Unconventional computing models

Egyetemi doktori (PhD) értekezés

Szerző: Kántor Kristóf

Témavezető: Dr. Vaszil György

Debreceni Egyetem

Természettudományi és Informatikai Doktori Tanács

Informatikai Tudományok Doktori Iskola

Debrecen, 2019

Ezen értekezést a Debreceni Egyetem Természettudományi Doktori Tanács **Informatikai Tudományok** Doktori Iskola **Elméleti számítástudomány, adatvédelem és kriptográfia** programja keretében készítettem a Debreceni Egyetem természettudományi doktori (PhD) fokozatának elnyerése céljából. Nyilatkozom arról, hogy a tézisekben leírt eredmények nem képezik más PhD disszertáció részét, értekezést korábban más intézményben nem nyújtottam be, és azt nem utasították el.

Debrecen, 2019 .....

a jelölt aláírása

Tanúsítom, hogy **Kántor Kristóf** doktorjelölt 2015–2018 között a fent megnevezett Doktori Iskola **Elméleti számítástudomány, adatvédelem és kriptográfia** programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Nyilatkozom továbbá arról, hogy a tézisekben leírt eredmények nem képezik más PhD disszertáció részét, értekezést korábban más intézményben nem nyújtotta be, és azt nem utasították el.
Az értekezés elfogadását javasolom.

Debrecen, 2019 .....

a témavezető aláírása

Unconventional computing models

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében az **Informatika** tudományágban

Írta: **Kántor Kristóf** okleveles **Programtervező Informatikus**

Készült a Debreceni Egyetem **Informatikai Tudományok** doktori iskolája (**Elméleti számtástudomány, adatvédelem és kriptográfia** programja) keretében

Témavezető: Dr. Vaszil György

A doktori szigorlati bizottság:

elnök: Dr. Halász Gábor József .....

tagok: Dr. Porkoláb Zoltán .....

Dr. Horváth Géza .....

A doktori szigorlat időpontja: 2019. március 7

Az értekezés bírálói:

Dr. ....

Dr. ....

A bírálóbizottság:

elnök: Dr. ....

tagok: Dr. ....

Dr. ....

Dr. ....

Dr. ....

Az értekezés védésének időpontja:

.....

Contents

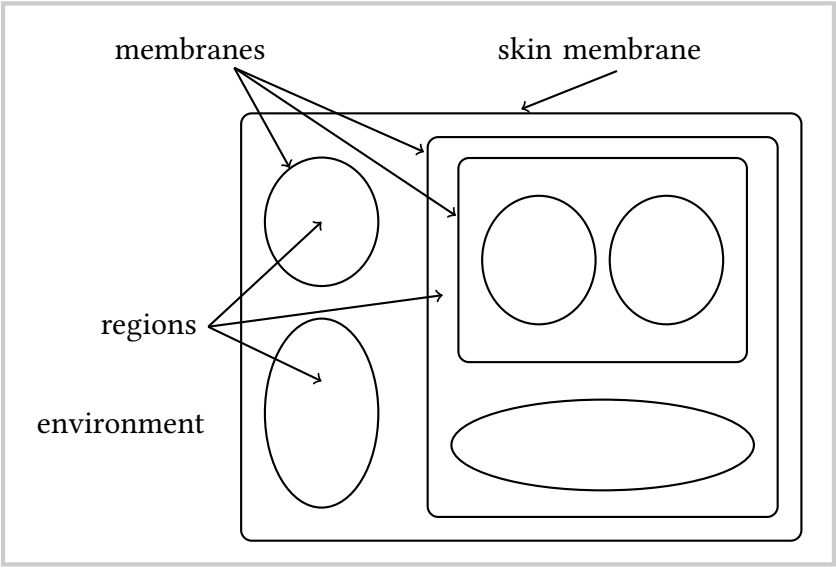
1 Introduction

In the last few decades, unconventional computing became more and more popular as a research area. One of the main pioneers of unconventional computing, natural computing refers to computational processes observed in nature and to nature inspired, human-designed computing. Our understanding of the essence of computation is reinforced when analyzing different, complex natural phenomena in regard of computational processes. The principles and mechanisms underlying natural systems are the main characteristics for nature inspired, human-designed computing. Neural networks, quantum and molecular computing and many more are an integral part of natural computing.

1.1 Membrane computing

Natural computing gave birth to membrane computing. Membrane systems were introduced in [35]. These so-called P systems (named "P"

after their inventor, Gheorghe Păun) are based on an abstraction where the structure and operation of living cells are modelled in the following way: a unique outer membrane contains several cell-like membranes, delimiting different zones, where objects (molecules) can be found. Let us see a graphical representation of a P system using Euler-Venn diagrams:



A molecule is capable of evolving on its own to a different molecule (same way it would happen in nature), dissolving its containing membrane or passing through a membrane, changing its containment zone. These operations are formulated by rules, where evolution is executed in parallel for all objects that are capable of evolving.

All computation models originating from membrane systems are called P systems. P systems are mainly observed in terms of computa-

tional power and complexity, comparing them to other P systems and systems from the field of classical computing. Most of the initial results were connected to computational power, where Turing completeness were observed regarding most classes of P systems. Researchers then sought ways to yield results relating computational complexity and efficiency of various P systems. Other research motivations for P systems are different application areas, such as biology and bio-medicine applications, optimization, software development, economics, networks and computer science in general. Research regarding applications started later than the first P system was introduced, but nowadays it is also a very popular topic. A very important aim of the direction of today's research trend is to establish bridges with other well-known models of computations motivated by computer science, mathematics or biology.

Originally, initial models were based on a hierarchical membrane structure, separating different parts inside a cell, where chemicals (multiset of objects) could evolve due to different regulations. To model the semipermeable property of living membranes, that is, the ability to pass through specific molecules while restricting others, symport (directional) and antiport (bi-directional) types of rules were established. After the initial models were created, researchers turned to models that had non-hierarchical arrangements of membranes. Relating to computer science, the underlying structures turned from trees to arbitrary graphs. A short introduction about the main types of P systems will be presented,

but the interested reader is referred to [36] to obtain a broader understanding of different results in the world of membrane computing.

1.2 P systems with active membranes

To model different biological procedures in cells more precisely, a continuously changing cell structure is beneficial. Active membrane terminology embraces this aspect, that is, during a computation membrane structures might also change in addition to objects. Among the first membrane evolving rules were the membrane dissolution rules, where in addition to an object evolution rule $(original\ object \rightarrow new\ object)$, a special symbol marked the membrane, responsible for the containment of the original object, to be dissolved. After dissolution, the new containment zone was responsible for providing the new set of rules for the new object.

Many membrane evolution rules were defined to model biological operations like exocytosis or endocytosis. A more sophisticated form of rules that include active membranes is the division of membranes. An object evolution rule is decorated with membrane labels in the form of $[\ a\ ]_h \rightarrow [\ b\ ]_h[\ c\ ]_h$, where a, b, c are objects and h is the label of the membrane. By executing this rule, the membrane h is replicated with all of its objects, with object a replaced in the first copy to b and in the second copy to c. Note that the number of regions in the P system can

grow exponentially. The same applies to membrane creation rules, where an elementary membrane is created in addition to the object evolution. Formally, they are the form of $[\ a \rightarrow [\ b\ ]_h\ ]_g$, where in region g, the object a is transformed to an elementary membrane h, containing object b.

1.3 Tissue-like and neural-like P systems

A major difference between cell-like and tissue-like membrane systems is the arrangements of membranes inside them. Tissue-like membrane systems were introduced as the first non-hierarchical models, that are described by graphs. Edges are defined between two nodes (membrane bordered cells), that correspond to a communication channel between them. Communication channels allow direct communication between two membranes by means of symport/antiport rules, for instance. An example rule is shown in the form of $(i, u/v, j)$, where u, v are multisets of objects. In this case, a communication channel is present between membranes i and j, where they exchange objects marked by u and v in an antiport manner. Additionally, if no communication channel is present between cells, they can still use indirect communication by the environment. Models differ mainly based on the structure/functioning of the communication channels, for instance channels are dynamically created and destroyed during computations considering the so-called population

P systems. One can easily introduce cell division into these models as well, to obtain a closer representation of biology.

