptyset, (ee, P), F)$ be a genPCol automaton where

$$\begin{aligned} P = & \{ \langle e \rightarrow a, e \xleftrightarrow{T} e \rangle, \langle e \rightarrow a, a \xleftrightarrow{T} e \rangle, \\ & \langle e \rightarrow b, a \xleftrightarrow{T} e \rangle, \langle e \rightarrow b, b \xleftrightarrow{T} a \rangle, \langle a \rightarrow b, b \xleftrightarrow{T} a \rangle, \\ & \langle a \rightarrow c, b \xleftrightarrow{T} a \rangle, \langle a \rightarrow c, a \xleftrightarrow{T} b \rangle, \langle b \rightarrow c, c \xleftrightarrow{T} b \rangle, \\ & \langle b \rightarrow e, c \xleftrightarrow{T} b \rangle, \langle b \rightarrow e, e \xleftrightarrow{T} c \rangle, \langle c \rightarrow e, e \xleftrightarrow{T} c \rangle \} \end{aligned}$$

with all the communication rules being tape rules. Let also $F = (\varepsilon; \emptyset, ec)$ be the final configuration. It is not difficult to see that this system generates the language $L(\Pi) = \{a^n b^n c^n \mid n \geq 1\}$.

Now we introduce subclasses of the model based on the presence of tape rules in the programs. Let the class of languages accepted by generalized PCol automata where all the communication rules are tape rules be denoted by $\mathcal{L}(\text{genPCol}, \text{com-tape})$, the class of languages accepted by generalized PCol automata with all programs having at least one tape rule by $\mathcal{L}(\text{genPCol}, \text{all-tape})$, and the class of languages accepted by generalized PCol automata where programs with any kinds of rules are allowed by $\mathcal{L}(\text{genPCol}, *)$. Let also $\mathcal{L}(\text{REG})$, $\mathcal{L}(\text{CF})$, and $\mathcal{L}(\text{RE})$ denote the classes of regular, context-free, and recursively enumerable languages respectively.

Proposition 1

1. $\mathcal{L}(\text{genPCol}, \text{com-tape}) \cup \mathcal{L}(\text{genPCol}, \text{all-tape}) \subseteq \mathcal{L}(\text{genPCol}, *)$,
2. $\mathcal{L}(\text{genPCol}, \text{com-tape}) \cap \mathcal{L}(\text{genPCol}, \text{all-tape}) - \mathcal{L}(\text{CF}) \neq \emptyset$,
3. $\mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{genPCol}, \text{com-tape}) \cap \mathcal{L}(\text{genPCol}, \text{all-tape})$.

Proof. The first statement is the consequence of the definitions. Whereas the second statement is an implication of Example 4, where we showed that a context-sensitive/non

context-free language, namely $\{a^n b^n c^n \mid n \geq 1\}$ could be generated by a genPCol automaton, where in addition to all programs having at least one tape rule, all the communication rules were tape rules. Considering the second statement with Example 2, where we showed that a genPCol automaton can accept an arbitrary regular language by simulating a finite automaton that accepts this language, the last statement also holds. $\square$

3.2 GenPCol Automata with Restricted Programs

Now we examine the class of languages characterized by genPCol automata where all communication rules are tape rules, or where there is at least one tape rule in each program.

First, we recall a notion from [4]. A nondeterministic Turing machine with a one-way input tape is *restricted logarithmic space bounded* if for every accepted input of length n , there is an accepting computation where the number of nonempty cells on the work-tape(s) is bounded by $O(\log d)$ where $d \leq n$ is the number of input tape cells already read, that is, the *distance* of the reading head from the left end of the one-way input tape. The class of languages accepted by such machines is denoted by r-1LOGSPACE. In [4] it is shown that this class is strictly included in 1LOGSPACE, the class of languages accepted by Turing machines with a one-way input tape using logarithmic space on the worktapes.