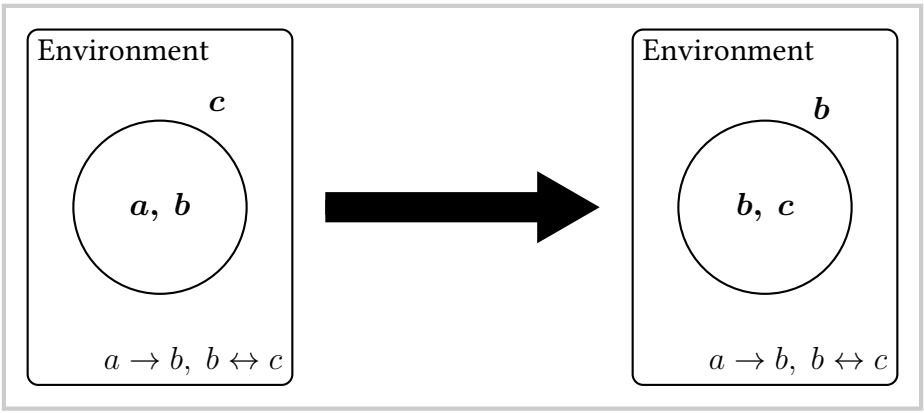
A very similar group of models were defined to model neurological synapses that occur in the brain. Every cell (now called a neuron) is assigned a state that is responsible for the flow of objects that are produced by evolution rules. Rules are in the form of $s\ u \rightarrow s'\ v$, where s and s' are states and $u \rightarrow v$ is a normal multiset rewriting rule. Additionally, v is marked with a target indication, defining a tunnel (synapse) for object movement towards that neuron. Either of the cells is also marked as an output neuron, that sends objects to the environment. Objects then are able to replicate on their way between neurons and the copies are sent to the cells that are connected with the original cell, where the rule was applied.

Spiking neural P systems provide an even more accurate representation of actual synapses. In these models, only one object is considered that is often called the spike. The base of the model is built around time units, to model the refractory period in neurology. A rule describes how many spikes need to be present, how many are consumed and how soon (how many time unit delay is given) the newly produced spike is being sent out to all neurons connected by an outgoing synapse. An output neuron is also used for each rule, that releases its spikes to the environment. This functioning describes a "spike train", that encapsulates different possible computation processes that can be defined for this model.

1.4 P colonies, P colony automata

Many efforts have been made to model information exchange and modification in its simplest forms between cells occurring in nature. The model called P colony represents a perfect example for the term "simplest form", see [27, 28]. P colonies are tissue-like membrane systems modeling cells and their shared environment. The name "colonies" was taken into consideration in correspondence to the field of *colony of grammars* (see [26, 9]), that defined cooperating systems based on a collection of simple generative grammars (generating finite languages each), that as a whole have a considerably increased computing power compared to the power of individual components. P colonies describe both their simple computing agents and their environment by multisets of objects. Colony member cells process these objects by either transforming them internally or by exchanging them with the environment. Considering information processing and storing capability, they are very limited: only a bounded number of objects may reside inside cells simultaneously (this restriction is called the capacity of a P colony), moreover, information cannot be exchanged between cells directly, only through the environment in the form of $a \leftrightarrow b$, or it is possible that information evolves inside a cell in the form of $a \rightarrow b$. This form of behaviour is grouped together to a set, that we call a program, which consists of k rules, where k is the capacity of the system. Arbitrarily many programs describe the

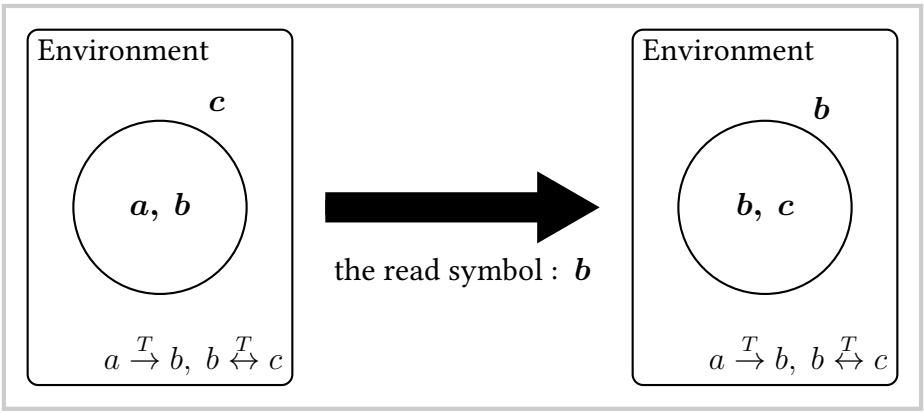
exact behaviour of a colony member cell. A visual example is given below:



When considering a discrete time-scale, P colonies apply their rules in a maximal parallel way at every time-step until they reach one of their final configurations. Sometimes the final configuration is defined by halting, that is, when no programs can be applied by any of the cells. As the result of a computation, it is natural to count the number of different objects that are present in the environment at that final configuration, thus P colonies describe computations over the sets of numbers. Although extremely simple, computational completeness of P colonies has been shown given very restricted capacity parameters and/or syntactic or functioning restrictions, see [7, 8, 11, 12, 5, 6, 20, 19].

In [4], P colony automata have been born as a result of the aim to describe sets of strings instead of numbers. By assuming the presence of an input tape with an input string in the environment, thus adding the

tape version of existing types of rules in the form of $a \xrightarrow{T} b$ or $a \xleftrightarrow{T} b$, it was possible to accept strings and string languages with the model. To visualize the acceptance of strings, an example is given below:

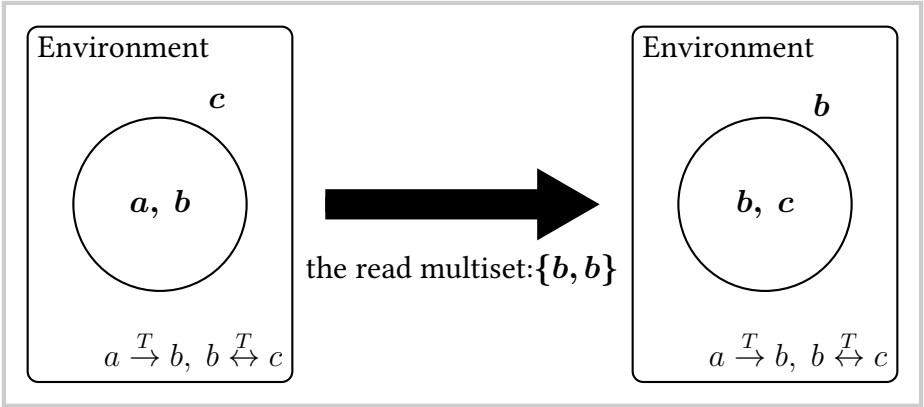


The new types of rules enable P colony automata to have a slightly extended behaviour: in addition to evolving an object inside a cell or exchanging it with the environment, the processed object must be read from the prefix of the input string. The number of objects that can be read at every time-step is limited to one, so in order to accept a string of length n, a P colony automaton must work for at least n discrete time-steps. Unfortunately, rule application gets deceptively simple considering the possibility of applying several tape rules simultaneously, that could be the cradle of possible conflicts between tape rules which would need to read different symbols from the same input at the same time. Nevertheless, several variants of P colony automata turned out to be computationally complete, as shown in [4] and later in [2].

To address the tape rule matter, a new model was introduced in [21] called generalized P colony automata or genPCol automata in short. The definitions have since been extended in [22]. The main idea was to create a model that processes strings more naturally by associating them to the computations by keeping track of the communication with the environment instead of just reading input tapes. The model was inspired by antiport P systems and P automata in particular, introduced in [15] (see also [14]).

P automata are symport/antiport P systems ([33]) accepting string languages using a different approach than "ordinary" P colony automata. Instead of having input tapes and input strings, they keep track of their communications with the environment. Computing agents communicate freely with their environment without forcing themselves to a certain behaviour through a given input, where each object that can be requested for input by the communication rules are assumed to be available in an unbounded supply. During computations, sequences of multisets can be determined that enter the system, thus characterizing the strings that are read by the P automata.

This string processing behaviour is employed in genPCol automata, as illustrated in the following figure.



The computations describe accepted multiset sequences that are turned into accepted strings using a mapping function, that assigns strings to multisets over some alphabet Σ. Both non-tape and tape rules can be given to descibe the operation of genPCol automata as in "ordinary" P colony automata, where tape rules imply the reading of the processed symbols over the course of a computation, however an automaton is allowed to read more than one symbol in a single computational step. The corollary to this is the complete nullification of conflicts between tape rules during computation since it is now possible to apply arbitrarily many tape rules in one time-step.

1.5 Overview

This thesis is focused on genPCol automata, which is a P system, that combines the characteristics of P automata and P colonies. The structure

of the dissertation is the following. In chapter 2, some basic notions will be defined regarding multisets and classical computational models that will be used in proofs. Chapter 3 defines the model of computation formally. Moving on, chapter 4 is divided into different sections, that are based on different multiset to string mapping functions that act as classifiers for the genPCol automata. It is studied in section 4.1 how permutation mapping defines the computational power for different sub-types of genPCol au tomata. A thorough examination is performed considering the capacity of the system as well. Section 4.2 consists of results considering finite transducers that are responsible for the actual multiset mapping. The last subsection of this chapter compares genPCol automata to P automata. The last chapter introduces an application of genPCol automata by constructing them in a way that can be used in deterministic parsing. Finally, in chapter 6 and chapter 7, two summaries are given, one in English and one in Hungarian, respectively.

2 Preliminaries

In this section we introduce basic definitions regarding formal languages, whilst introducing the notations that will be used throughout the dissertation.

In formal language theory, an alphabet is a non-empty finite set, its contents are called objects or symbols or letters. Alphabets will be denoted with the symbols V or Σ. A word over a finite alphabet V is a finite sequence of letters, that is $w = w_1 w_2 \dots w_n$, where $w_i \in V$, $1 \leq i \leq n$ and $n \in \mathbb{N}$. 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 $|w|_a$, and the length (the number of symbols) of a word w by $|w|$. By concatenation one can combine two words to form a new word, whose length is the sum of the lengths of the original words, that is, if $w = w_1 \cdot w_2 \cdot \ldots \cdot w_n$ (or $w = w_1 w_2 \dots w_n$ in short) with $w \in V^*$, then $|w| = \sum_{i=1}^{n} |w_i|$.

We are going to use different classes of languages. We will denote

them with $\mathcal{L}(X)$, where X is either a class of grammars in the conventional formal language theory sense or the unconventional models used in the dissertation.

Now we define the notions related to multisets. If the set of non-negative integers is denoted by $\mathbb{N}$, then a multiset over a set V is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, that is, $supp(M) = \emptyset$. The set of finite multisets over the alphabet V is denoted by $\mathcal{M}(V)$. 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, $card(M)$ is defined as $card(M) = \Sigma_{a \in M} M(a)$. For two multisets $M_1, M_2 : V \to \mathbb{N}$, $M_1 \subseteq M_2$ 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 $(M_1 \cup M_2) : V \to \mathbb{N}$ 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 $(M_1 - M_2) : V \to \mathbb{N}$ with $(M_1 - M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$.

Now we introduce the different computational models that will be used in proofs.

2.1 Register machines

Formally, a *r*egister machine is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and R is the set of instructions; each label from H labels only one instruction from R. There are several types of instructions which can be used. For $l_i, l_j, l_k \in H$ and $r \in \{1, \ldots, m\}$ we have

- l_i : (ADD$(r), l_j$) - *add*: Add 1 to register r and then go to the instruction with label l_j.
- l_i : (CHECKSUB$(r), l_j, l_k$) - *zero check and subtract*: If the value of register r is not zero, subtract one from it and go to instruction l_j, otherwise leave it unchanged and go to l_k.
- l_h : HALT - *halt*: Stop the machine.

A register machine accepts a number n if starting the computation with the instruction labeled by l_0 while having n in the first register (and all other registers empty), it reaches the halting instruction. This way a register machine computes a set of numbers.

To be able to accept strings, we might also add an input tape to a register machine, together with a new type of instruction

- l_i : (READ$(a), l_j$) for a symbol $a \in \Sigma$ of some input alphabet Σ - *read*: Read symbol a from the input tape and go to the instruction

with label l_j.

Such an instruction can be applied if the reading head scans a symbol $a \in \Sigma$ on the input tape, and the head moves to the next tape cell after the application of the instruction.

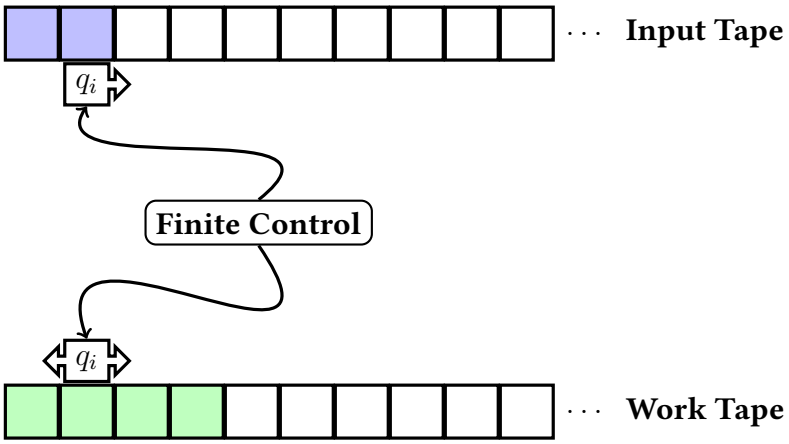
It is not difficult to see that register machines with input tape characterize the class of recursively enumerable languages, as they can simulate two-counter machines. Two-counter machines (see [17]) are Turing machines with an input tape which is read only from left to right in one direction, and work-tapes which are only used as counters (by moving the reading heads left or right without writing anything to the tape cells).

2.2 Turing machines with restricted logarithmic space

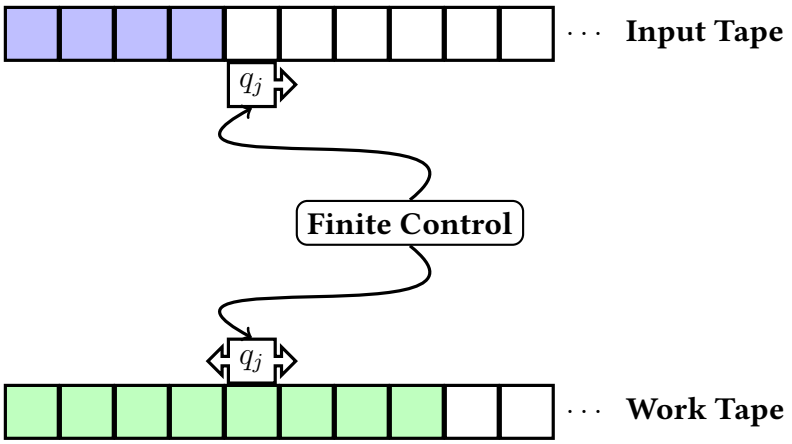
A Turing machine is $S(n)$ space bounded if it can use a maximum of $S(n)$ space on its work-tapes, where n is the length of the input word, and $S : \mathbb{N} \to \mathbb{N}$ is a function on natural numbers. Let us consider space bounded Turing machines with a unique property: the length of workspace they can use in any given step of a computation depends on the number of input symbols that were already consumed until that point (instead of the length of the whole input). We call these Turing machines *restricted* space bounded. Moreover, we call a Turing machine one-way,

if it can only read the input word from left to right, one time only.

For the sake of understandability, a visual representation is given below. This Turing machine is restricted $S(n)$ space bounded for $S(n) = 2n$, the already read cells are marked with blue on the input tape, and the space available on the work-tape in the current state is marked with green.



As we can see, the first two symbols from the input word have been read and this allows the automaton to work on four cells. Now let us see what happens after two more symbols are read.



When reading an additional symbol, the space we can use grows by two additional cells on the work tape, thus, after reading four cells, the available workspace grows to eight.

Formally, a non-deterministic Turing machine with one-way input tape is a 7-tuple $M = (m, V, \Gamma, Q, \delta_M, q_0, F)$, where

- m is the number of work-tapes,
- V is the input alphabet,
- Γ is the work-tape alphabet,
- Q is the set of internal states with q_0 being the initial state,
- $F \subseteq Q$ is the set of accepting states, and
- $\delta_M : V \cup \{\varepsilon\} \times Q \times \Gamma^m \to 2^{Q \times \Gamma^m \times \{left, right, none\}^m}$ is the transition relation of M.

Let us denote by NSPACE($S(n)$) the class of languages accepted by non-deterministic Turing machines using a workspace which is bounded by a function $S : \mathbb{N} \to \mathbb{N}$ of the length of the input. We say that $L \in$ r1NSPACE($S(n)$) if there is a Turing machine which accepts L by reading the input from a read only input tape once from left to right, and for every accepted word of length n, there is an accepting computation during which the number of non-empty cells on the work-tape(s) is bounded in each step by $c \cdot S(d)$ where c is an integer constant, and $d \leq n$ is the number of input tape cells that have already been read, that is, the actual distance of the reading head from the left end of the one-way input tape. If $S(n) = \log(n)$, we denote the class r1NSPACE($S(n)$) by r1LOGSPACE.

The power of restricted logarithmic space bounded Turing machines is an interesting question which we will not investigate further, but we would like to mention the following. In [13] it was 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 work-tapes. In [30], deterministic variants of one-way Turing machines with sublinear space bounds were examined. It was shown that the classes of languages accepted by restricted and strongly logarithmic space bounded one-way deterministic Turing machines coincide. (A Turing machine is said to be strongly space bounded if the workspace is bounded not only for the accepting, but for any possible computations.)