Theorem 2 $\mathcal{L}(\text{genPCol, com-tape}) \cup \mathcal{L}(\text{genPCol, all-tape}) \subseteq \text{r-1LOGSPACE}$.

Proof. To prove the theorem, we first describe how a Turing machine using restricted one-way logarithmic space can simulate $\Pi = (V, e, w_E, (w_1, P_1), \dots, (w_n, P_n), F)$, a genPCol automaton.

Let us define the set of transitions of Π as $Tr \subseteq P$, such that, $p_1, p_2 \in Tr$ and $p_1 \in P_i$, $p_2 \in P_j$ implies $i \neq j$ (at most one program is executed by each cell during a transition).

For any transition $t \in Tr$, we define the following multisets:

- $read(t) \in V^*$ is the multiset of symbols read by Π during transition t from the input, that is, $read(t) = \bigcup_{p \in Tr} read(p)$ (where $\bigcup$ denotes union in the “multiset sense”),
- $export(t) \in V^*$ is the multiset of symbols sent out by the cells of Π to the environment during transition t , that is, $export(t) = \bigcup_{p \in Tr} export(p)$,
- $import(t) \in V^*$ is the multiset of symbols imported by the cells of Π from the environment during transition t , that is, $import(t) = \bigcup_{p \in Tr} import(p)$.

Let us also define $blocked(t) \subseteq \{1, \dots, n\}$ as the set of indices of those component cells which do not apply any program during transition t , that is, $i \in blocked(t)$ if and only if, there is no $p \in Tr$ such that $p \in P_i$.

Finally, let $inside(t) \in (V^k \cup \{\$\})^n$ be the set of internal configurations compatible with transition t , that is, the n -tuples of those k element multisets which must be present inside the n component cells of Π in order for t to be applicable. If a cell does not apply any programs during transition t , then $\$$ is the corresponding element of the the n -tuple $inside(t)$.

Now we construct a one-way Turing machine M which simulates the work of Π using restricted logarithmic workspace. Let $M = (m, V, \Gamma, Q, \delta_M, q_0, F')$ where

- $m = 2 \cdot |V|$ is the number of work-tapes, thus, M has two tapes for each element of the object alphabet V ,
- V is the input alphabet,
- $\Gamma = \{0, \dots, 9\}$ is the work-tape alphabet,
- $Q = \{q_0\} \cup \{[t, u_1, \dots, u_n] \mid t \in Tr, u_i \in V^k, 1 \leq i \leq n\}$ is the set of internal states with q_0 being the initial state, and $F' = \{q_f\}$ the set of accepting states,

- $\delta_M : V \cup \{\varepsilon\} \times Q \times \Gamma^m \rightarrow Q \times \Gamma^m \times \{left, right, none\}^m$ is the transition relation of M .

The machine M can record the configurations of Π by storing the contents $u_i \in V^k$ of cell i , $1 \leq i \leq n$ in the internal states $q \in Q$ and by recording on the corresponding work-tape the number of occurrences of each object from V in the environment. The computations of Π are simulated by M in several phases.

First, M nondeterministically chooses a transition $t \in Tr$ which is also recorded in the internal state, and then it reads the elements of $read(t)$ from the input tape in a nondeterministically chosen order in a series of computational steps.

In the next phase, M checks whether the chosen transition t of Π is applicable in the current configuration of the genPcol automaton. To this aim, M first compares the non-\$ elements of $inside(t)$ to the contents of the corresponding cells: a program can only be applicable if the objects which are to be rewritten or exported to the environment are present inside the cell.

Then for each $a \in V$, M writes the numbers $|import(t)|_a$ on the second work-tape corresponding to a , and compares them with the number of the first tape making sure that there are enough objects in the environment for the execution of transition t . At this point, for each $i \in blocked(t)$, M examines the programs of P_i and compares them to the contents of the corresponding cell i : if it finds a program p with rules having left hand sides which are able to process all the objects in the cell, then it makes sure that the objects which remain after subtracting the multiset $import(t)$ from the environment are not sufficient for the application of p (the numbers of different objects in the environment and the number of different objects in $import(t)$ are recorded on the two sets of work-tapes corresponding to each symbol).