3 Definitions and examples

In this chapter we focus on giving the basic definitions regarding genPCol automata, as seen in [21, 22]. In the following chapters, the results will be structured around the capacity of the system and the semantic constraints that are placed upon programs. We consider three possible subtypes of genPCol automata regarding the restrictions placed on their programs: (1) the unrestricted case, (2) the case when all programs must contain at least one tape rule (all-tape programs), and (3) the case when all communication rules are tape rules (com-tape programs).

3.1 Generalized P colony automata

Now let us define genPCol automata formally.

Definition 3.1. *A genPCol automaton of capacity* k *and with* n *cells,* $k, n \geq 1$, *is a construct* $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ *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, that are always present in the environment in an infinite number;*
- $w_E \in (V - \{e\})^*$ *is a string representing the multiset of objects different from* e *which is found in the environment initially;*
- $(w_i, P_i), 1 \leq i \leq n$*, specifies the* i*-th* cell *where* $w_i \in \mathcal{M}(V)$ *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. The set* P_i *is a set of* programs*, each program is formed from* k *rules of the following types (where* $a, b, c \in V$*):*
  - tape rules *of the form* $a \overset{T}{\rightarrow} b$*, or* $a \overset{T}{\leftrightarrow} b$*, 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.*

  A program is called a tape program *if it contains at least one tape rule.*
- F *is the set of* accepting configurations *of the automaton which we will specify in more detail below.*

An $(n+1)$-tuple $(u_E, u_1, \ldots, u_n)$ describes a *configuration* of a gen-

PCol automaton, where $u_E \in (V - \{e\})^*$ represents the multiset of objects different from e in the environment, and $u_i \in V^*, 1 \leq i \leq n$, represent the contents of the i-th cell. The initial contents of the computing agents and the environment is given by the *initial configuration*: $(w_E, w_1, \ldots, w_n)$. The set of *accepting configurations* F is a set of configurations which is given as a set of $n + 1$-tuples of the form $(v_E, v_1, \ldots, v_n)$, as above.

During a computation, a multiset sequence is read by the genPCol automaton. An element of the sequence (possibly consisting of more than one symbol) is read during each configuration change: it corresponds exactly to the multiset of symbols introduced by the tape rules of the applied program of the system (the symbols that are on the righthand sides of rewriting tape rules, and those that are on the lefthand sides of communication tape rules).

Formally speaking, the computation procedure is as follows: for any rule r we define the following multisets. Based on rules that are in the following form: $r = a \stackrel{T}{\rightarrow} b$, $r = a \stackrel{T}{\leftrightarrow} b$, $r = a \leftrightarrow b$ or $r = a \rightarrow b$, let $left(r) = a$, $right(r) = b$. Extending this notation also for programs, we get $\alpha \in \{left, right\}$ 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 $p = \langle r_1, \ldots, r_k \rangle$, the notation $r \in p$ denotes the fact that $r = r_j$ for some $j,\ 1 \leq j \leq k$. So for a program p, $left(p)$ and $right(p)$ are the collection (multiset) of symbols on the left or right sides of their

rules. Moreover, for any tape program p, that contain rules of the form $a \xrightarrow{T} b$ or $a \overset{T}{\leftrightarrow} b$ we also define $read(p)$ as the multiset of symbols that are read by the automaton as the result of the applications of the tape rules of p, more precisely

$$read(p) = \bigcup_{r \in p, r = a \xrightarrow{T} b, b \neq e} right(r) \cup \bigcup_{r \in p, r = a \overset{T}{\leftrightarrow} b, a \neq e} left(r).$$

Thus, $read(p)$ is the multiset (collection) of symbols (different from e) on the right side of rewriting tape rules or the left side of communication tape rules.

We also define the multisets $export(p)$, $import(p)$ and $create(p)$, the multisets that indicate the objects that are sent out to the environment and brought inside the cell and the multiset of symbols produced by the rewriting rules of program p. Formally,

$$export(p) = \bigcup_{r \in p, r = a \overset{T}{\leftrightarrow} b, a \neq e} left(r) \cup \bigcup_{r \in p, r = a \leftrightarrow b, a \neq e} left(r),$$

$$import(p) = \bigcup_{r \in p, r = a \overset{T}{\leftrightarrow} b} right(r) \cup \bigcup_{r \in p, r = a \leftrightarrow b} right(r), \text{ and}$$

$$create(p) = \bigcup_{r \in p, r = a \xrightarrow{T} b} right(r) \cup \bigcup_{r \in p, r = a \rightarrow b} right(r).$$

Let the programs of each P_i be labeled in a one-to-one manner by labels from the set $lab(P_i)$, $lab(P_i) \cap lab(P_j) = \emptyset$ for $i \neq j$, $1 \leq i, j \leq n$. 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 $p \in lab(P_i)$, we also write $p \in P_i$.

Let $c = (u_E, u_1, \ldots, u_n)$ be a configuration of a genPCol automaton Π and let $U_E = u_E \cup \{e, e, \ldots\}$ be the multiset of objects found in the environment together with the infinite number of es which are always present. A *sequence of programs* P_c is *applicable in configuration* c, if the following conditions hold:

1. At most one program is selected for each cell, that is, if $p, p' \in P_c, p \neq p'$, $p \in P_i$, and $p' \in P_j$, then $i \neq j$;
2. 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 $left(p) = u_i$;
3. the symbols which are brought inside the cells by the programs are present in the environment, that is, $\bigcup_{p \in P_c} import(p) \subseteq U_E$;
4. 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 $c = (u_E, u_1, \ldots, u_n)$ is *changed* to a configuration $c' = (u'_E, u'_1, \ldots, u'_n)$, denoted by $c \Rightarrow c'$, by applying the set P_c of applicable programs if the following properties hold:

1. If there is a $p \in P_c$ such that $p \in P_i$, then $u'_i = create(p) \cup import(p)$, otherwise $u'_i = u_i$, $1 \leq i \leq n$; and

2. $U'_E = U_E - \bigcup_{p \in P_c} import(p) \cup \bigcup_{p \in P_c} export(p)$ (where U'_E again denotes $u'_E \cup \{e, e, \ldots\}$ with an infinite number of es).

We denote the reflexive and transitive closure of $\Rightarrow$ by $\Rightarrow^*$.

The set of all applicable sequences of programs in the configuration $c = (u_E, u_1, \ldots, u_n)$ is denoted by App_c, that is,

$$App_c = \{P_c = (p_1, \ldots, p_n) \in (P_1 \cup \{\varepsilon\}) \times \ldots \times (P_n \cup \{\varepsilon\}) \mid \text{where } P_c \text{ is a sequence of applicable programs in configuration } c\}.$$

A configuration c is called *a halting configuration* if the set of applicable sequences of programs is the singleton set $App_c = \{(p_1, \ldots, p_n) \mid p_i = \varepsilon \text{ for all } 1 \leq i \leq n\}$.

Contrary to the different computational modes used in P colony automata as defined in [4], the above definitions describe a system where programs are applied in a maximal parallel way during a computation, that is, in each computational step, every component cell must nondeterministically choose and apply one of its applicable programs. Afterwards at the end of the operation, we look at the rules which were tape rules (in the applied set of programs) and collect all the symbols that they "read": this multiset of collected symbols is the multiset read by the system in the given computational step.

Remark 3.1. Although the set of configurations of a genPCol automaton Π is infinite (because the cardinality of the multiset corresponding to the

contents of the environment is not bounded, this multiset might even be infinite), the set of possible input multisets is finite. To see this, note that the applicability of a program by a component cell depends on the contents of the particular component. Since at most one program can be applied in a component in one computational step, and the number of programs associated to each component is finite, the number of different sets of applicable programs in any configuration is also finite.