So at the end of this phase, M “knows” that all the programs in transition t can be applied,

and that none of the cells which are blocked according to t can apply any programs, in short, that transition t is applicable in the currently simulated configuration of Π .

In the last phase, transition t is applied, that is, the cell contents are updated according to the programs of t , and the contents of the environment is updated by adding the multiset $\text{export}(t)$ and subtracting the multiset $\text{import}(t)$. Now M checks whether the complete input is processed, and if so, whether the recorded configuration of Π corresponds to a final configuration, in which case it enters the state q_f and accepts the input. Otherwise the simulation of another transition of Π can be performed.

To see that the above described simulation requires restricted logarithmic space, note that in the case when all programs contain at least one tape rule Π reads a nonempty (but bounded) multiset of symbols in each computational step, thus, the number of objects present in any configuration of Π can be bounded by a linear function of the number of executed computational steps. This is also true for systems where all communication rules are tape rules, because of the fact that if no symbols are read from the tape, then no symbols enter the environment from any of the cells, which means that the Turing machine workspace needed to record the P colony configuration does not increase during the simulation of these steps. Thus, the number of symbols in any configuration can be bounded by a linear function of the number of processed input symbols, which means that the configuration can be recorded in logarithmic space with respect to the distance of the input head of the Turing machine from the leftmost cell of the one-way read only input tape. $\square$

3.3 GenPCol Automata Computing in the Non-Restricted Mode

In this section we show that if we consider the non-restricted functioning mode of the GenPCol automata which does not require the automata to have tape rules in all of its programs, nor do we restrict its rules by considering only communication rules as tape rules, then GenPCol automata characterizes the class of recursively enumerable languages. First we recall the notion of a two-counter machine which will be used in the proof.

A *two-counter machine*, see [10], $M = (\Sigma \cup \{Z, B\}, Q, q_0, q_F, Tr)$ is a 3-tape Turing machine where Σ is an *alphabet*, Q is a set of *internal states* with $q_0, q_F \in Q$ being the initial and the final states, and Tr is a set of *transition rules*. The machine has a read-only input tape and two semi-infinite storage tapes which are used as counters. The alphabet of the storage tapes contains only two symbols, Z and B (blank), while the alphabet of the input tape is $\Sigma \cup \{B\}$. The symbol Z is written on the first, leftmost cells of the storage tapes which are scanned initially by the tape heads. An integer t can be stored by moving a tape head t cells to the right of Z . A stored number can be incremented or decremented by moving the tape head right or left. The machine is capable of checking whether a stored value is *zero* or not by looking at the symbol scanned by the tape heads. If the scanned symbol is Z , then the value stored in the corresponding counter is *zero*.

Without the loss of generality, we assume that two-counter machines check and modify only one of their counters during any transition, thus, the rule set R contains transition rules of the form $(q, x, i, \alpha) \rightarrow (q', \beta)$ where $x \in \Sigma \cup \{B\} \cup \{\varepsilon\}$ corresponds to the symbol scanned on the input tape in state $q \in Q$, and $\alpha \in \{Z, B\}$, $i \in \{1, 2\}$ correspond to the symbols scanned on the i th storage tape. By a rule of the above form, M enters state $q' \in Q$, and the i th counter is modified according to $\beta \in \{-1, 0, +1\}$. If $x \in \Sigma \cup \{B\}$, then the machine was scanning x on the input tape, and the head moves one cell to the

right; if $x = \varepsilon$, then the machine performs the transition irrespective of the scanned input symbol, and the reading head does not move.

A word $w \in \Sigma^*$ is accepted by the two-counter machine if starting in the initial state q_0 , the input head reaches and reads the rightmost non-blank symbol on the input tape, and the machine is in the accepting state q_f . Two-counter machines are computationally complete; they are just as powerful as Turing machines.