Now let us formally define a successful computation, where a sequence of accepted multisets enter the system during the steps of the computation:

Definition 3.2. *Let* $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ *be a genPCol automaton. The set of input sequences accepted by* Π *is defined as*

$$\begin{aligned} A(\Pi) = \{u_1 u_2 \ldots u_s \mid\ & u_i \in (V - \{e\})^*, 1 \leq i \leq s, \text{ and there is a} \\ & \text{configuration sequence } c_0, \ldots, c_s, \text{ with } c_0 = (w_E, w_1, \ldots, w_n), \\ & c_s \in F, \text{ and } c_i \Rightarrow c_{i+1} \text{ with } \bigcup_{p \in P_{c_i}} read(p) = u_{i+1} \\ & \text{for all } 0 \leq i \leq s-1\}. \end{aligned}$$

As we have mentioned previously, in order for genPCol automata to accept string languages, a special mapping function is given. Let us define them formally for genPCol automata:

Definition 3.3. *Let* Π *be a genPCol automaton, let* Σ *be an alphabet and let* $f : (V - \{e\})^* \to 2^{\Sigma^*}$ *be a mapping, that maps a muliset to a set of*

strings over Σ^**, moreover,* $f(u) = \varepsilon$ *if and only if* u *is the empty multiset. The* language accepted by Π *with respect to* f *is defined as*

$$L(\Pi, f) = \{f(u_1) \cdot f(u_2) \cdot \ldots \cdot f(u_s) \in \Sigma^* \mid u_1 u_2 \ldots u_s \in A(\Pi)\}.$$

As mentioned previously, the restrictions placed on the programs of a genPCol automaton greatly influence their computational power. Let us use the following notions for different subtypes of genPCol automata:

1. $\mathcal{L}_{\mathcal{F}}(\text{genPCol}, \text{com-tape}(k))$ is the class of languages accepted by genPCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the communication rules are tape rules,

2. $\mathcal{L}_{\mathcal{F}}(\text{genPCol}, \text{all-tape}(k))$ is the class of languages accepted by genPCol automata with capacity k and with mappings from the class $\mathcal{F}$ where all the programs must have at least one tape rule,

3. $\mathcal{L}_{\mathcal{F}}(\text{genPCol}, *(k))$ is the class of languages accepted by genPCol automata with capacity k and with mappings from the class $\mathcal{F}$ where programs with any kinds of rules are allowed.

For the sake of easier readability, we will name the subtypes as com-tape, all-tape and unrestricted or $*$ case respectively.

In the dissertation we will focus on two special classes of mappings $\mathcal{F}$. The first one is called the permutation mapping. Formally, $f_{perm} : (V - \{e\})^* \to 2^{\Sigma^*}$, where $\Sigma = (V - \{e\})^*$, and

$$f_{perm}(x) = \{y \in (V - \{e\})^* \mid y \in perm(x)\},$$

where $perm(x) \subseteq V^*$ denotes the set of strings representing the multisets composed of the symbols of x, where each multiset is a permutation of x. Let us denote the class of languages accepted by genPCol automata using f_{perm} mapping the following way:

$$\mathcal{L}_{perm}(\text{genPCol}, X(k)), \text{ where } X \in \{\text{com-tape}, \text{all-tape}, *\}.$$

Example 3.1 If $V = \Sigma = \{a, b\}$ and $u \in \mathcal{M}(V)$ is the multiset represented by the string $aabb$ then $f_{perm}(u) = \{aabb, abab, abba, baab, baba, bbaa\} \subseteq \Sigma^*$.

The second special mapping is defined as follows: Let V and Σ be two alphabets, and let $\mathcal{M}_{FIN}(V) \subseteq \mathcal{M}(V)$ denote the set of finite subsets of the set of finite multisets over an alphabet V. Consider a mapping $f : D \to 2^{\Sigma^*}$ for some $D \in \mathcal{M}_{FIN}(V)$. We say that $f \in \mathcal{F}_{\text{TRANS}}$, if for any $v \in D$, we have $card(f(v)) = 1$, and we can obtain $f(v) = \{w\}, w \in \Sigma^*$ by applying a deterministic finite transducer to any string representation of the multiset v, (as w is unique, the transducer must be constructed in such a way that all string representations of the multiset v as input result in the same $w \in \Sigma^*$ as output, and moreover, as f should be nonerasing, the transducer produces a result with $w \neq \varepsilon$ for any nonempty input). Let us denote the class of languages accepted by genPCol automata using $f \in \mathcal{F}_{\text{TRANS}}$ mapping the following way:

$$\mathcal{L}_{\text{TRANS}}(\text{genPCol}, X(k)), \text{ where } X \in \{\text{com-tape}, \text{all-tape}, *\}.$$

Example 3.2 Consider $V = \{a, b, c\}$ and let $D = \{u_1, u_2, u_3\} \in \mathcal{M}_{FIN}(V)$ where $u_i,\ 1 \leq i \leq 3$, are finite multisets over V, given by their string representations as $u_1 = a$, $u_2 = b$, and $u_3 = c$.

If $f_1 : D \to 2^{\Sigma^*}$ with $\Sigma = \{d, e, f, g, h, i\}$ such that $f_1(a) = \{de\}$, $f_1(b) = \{fg\}$, $f_1(c) = \{hi\}$, then it is easy to see that $f_1 \in \mathcal{F}_{TRANS}$ by constructing the finite transducer which reads strings over V and outputs the symbols $f_1(x)$ after consuming each $x \in V$ from the input string. ∎

For the sake of understandability, we give two examples of genPCol automata. In the first example, we demonstrate that all regular languages can be accepted with a system of capacity two.

Example 3.3 Let $L \subseteq V^*$ be a regular language, and let $M = (V, Q, \delta, q_0, F)$ be a finite automaton with $L(M) = L$, with alphabet V, set of states Q, initial state q_0, set of final states F, and transition function $\delta : V \times Q \to Q$.

Now let $\Pi = (V \cup \{q_I\} \cup Q, e, w_E, (w, P), F')$ a genPCol automaton of capacity two, where $q_I \notin V \cup Q$, with initial environment $w_E = \bigcup_{a \in V} \{a, a\}$, initial cell contents $w = eq_I$, set of rules

$$P = \{\langle e \overset{T}{\leftrightarrow} a,\ q_I \to q_0 \rangle \mid a \in V\} \cup \{\langle x \overset{T}{\leftrightarrow} a,\ q \to q' \rangle \mid \text{for all } x \in V \text{ such that } \delta(x, q) = q' \text{ for some } q, q' \in Q\},$$

set of final configurations $F' = \{(w_E - \{x\}, xq_f) \mid \text{for all } x \in$

$V,$ and $q_f \in F\}$, and mapping function $f(a) = a$, for every $a \in V - \{e\}$.

Π works the following way: It randomly chooses a symbol from the environment to swap it with e and transforms it's other symbol q_I to q_0, that represents the initial state of M. Then, it starts applying programs that simulate the transition function δ of M. We are able to exchange the first object inside the cell to any $a \in V$, because initially there were two copies of each in the environment and we only exchange these objects with the cell, without evolving them. The computation ends, when we have $q_f \in F$ as the second object in the cell. At this point, the environment has only one copy of the last read symbol, the other one is inside the cell. It is not hard to see that Π simulates the computations of M, that is, $L(\Pi, f) = L(M)$.

Note that $f \in \mathcal{F}_{TRANS}$, and note also that $L(\Pi, f) = L(\Pi, f_{perm})$ (since Π can read at most one symbol in any of its computational steps).

■

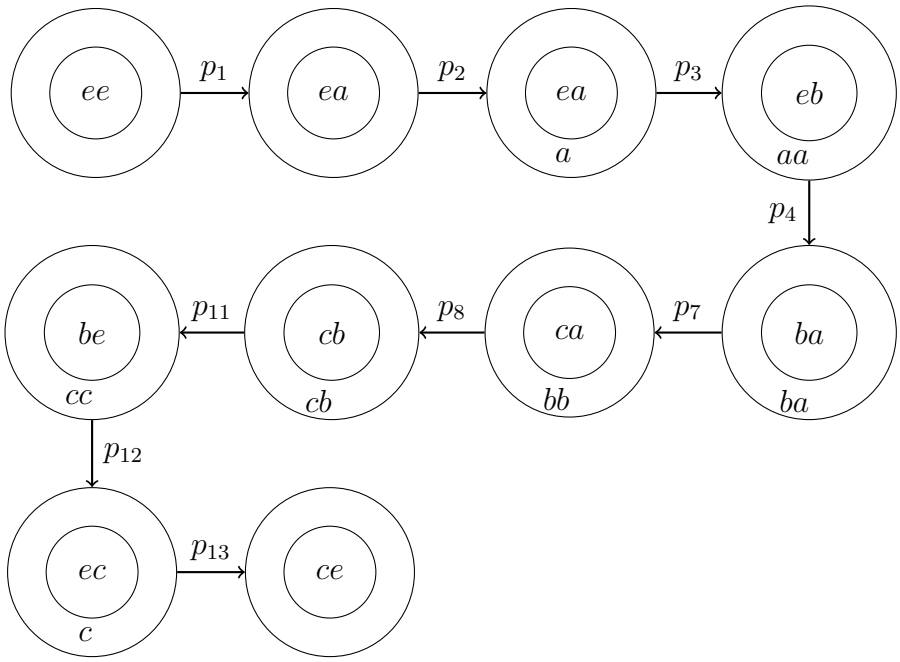
The next example demonstrates how non-context-free languages can be accepted by genPCol automata.

Example 3.4 Let $\Pi = (\{a, b, c\}, e, \emptyset, (ee, P), F)$ be a genPCol automa-

ton where

$$P = \Big\{ p_1 : \langle e \to a, e \overset{T}{\leftrightarrow} e \rangle, p_2 : \langle e \to a, a \overset{T}{\leftrightarrow} e \rangle, p_3 : \langle e \to b, a \overset{T}{\leftrightarrow} e \rangle,$$
$$p_4 : \langle e \to b, b \overset{T}{\leftrightarrow} a \rangle, p_5 : \langle e \to c, b \overset{T}{\leftrightarrow} a \rangle, p_6 : \langle a \to b, b \overset{T}{\leftrightarrow} a \rangle,$$
$$p_7 : \langle a \to c, b \overset{T}{\leftrightarrow} a \rangle, p_8 : \langle a \to c, c \overset{T}{\leftrightarrow} b \rangle, p_9 : \langle a \to e, c \overset{T}{\leftrightarrow} b \rangle,$$
$$p_{10} : \langle b \to c, c \overset{T}{\leftrightarrow} b \rangle, p_{11} : \langle b \to e, c \overset{T}{\leftrightarrow} b \rangle, p_{12} : \langle b \to e, e \overset{T}{\leftrightarrow} c \rangle,$$
$$p_{13} : \langle c \to e, e \overset{T}{\leftrightarrow} c \rangle \Big\},$$

with all the communication rules being tape rules. Let also $F = \{(\emptyset, ec)\}$ be the final configuration. At first let us consider the mapping f_{perm}. We give a visual example of a possible computation of this system:



The symbols in the inner circle represent the contents of the cell and the symbols in the outer circle represent the contents of the environment different from e. The labels on the arrows represent the chosen rule from P. The computation accepts the string $a^2b^2c^2$. In consequence to the chosen mapping function and the fact that maximum one symbol is read during one computational step, it is not difficult to see that similarly to the one above, the computations which end in the final configuration accept a word of the language $L(\Pi, f_{perm}) = \{a^n b^n c^n \mid n \geq 1\}$.

On the other hand, if we consider the mapping $f_1 \in \mathcal{F}_{\text{TRANS}}$ from Example 3.2, that is, $f_1 : \{a, b, c\} \to 2^{\Sigma^*}$ with $\Sigma = \{d, e, f, g, h, i\}$ and $f_1(a) = \{de\}$, $f_1(b) = \{fg\}$, $f_1(c) = \{hi\}$, we get the language $L(\Pi, f_1) = \{(de)^n (fg)^n (hi)^n \mid n \geq 1\}$. ∎

4 On the power of generalized P colony automata

In this chapter we present the results regarding genPCol automata. First of all, we are going to look at the case of when the permutation mapping is used to transform accepted multiset sequences into accepted strings. Moving on, we are going to take a look at the more generalized approach to mapping functions by using finite transducers.

Before moving on to the details, consider following, which are immediate consequences of the definitions.

Proposition 4.1. *For any class of mappings $\mathcal{F}$, we have*

1. $\mathcal{L}_{\mathcal{F}}(\textit{genPCol}, \textit{com-tape}(k)) \subseteq \mathcal{L}_{\mathcal{F}}(\textit{genPCol}, *(k))$ *for* $k \geq 1$;
2. $\mathcal{L}_{\mathcal{F}}(\textit{genPCol}, \textit{all-tape}(k)) \subseteq \mathcal{L}_{\mathcal{F}}(\textit{genPCol}, *(k)$ *for* $k \geq 1$;
3. $\mathcal{L}_{\mathcal{F}}(\textit{genPCol}, X(k)) \subseteq \mathcal{L}_{\mathcal{F}}(\textit{genPCol}, X(k+1))$ *for* $k \geq 1$ *and* $X \in \{\textit{com-tape}, \textit{all-tape}, *\}$;

4. $(\mathcal{L}_{\mathcal{F}}(\textit{genPCol}, \textit{com-tape}(k)) \cap \mathcal{L}_{\mathcal{F}}(\textit{genPCol}, \textit{all-tape}(k))) \setminus \mathcal{L}(\textit{CF}) \neq \emptyset$ *for* $k \geq 2$*; and*

5. $(\mathcal{L}_{\mathcal{F}}(\textit{genPCol}, \textit{com-tape}(k)) \cap \mathcal{L}_{\mathcal{F}}(\textit{genPCol}, \textit{all-tape}(k))) \supset \mathcal{L}(\textit{REG})$ *for* $k \geq 2$.

Proof. The first two inclusions hold, as com-tape and all-tape systems are special cases of the unrestricted variant. If we consider adding the object e to the initial cell contents, and a rule $e \to e$ to the programs of all cells in a system, it is easy to see that it does not change the accepted language of the genPCol automaton, therefore proving the third statement. In Example 3.4, it was shown, that $\{a^n b^n c^n | n \geq 1\} \notin \mathcal{L}(\text{CF})$ is accepted by genPCol automata with capacity two, where in addition to all programs having at least one tape rule, all the communication rules are tape rules. Thus, the fourth statement holds for capacity $k = 2$. If the third statement is considered as well, the fourth statement holds for any $k \geq 2$. Considering the fourth inclusion and Example 3.3, we can see that the last statement also holds. ■

Now, let us examine the power of genPCol automata in more details. It was shown in [21] that in the unrestricted case genPCol automata accept the class of recursively enumerable languages with capacity two. However, in [22] a new result proved that systems even with capacity one are able to achieve this computational power. Additionally, genPCol

automata with restricted programs have different computational power: all-tape systems are able to characterize the class of recursively enumerable languages but only if they are at least capacity two, while the power of com-tape systems is bounded by the power of so called restricted logarithmic space Turing machines, a Turing machine model considered in [13] and later also in [16] for describing the computational power of certain variants of P automata. Lastly, genPCol automata will be compared to ordinary P automata. The results of this chapter are based mainly on [21] and [22].

4.1 Basic multiset to string mappings: permutations

4.1.1 Unrestricted programs: characterizing the class of recursively enumerable languages

First we consider genPCol automata, where there are no semantical/syntactical restrictions for the rules. In [21], it was shown that unrestricted genPCol automata with a permutation mapping function and a capacity of two are able to characterize recursively enumerable string languages by simulating *two-counter machines* [17]. Later in [22], it was shown that if we restrict the capacity of the system even more, the char-

acterization of recursively enumerable string languages can still be acquired. The idea came from P colonies, where all recursively enumerable sets of integers can be characterized by systems of capacity one (see [8]). For the sake of brevity, only the more restricted theorem will be presented.

In the proof we will use the notion of a *register machine*, and we also consider the variant of a *register machine with input tape*, defined in section 2.1. Recall that such a register machine consists of a given number of registers each of which can hold an arbitrarily large non-negative integer number (we say that the register is empty if it holds the value zero), and a set of labeled instructions which specify how the numbers stored in registers can be manipulated (see also [32] for more information).

The following theorem is from [22].

Theorem 4.1.

$$\mathcal{L}_{perm}(genPCol, *(1)) = \mathcal{L}(RE).$$

Proof. In the following we show that genPCol automata can simulate register machines with input tape, and thus, characterize the class of recursively enumerable languages. The idea of the simulation is to have an object in the environment corresponding to the label of the instruction which is to be simulated next. The cells of the system "process" the instruction label in such a way that the necessary modifications of the configuration are implemented, and the label of the next instruction is

sent to the environment.

Let us consider an m-register machine $M = (m, H, l_1, l_h, R)$ with input tape. The contents of the register r will be represented by the number of copies of a specific object a_r in the environment. The set of instruction labels is H and the labels themselves are denoted by $l_i,\ 1 \leq i \leq |H|$, with the initial and the halting instructions being labeled as l_1 and l_h, respectively, $h = |H|$.