Theorem 3

$$\mathcal{L}(\text{genPCol}, *) = \mathcal{L}(\text{RE})$$

Proof. Let $L \subseteq \Sigma^*$ be an arbitrary recursively enumerable language, and let $M = (\Sigma \cup \{Z, B\}, Q, q_0, q_f, Tr)$ be a two-counter machine as above with $L = L(M)$.

Let us construct the GenPCol automaton $\Pi = (V, e, w_E, (w_1, P_1), (w_2, P_2), F)$ where $V = \Sigma \cup Q \cup \{t, t', t'', t''' \mid t \in Tr\} \cup \{c_1, c_2, A\}$, $w_1 = q_0 e$, $w_2 = ee$, $F = \{(\varepsilon; u, q_f e, ee) \mid u \in V^*\}$, and the sets of programs are defined as follows.

For any $\alpha \in \{B, Z\}$, $\beta \in \{-1, 0, +1\}$, we define the disjoint sets of transitions $Tr_{\alpha, \beta} \subseteq Tr$ as follows: $t \in Tr_{\alpha, \beta}$, if and only if, $t : (q, x, i, \alpha) \rightarrow (q', \beta)$, $x \in \Sigma \cup \{\varepsilon\}$, $i \in \{1, 2\}$. Thus, $Tr = Tr_{B, -1} \cup Tr_{B, 0} \cup Tr_{B, +1} \cup Tr_{Z, 0} \cup Tr_{Z, +1}$.

Now we define the sets of programs as

$$P_1 = \bigcup_{t \in Tr} P_{1,t} \text{ and } P_2 = \bigcup_{t \in Tr} P_{2,t},$$

where for $t \in (Tr_{B, -1} \cup Tr_{B, 0})$ we have

$$\begin{aligned} P_{1,t} = & \{p_{t,1} : \langle q \rightarrow t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow c_i; r_{t,2} \rangle, p_{t,3} : \langle t' \rightarrow t''; c_i \rightarrow e \rangle, \\ & p_{t,4} : \langle t'' \rightarrow e; e \leftrightarrow t''' \rangle, p_{t,5} : \langle t''' \rightarrow q'; e \rightarrow e \rangle\}, \end{aligned}$$

where $r_{t,1}$ and $r_{t,2}$ are the rules $e \xrightarrow{T} a$ and $a \rightarrow t'$, respectively, if $t \in Tr$ is such, that $x = a \in \Sigma$, otherwise, if $x = \varepsilon$, then $r_{t,1} = e \rightarrow e$ and $r_{t,2} = e \rightarrow t'$.

If $t \in Tr_{B,+1}$, then we have

$$P_{1,t} = \{p_{t,1} : \langle q \rightarrow t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow c_i; r_{t,2} \rangle, p_{t,3} : \langle t' \rightarrow t''; c_i \rightarrow c_i \rangle, \\ p_{t,4} : \langle t'' \rightarrow e; c_i \leftrightarrow t''' \rangle, p_{t,5} : \langle t''' \rightarrow q'; e \rightarrow e \rangle\}$$

with $r_{t,1}$ and $r_{t,2}$ as above.

For these types of transitions, the set P_2 is defined as follows. If $t \in Tr_{B,-1}$, then we have

$$P_{2,t} = \{p_{t,6} : \langle e \leftrightarrow t; e \rightarrow e \rangle, p_{t,7} : \langle t \rightarrow t'''; e \rightarrow e \rangle, p_{t,8} : \langle t''' \leftrightarrow e; e \rightarrow e \rangle\},$$

otherwise, if $t \in (Tr_{B,0} \cup Tr_{B,+1})$, then

$$P_{2,t} = \{p_{t,6} : \langle e \leftrightarrow t; e \rightarrow c_i \rangle, p_{t,7} : \langle t \rightarrow t'''; c_i \leftrightarrow e \rangle, p_{t,8} : \langle t''' \leftrightarrow e; e \rightarrow e \rangle\}.$$