Let us now construct a genPCol automaton

$$\Pi = (V, e, w_e, (e, P_1), \ldots, (e, P_6), (e, P_{l_1}), \ldots, (e, P_{l_h}), F),$$

where $l_i,\ 1 \leq i \leq h$ are labels of reading instructions of the form l_i : (READ$(a), l)$, and let

$$\begin{aligned} V = &\left\{e, l_i, l_i', l_i'', \bar{l}_i, K_i, L_i, L_i', L_i'', L_i''', E_i, F_i, \$_i \ \mid \text{ for each } l_i \in H\right\} \cup \\ &\{a_i, a_{i,j} \ \mid \ 1 \leq i \leq m,\ 1 \leq j \leq |H|\} \cup \{D, D', T\}\,, \text{ and} \\ F = &\{(u, v_1, \ldots, v_{n+6}) \ \mid \ l_h \in u,\ t \notin v_i, \text{ and } u, v_i \in V^*, \\ &\quad 1 \leq i \leq n+6\}\,. \end{aligned}$$

Because initially there are only copies of e in the environment and inside the cells, we have to initialize the simulation of the computation of M by generating the initial instruction label l_1, and an arbitrary number of l_i', l_i'' for all $l_i \in H$ which will be helpful when implementing the simulation of the register machine. These symbols are generated by the

first two components with the following programs. (Note that the gen-PCol automaton Π is non-deterministic, if not enough objects are produced by this initialization phase, then the simulation will fail, but on the other hand, there will always be computations where enough objects are produced.)

$$\begin{aligned} P_1 \;&\supset\; \{\langle e \to l_i' \rangle, \langle l_i' \leftrightarrow e \rangle, \langle e \to l_i'' \rangle, \langle l_i'' \leftrightarrow e \rangle \mid l_i \in H\} \cup \\ &\qquad \{\langle e \leftrightarrow D' \rangle, \langle D' \to l_1 \rangle, \langle l_1 \leftrightarrow D \rangle\}\,, \\ P_2 \;&=\; \{\langle e \to D' \rangle, \langle D' \to D' \rangle, \langle D' \leftrightarrow l_1' \rangle, \langle l_1' \to D \rangle, \langle D \leftrightarrow l_1'' \rangle\}\,. \end{aligned}$$

From the initial configuration, with the use of these programs, we obtain a configuration where the environment contains the label of the initial instruction, l_1, and a multiset of primed and double primed instruction labels.

To simulate an instruction $l_i : (\text{ADD}(r), l_j)$, the object l_i should be consumed from the environment, then a copy of the object a_r and object l_j should be added to the environment. This is done by the cooperation of the first and the third components using the following programs. (These programs should be added to P_1 and P_3 for each instruction label

corresponding to the "add" instructions, as above.)

$$
\begin{aligned}
P_1 \;\supset\; & \{\langle D \leftrightarrow a_{r,i}\rangle, \langle a_{r,i} \rightarrow a_r\rangle, \langle a_r \leftrightarrow K_j\rangle, \langle K_j \rightarrow l_j\rangle, \langle l_j \leftrightarrow D\rangle \\
& \mid l_i : (\text{ADD}(r), l_j) \in R\}\,, \\
P_3 \;\supset\; & \{\langle e \leftrightarrow l_i\rangle,\ \langle l_i \rightarrow a_{r,i}\rangle, \langle a_{r,i} \leftrightarrow l_i'\rangle, \langle a_{r,i} \rightarrow t\rangle,\ \langle l_i' \rightarrow K_j\rangle, \\
& \langle K_j \leftrightarrow e\rangle \mid l_i : (\text{ADD}(r), l_j) \in R\}\,.
\end{aligned}
$$

The interplay of these programs in the two cells consumes the instruction label l_i from the environment, and after simulating the effect of the instruction corresponding to l_i (that is, after exporting an object a_r into the environment), it produces the label of the next instruction l_j. Note that if there is no l_i' present in the environment when the program $\langle a_{r,i} \leftrightarrow l_i'\rangle$ of P_3 should be used, then the object t is produced by the program $\langle a_{r,i} \rightarrow t\rangle$ which prevents the system from ever reaching an accepting configuration.

To simulate the instructions of type $l_i : (\text{READ}(a), l_j)$, the system uses its component which corresponds to the particular instruction, (e, P_{l_i}), with the programs

$$
P_{l_i} = \left\{\langle e{\leftrightarrow}l_i\rangle,\ \langle l_i \xrightarrow{T} a\rangle,\ \langle a{\rightarrow}l_j\rangle, \langle l_j{\leftrightarrow}e\rangle\right\}.
$$

These programs can be applied when l_i appears in the environment. They read an input symbol a while exchanging l_i for l_j in the environment.

For each subtract instruction $l_f : (\text{CHECKSUB}(r), l_g, l_n)$ there are the following programs in P_1, P_4, P_5 and in P_6.

$$
\begin{aligned}
P_1 \ &\supset \{\langle D \leftrightarrow L_f\rangle,\ \langle L_f \rightarrow E_f\rangle,\ \langle E_f \rightarrow F_f\rangle,\ \langle F_f \rightarrow \$_f\rangle,\ \langle \$_f \leftrightarrow D\rangle\}, \\
P_4 \ &\supset \big\{\langle e \leftrightarrow l_f\rangle, \langle l_f \rightarrow L_f\rangle, \langle L_f \leftrightarrow l'_f\rangle, \langle l'_f \rightarrow L'_f\rangle, \langle L'_f \leftrightarrow l''_f\rangle, \\
&\qquad \langle l''_f \rightarrow L'''_f\rangle, \langle L'''_f \rightarrow L''_f\rangle, \langle L''_f \leftrightarrow e\rangle, \langle L_f \rightarrow t\rangle, \langle L'_f \rightarrow t\rangle\big\}, \\
P_5 \ &\supset \big\{\langle e \leftrightarrow L'_f\rangle, \langle L'_f \rightarrow l'_f\rangle, \langle l'_f \leftrightarrow a_r\rangle, \langle l'_f \leftrightarrow \$_f\rangle, \langle \$_f \rightarrow \bar{l}_n\rangle, \\
&\qquad \langle a_r \rightarrow e\rangle, \langle \bar{l}_n \leftrightarrow e\rangle\big\}, \\
P_6 \ &\supset \big\{\langle e \leftrightarrow L''_f\rangle, \langle L''_f \rightarrow l'_f\rangle, \langle l'_f \leftrightarrow \$_f\rangle, \langle \$_f \rightarrow l_g\rangle, \langle l_g \leftrightarrow e\rangle, \\
&\qquad \langle l'_f \leftrightarrow \bar{l}_n\rangle, \langle \bar{l}_n \rightarrow l_n\rangle, \langle l_n \leftrightarrow e\rangle\big\}.
\end{aligned}
$$

The subtract instructions are simulated by the rules above as follows. We will denote some of the objects in the environment by $[\ldots]$, and we first assume that they always contain a sufficient amount of l'_i, l''_i objects for any $l_i \in H$.

Let us first consider the case when the subtract instruction is applied with at least one object a_r in the environment. The configuration of such a system can be denoted by $(l_f a_r[\ldots], D, e, e, e, e, e, \ldots, e)$.

In the beginning, the fourth component can apply its first three pro-

grams, producing the configuration

$$(l_f a_r[\ldots], D, e, e, e, e, e, \ldots, e) \Rightarrow \ldots$$
$$\ldots \Rightarrow (L_f a_r[\ldots], D, e, e, l'_f, e, e, \ldots, e),$$

when the first component can also apply its first program, the two components producing together

$$(L_f a_r[\ldots], D, e, e, l'_f, e, e, \ldots, e) \Rightarrow (D a_r[\ldots], L_f, e, e, L'_f, e, e, \ldots, e),$$

and then

$$(L'_f D a_r[\ldots], E_f, e, e, l''_f, e, e, \ldots, e).$$

Now the first, fourth, and fifth components work simultaneously to produce in three steps

$$(L'_f D a_r[\ldots], E_f, e, e, l''_f, e, e, \ldots, e) \Rightarrow \ldots$$
$$\ldots \Rightarrow (\$_f L''_f[\ldots], D, e, e, e, a_r, e, \ldots, e),$$

removing one object a_r from the environment (corresponding to the decrement of the register r).

In the final phase, the fifth component applies its program $\langle a_r \rightarrow e \rangle$, while the sixth component uses its first five programs to produce in five steps the instruction label l_g in the environment

$$(\$_f L''_f[\ldots], D, e, e, e, a_r, e, \ldots, e) \Rightarrow \ldots \Rightarrow (l_g[\ldots], D, e, e, e, e, e, \ldots, e),$$

corresponding to the next instruction to be executed.

Let us now consider the case when the environment does not contain any a_r objects. This corresponds to the situation when the register to be decremented stores the value zero, which means that the label of the next instruction that the simulation should produce is l_n.

The first seven steps of the system are identical to the previous case, thus, we obtain

$$(l_f[\ldots], D, e, e, e, e, e, \ldots, e) \Rightarrow \ldots \Rightarrow (D[\ldots], \$_f, e, e, L''_f, l'_f, e, \ldots, e).$$

Now, the first and the fourth components can use their programs $\langle \$_f \leftrightarrow D \rangle$ and $\langle L''_f \leftrightarrow e \rangle$, respectively, but the fifth component has no applicable program, so we get

$$(D[\ldots], \$_f, e, e, L''_f, l'_f, e, \ldots, e) \Rightarrow (\$_f L''_f[\ldots], D, e, e, e, l'_f, e, \ldots, e),$$

and now the fifth component also can start to apply its programs again. Together with the sixth component they produce in six steps

$$(D[\ldots], \$_f, e, e, L''_f, l'_f, e, \ldots, e) \Rightarrow \ldots \Rightarrow (l_n[\ldots], D, e, e, e, e, e, \ldots, e),$$

which is the desired configuration.