Now, if $t \in Tr_{Z,0}$, then we have in P_1

$$P_{1,t} = \{p_{t,1} : \langle q \rightarrow t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow e; r_{t,2} \rangle, p_{t,3} : \langle e \leftrightarrow c_i; t' \rightarrow A \rangle, \\ p_{t,4} : \langle t' \rightarrow e; e \leftrightarrow t'' \rangle, p_{t,5} : \langle e \leftrightarrow e; t'' \rightarrow q' \rangle\}$$

where $r_{t,1}$ and $r_{t,2}$ are the rules $e \xrightarrow{T} a$ and $a \rightarrow t'$, respectively, if the transition is such, that the input symbol is $x = a \in \Sigma$, otherwise if $x = \varepsilon$, then $r_{t,1} = e \rightarrow e$ and $r_{t,2} = e \rightarrow t'$.

If $t \in Tr_{Z,+1}$, then

$$P_{1,t} = \{p_{t,1} : \langle q \rightarrow t; r_{t,1} \rangle, p_{t,2} : \langle t \leftrightarrow e; r_{t,2} \rangle, p_{t,3} : \langle e \leftrightarrow c_i; t' \rightarrow A \rangle, \\ p_{t,4} : \langle t' \rightarrow c_i; e \leftrightarrow t'' \rangle, p_{t,5} : \langle c_i \leftrightarrow e; t'' \rightarrow q' \rangle\}$$

where $r_{t,1}$ and $r_{t,2}$ are as above.

The set P_2 contains only two programs in both cases, these are defined as

$$P_{2,t} = \{p_{t,6} : \langle e \leftrightarrow t; e \rightarrow t'' \rangle, p_{t,7} : \langle t \rightarrow e; t'' \leftrightarrow e \rangle\}$$

for all $t \in (Tr_{Z,0} \cup Tr_{Z,+1})$.

The PCol automaton Π simulates the work of the two-counter machine M by reading the input symbols with its tape programs and keeping track of the contents of the i th counter as the number of c_i , $i \in \{1, 2\}$ objects present in the environment.

Each transition of M is simulated separately. One of the symbols inside the first cell of Π is from Q , it corresponds to the internal state of M during the simulation process. This symbol is changed through a series of programs into the symbol $q' \in Q$ if and only if M changes its state from q to q' while the counter contents are also checked and modified with an interplay of programs from the two cells of Π .

For instance, let us see step by step how the simulation works when a transition rule $t \in Tr_{B,0}$ of M is considered. At the beginning of the simulation, the contents of the first and second cells are the following: qe and ee , respectively. Remember, that the contents of the i th counter is kept track in the environment in the form of c_i , $i \in \{1, 2\}$ objects. Since $t \in Tr_{B,0}$, t has the following form: $t : (q, x, i, B) \rightarrow (q', 0)$, $x \in \Sigma \cup \{\varepsilon\}$, $i \in \{1, 2\}$, thus there is at least one c_i object in the environment. The following two tables summarize the contents of the cells that are modified with an interplay of programs, whilst simulating t . Table 3.1 represents the case, where $x \in \Sigma$, or in other words where an input character is read, while table 3.2 represents the scenario, where $x = \varepsilon$.

It clearly seems, that given the transition rule $t : (q, x, i, B) \rightarrow (q', 0)$ the q symbol inside the first cell of Π is changed to q' if and only if M changes its state from q to q' , while the cardinality of c_i remains the same in the environment of Π .

It is easy to see that other types of transition rules $\{t | t \in Tr_{B,-1} \cup Tr_{B,+1} \cup Tr_{Z,0} \cup Tr_{Z,+1}\}$ can be simulated with a similar terminology and thus the GenPCol automaton Π may reach a final configuration after reading the whole input if and only if the simulated two-counter machine is able to reach the internal state q_f after processing the same input string using its transitions from Tr . $\square$

step	w_1	w_2	environment	unread part of tape (u)	applied program
1.	qe	ee	c_i	au_{rest}	$p_{t,1} : \langle q \rightarrow t; e \xrightarrow{T} a \rangle$
2.	ta	ee	c_i	u_{rest}	$p_{t,2} : \langle t \leftrightarrow c_i; a \rightarrow t' \rangle$
3.	$c_i t'$	ee	t	u_{rest}	$p_{t,3} : \langle t' \rightarrow t''; c_i \rightarrow e \rangle$ $p_{t,6} : \langle e \leftrightarrow t; e \rightarrow c_i \rangle$
4.	et''	tc_i	ϵ	u_{rest}	$p_{t,7} : \langle t \rightarrow t'''; c_i \leftrightarrow e \rangle$
5.	et''	$t'''e$	c_i	u_{rest}	$p_{t,8} : \langle t''' \leftrightarrow e; e \rightarrow e \rangle$
6.	et''	ee	$c_i t'''$	u_{rest}	$p_{t,4} : \langle t'' \rightarrow e; e \leftrightarrow t''' \rangle$
7.	$t'''e$	ee	c_i	u_{rest}	$p_{t,5} : \langle t''' \rightarrow q'; e \rightarrow e \rangle$
8.	$q'e$	ee	c_i	u_{rest}	

Table 3.1: Simulation of $t : (q, x, i, B) \rightarrow (q', 0)$, where $x \in \Sigma$

step	w_1	w_2	environment	unread part of tape (u)	applied program
1.	qe	ee	c_i	u_{rest}	$p_{t,1} : \langle q \rightarrow t; e \rightarrow e \rangle$
2.	te	ee	c_i	u_{rest}	$p_{t,2} : \langle t \leftrightarrow c_i; e \rightarrow t' \rangle$
3.	$c_i t'$	ee	t	u_{rest}	$p_{t,3} : \langle t' \rightarrow t''; c_i \rightarrow e \rangle$ $p_{t,6} : \langle e \leftrightarrow t; e \rightarrow c_i \rangle$
4.	et''	tc_i	ϵ	u_{rest}	$p_{t,7} : \langle t \rightarrow t'''; c_i \leftrightarrow e \rangle$
5.	et''	$t'''e$	c_i	u_{rest}	$p_{t,8} : \langle t''' \leftrightarrow e; e \rightarrow e \rangle$
6.	et''	ee	$c_i t'''$	u_{rest}	$p_{t,4} : \langle t'' \rightarrow e; e \leftrightarrow t''' \rangle$
7.	$t'''e$	ee	c_i	u_{rest}	$p_{t,5} : \langle t''' \rightarrow q'; e \rightarrow e \rangle$
8.	$q'e$	ee	c_i	u_{rest}	

Table 3.2: Simulation of $t : (q, x, i, B) \rightarrow (q', 0)$, where $x = \epsilon$

Chapter 4

Conclusion

Before drawing some general conclusions, we compare the generalized P colony automata model with similar models studied earlier in the membrane computing literature. To this aim we would like to give a short overview of the motivation behind the introduction of the model of P automata.

4.1 Comparison with Similar Models

P systems were first introduced as a membrane structure with objects and rules without any explicit reference to the “environment” in which the system is placed. This changed with the introduction of so called symport/antiport P systems which are systems with communication rules only, thus, in order to be able to execute meaningful computations (computations which are more complex than that of a finite automaton), they needed to rely on the environment for being the source of new objects to be introduced into the system, see [15]. The role of the environment became even more significant with the introduction of P automata, which are not only symport/antiport systems, but the

	all-tape/com-tape	unrestricted
Sequential P automata	r-1LOGSPACE	$\mathcal{L}(\text{RE})$
Parallel P automata	$\mathcal{L}(\text{CS})$	$\mathcal{L}(\text{RE})$
genPCol automata	$\mathcal{L}(\text{REG}) \subset \cdot \subseteq \text{r-1LOGSPACE}$	$\mathcal{L}(\text{RE})$

Table 4.1: The computational power of genPCol automata compared to similar variants of P automata. ($\mathcal{L}(\text{CS})$ denotes the class of context-sensitive languages.)

accepted language is defined through the interaction of the machine with its environment: The accepted words (strings) are obtained from the sequence of multisets which enter the system from the environment during a successful computation, [7].

P automata and generalized P colony automata are closely related. As the languages of P automata are defined by the multisets they read from the environment, their basic variant works as genPCol automata in the all-com mode. If they are allowed to discard some of the symbols which they read from the environment, then they are allowed to not read from the input string in certain computational steps, thus their operation resembles the application of programs without tape rules, that is, the unrestricted *-mode of operation of genPCol automata.

In Table 4.1 we compare our results with the power of the above described P automata variants, also considering them with sequential (one rule in each region in each step) or parallel (as many rules as possible) rule application.

In Table 4.2 we also compare the number of objects that these computational models need in order to achieve the corresponding amount of power. Note that we consider the number of objects inside the system. A genPCol automaton may be able to describe nonregular languages with just a finite number of objects inside, because it manipulates its environment in such a sophisticated manner that the “memory” or workspace it uses is provided by the objects in the environment.

	all-tape/com-tape	unrestricted
Sequential P automata	$O(n)$	arbitrary
Parallel P automata	$O(2^n)$	arbitrary
genPCol automata	$O(1)$	$O(1)$

Table 4.2: The number of objects inside the cells of genPCol automata compared to the number of objects needed by similar variants of P automata during computations with input length of n .

4.2 Future Work

As we have seen, primitive chemical components' behaviour can be really complex, so that they are even able to accept the class of recursively enumerable languages, given that there are no restrictions on the content of programs.

Although, we have seen that genPCol automata are better from the point of view of simplicity than several similar P automata variants, the fact whether this model is significantly better in describing the problems arising at the study of Future Internet is still left to be examined. Future research includes looking into other modes, especially where we need every program to have at least one tape rule, determining their exact computational power.

Chapter 5

Summary

As aforementioned previously, the “Internet of Services”, where a myriad of services and users are interacting autonomously would require relentlessly new approaches, such as new architectures, new ways of managing data, new interfaces and new ways of integrating all the different Internet entities. Service Oriented Architecture technologies are getting favored in the context of Future Internet. In order to express service coordination, decentralized, parallel and self-organizing systems are needed, such as nature inspired, unconventional computing models that have recently become vogue.

A brief introduction was made to the chemical computational model and membrane systems (P systems) , P colonies in general. The chemical computational model is able to relax the artificial sequentializing of algorithms by having parallelism as a property. P systems are parallel abstract models, that take the inspiration of chemistry into account.

The merits of the writing were introduced through necessary definitions and some examples to the reader, that is, the generalized P colony automata (GenPCol Automata) and three of their variants. Two theorems were proven regarding the computational capabilities of the variants of GenPCol Automata: The class of languages accepted by the

subclasses, where programs are restricted is in the class of languages accepted by Turing machines with a one-way input tape using logarithmic space on the worktapes. The next theorem states that the class of languages accepted by the subclass, where we allow programs to consist no tape rules at all is equal to the class of recursively enumerable languages.

We compared the computational power and the number of objects inside each cell of GenPCol Automata with the corresponding results on related variants of P automata (sequential and parallel). We have seen that GenPCol Automata have an essentially more simple description than the other P system variants, that is, the number of objects inside the system is bounded by a finite constant. If we consider an unrestricted ruleset, then all the automata models resemble to each other, that is, they accept the class of recursively enumerable languages, otherwise their computational power differ.

Acknowledgement

This research was supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP 4.2.4. A/2-11-1-2012-0001 'National Excellence Program'.

Bibliography

- [1] Banâtre, J.P., Priol, T. (2009). Chemical programming of future service-oriented architectures. *Journal of Software* 4(7) (2009), 738-746.
- [2] Cienciala, L., Ciencialová, L., Csuhaj-Varjú, E., and Vaszil, Gy. (2010). *PCol automata: Recognizing strings with P colonies*, Eighth Brainstorming Week on Membrane Computing, Sevilla, February 1-5, 2010, (M. A. Martínez del Amor, Gh. Păun, I. Pérez Hurtado, and A. Riscos Nuñez, editors, RGNC Reports 1/2010, Fénix Editora, 2010), pp 65–76.
- [3] Ciobanu, G., Lucanu, D. (2005). *Cellular meta-programming over membranes*, Unconventional Programming Paradigms, International Workshop UPP 2004, Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers, (J.P. Banâtre, P. Fradet, J.L. Giavitto, O. Michel, editors, volume 3566 of *Lecture Notes in Computer Science*, Springer, 2005), pp 196-206.
- [4] Csuhaj-Varjú, E., Ibarra, O.H., and Vaszil, Gy. (2005). *On the computational complexity of P automata*, 10th International Workshop on DNA Computing, (C. Ferretti, G. Mauri, and C. Zandron, editors, volume 3384 of *Lecture Notes in Computer Science*, Springer, 2005), pp 76–89.
- [5] Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Păun, Gh., and Vaszil, Gy. Computing with cells in environment: P colonies. *Journal of Multi-Valued Logic and Soft*

Computing, 12 (2006), 201–215.

- [6] Csuhaj-Varjú, E., Oswald, M., and Vaszil, Gy. (2010) P automata. In [16], chapter 6, pp 144–167.
- [7] Csuhaj-Varjú, E., Vaszil, Gy. (2002). P automata or purely communicating accepting P systems. In: Gh. Paun, G. Rozenberg, A. Salomaa and C. Zandron (eds.), *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers. Lecture Notes in Computer Science 2597*, Springer, Berlin, 2003, 219–233.
- [8] Freund, R., Oswald, M. A short note on analysing P systems. *Bulletin of the EATCS*, 78 (2002), 231–236.
- [9] Di Napoli, C., Giordano, M., Németh, Zs., and Tonellotto, N. (2010). *Using chemical reactions to model service composition*. Proceedings of the Second International Workshop on Self-Organizing Architectures (SOAR '10), ACM, New York, NY, USA, pp 43–50.
- [10] Fischer, P.C. Turing machines with restricted memory access. *Information and Control*, 9 (1966) 364–379.
- [11] *Future internet 2020: Visions of an industry expert group, May 2009*. DG Information Society and Media, Directorate for Converged Networks and Service.
[http://www.future-internet.eu/fileadmin/documents/reports/
FI_Panel_Report_v3.1_Final.pdf](http://www.future-internet.eu/fileadmin/documents/reports/FI_Panel_Report_v3.1_Final.pdf)
- [12] K. Kántor, Gy. Vaszil, Generalized P colony automata. *Journal of Automata, Languages and Combinatorics*, accepted for publication.
- [13] Kelemen, J., Kelemenová, A., Păun, Gh. (2004). *Preview of P colonies: A biochemically inspired computing model*, Workshop and Tutorial Proceedings. Ninth Inter-

national Conference on the Simulation and Synthesis of Living Systems (Alife IX). (M. Bedau et al., editor), Boston Mass., 2004, pp 82–86.

- [14] Kelemenová, A. (2010). P Colonies. In [16], chapter 23.1, pp 584–593.
- [15] Păun, A. Păun, Gh. The power of communication: P systems with symport/antiport. *New Generation Computing* 20(3) (2002) 295-306.
- [16] Păun, Gh., Rozenberg, G., and Salomaa, A., editors (2010). *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.