Note that if there is an insufficient amount of objects l'_i, l''_i for $l_i \in H$ is present in the environment at some moment of the computation (not enough of them were produced in the initial phase of the simulation by the first and the second component), then the programs $\langle L_f \rightarrow t \rangle$ and

$\langle L'_f \rightarrow t \rangle$ in the fourth component do not allow the computation to enter into a final configuration.

From these considerations we can see that after the initialization phase, all instructions of the register machine M can be simulated by the genPCol automaton, maintaining the counter contents and reading input symbols, as necessary. If the label of the halt instruction, l_h is produced, and there is no object t present in any of the component cells, then the configuration is considered to be final (it also halts, since there is no program for processing the object l_h). ■

4.1.2 Restricted programs and their restricted power

We have shown that genPCol automata, when considering no extra restrictions towards their rules/programs, are able to recognize the class of recursively enumerable languages even with capacity one. Now let us show the capabilities of different restricted modes, namely com-tape and all-tape. The next theorem is from [22], it shows that the systems in question with permutation mapping and capacity one can accept non-regular languages.

Theorem 4.2.

$$\mathcal{L}_{perm}(\textit{genPCol}, X(1)) \setminus \mathcal{L}(\textit{REG}) \neq \emptyset$$

$$\textit{for } X \in \{\textit{all-tape}, \textit{com-tape}\}.$$

Proof. Consider $\Pi = (\Sigma \cup \{e\}, e, w_E, (e, P_1), (e, P_2), (e, P_3), F)$, $\Sigma = \{a, b\}$, the genPCol automaton with the sets of programs as

$$
\begin{aligned}
P_1 &= \left\{ p_{1,1} : \langle e \overset{\alpha}{\to} \$ \rangle,\ p_{1,2} : \langle \$ \overset{\beta}{\leftrightarrow} e \rangle \right\}, \\
P_2 &= \left\{ p_{2,1} : \langle e \overset{\alpha}{\to} a \rangle,\ p_{2,2} : \langle a \overset{\beta}{\leftrightarrow} e \rangle,\ p_{2,3} : \langle e \overset{\alpha}{\to} b \rangle,\ p_{2,4} : \langle b \overset{\beta}{\leftrightarrow} \$ \rangle \right\}, \\
P_3 &= \left\{ p_{3,1} : \langle e \overset{\beta}{\leftrightarrow} b \rangle,\ p_{3,2} : \langle b \overset{\beta}{\leftrightarrow} a \rangle,\ p_{3,3} : \langle a \overset{\alpha}{\to} b \rangle \right\},
\end{aligned}
$$

where $\alpha, \beta \in \{\varepsilon, T\}$ (that is, the rules are either non-tape rules or tape rules), and the set of accepting configurations is defined as

$$F = \{(u, \$, \$, b) \mid u \in (\Sigma \setminus \{a\})^+ \text{ and } |u|_\$ \geq 2\}.$$

This system works as follows. The first cell starts producing $\$$ objects indefinitely, while the second cell produces as, both of them sending the produced objects to the environment. If the second cell produces a b instead of an a, then the third cell can also start to work, it consumes the as from the environment. Note that the number of these consuming computational steps are the same as the number of as sent to the environment before, so the number of as and bs in the strings of the accepted languages are related.

Let us see the example for computing $\{a, \$\}\{b, \$\}^2$ if we have com-tape programs, that is, if $\alpha = \varepsilon$ and $\beta = T$. The sequence of configurations are as follows. (The applied sequence of programs are

indicated over the arrows.)

$$(\varepsilon, e, e, e) \overset{\{p_{1,1},p_{2,1}\}}{\Longrightarrow} (\varepsilon, \$, a, e) \overset{\{p_{1,2},p_{2,2}\}}{\Longrightarrow} (\$a, e, e, e) \overset{\{p_{1,1},p_{2,3}\}}{\Longrightarrow}$$
$$(\$a, \$, b, e) \overset{\{p_{1,2},p_{2,4}\}}{\Longrightarrow} (a\$b, e, \$, e) \overset{\{p_{1,1},p_{3,1}\}}{\Longrightarrow} (a\$, \$, \$, b)$$
$$\overset{\{p_{1,2},p_{3,2}\}}{\Longrightarrow} (b\$\$, e, \$, a) \overset{\{p_{1,1},p_{3,3}\}}{\Longrightarrow} (b\$\$, \$, \$, b)$$

It is easy to see that the system reads sequences of multisets which are mapped to the strings of the language

$$L_{com-tape}(\Pi, f_{perm}) = \{a, \$\}^n \{b, \$\}^{n+1} \text{ , for } n \geq 1.$$

Moreover, if $\alpha = \beta = T$, that is, if we have all-tape programs, the accepted language is

$$L_{all-tape}(\Pi, f_{perm}) = \{a, \$\}^{2n} \{b, \$\}^{2n+2} \text{ , for } n \geq 1.$$

The accepted language is non-regular, in both cases. ■

We now know, that in case of capacity one and mapping function f_{perm}, all restricted genPCol automata are able to accept non-regular languages. Now let us examine systems with a higher capacity and a restricted rule set. As a consequence of Theorem 4.1 (see point 3. of Proposition 4.1), we have

$$\mathcal{L}_{perm}(\text{genPCol}, *(2)) = \mathcal{L}(\text{RE}).$$

(Note that the above statement was shown directly in [21].) Based on the construction of the proof of Theorem 4.1, we can show that also the class

of all-tape languages of systems with capacity of at least two include all recursively enumerable languages. Given a recursively enumerable language L, the idea is to take a system of capacity one which, when any kind of programs are allowed accept L, and transform it to a system of capacity two having a tape rule in each program by adding "dummy" tape rules which do not interfere with the work of the rest of the system. (Note that this idea does not work in the case of com-tape languages.)

The following proposition is from [22].

Proposition 4.2.

$$\mathcal{L}_{perm}(\textit{genPCol}, \textit{all-tape}(k)) = \mathcal{L}(\textit{RE}) \textit{ for } k \geq 2.$$

Proof. The proof is based on the construction of the proof of Theorem 4.1. Instead of

$$\Pi = (V, e, w_e, (e, P_1), \ldots, (e, P_6), (e, P_{l_1}), \ldots, (e, P_{l_n}), F),$$

let us consider

$$\Pi = (V, e, w_e, (\{ee\}, P_1), \ldots, (\{ee\}, P_6), (\{ee\}, P_{l_1}), \ldots, (\{ee\}, P_{l_n}), F),$$

that is, increasing the capacity of Π to two. Furthermore, for every $p \in P_1 \cup \ldots \cup P_6 \cup \bigcup_{i=1}^{n} P_{l_i}$, we add a second rule in the form of $e \overset{T}{\rightarrow} e$ or $e \overset{T}{\leftrightarrow} e$ to satisfy the capacity requirements. Analogously to the proof of Theorem 4.1, any recursively enumerable language can be accepted.

Based on the third point of Proposition 4.1, this statement holds for any $k \geq 2$. ■

More interesting is the case when com-tape programs are considered. We show that the power of such systems are rather limited, as they can be simulated by Turing machines with restricted one-way logarithmic space. See Section 2.2 for the formal definition of such systems and the language class r1LOGSPACE.

The following theorem is from [22].

Theorem 4.3.

$$\mathcal{L}_{perm}(\textit{genPCol}, \textit{com-tape}(k)) \subseteq \textit{r1LOGSPACE for any } k \geq 1.$$

Proof. We show that a genPCol automaton with com-tape programs can be simulated by a Turing machine in restricted logarithmic space, by creating a Turing machine program that is able to store the encodings of the genPCol automaton configurations on the work-tapes, and based on the currently stored configuration, simulates its computation by computing and storing the next configuration and reading the necessary input symbols from the input tape. We can encode a configuration on the work-tapes in such a way that it uses restricted logarithmic space.

Formally speaking, let $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ be a genPCol automaton. Let us define the set of transitions of Π as $Tr \subseteq 2^P$,

such that, $t \in Tr,\ p_1, p_2 \in t$ 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$(t) \subseteq \{1, \ldots, n\}$ 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 $t \in Tr$ 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 n-tuple inside(t).

Now we construct a one-way Turing machine M which sim-

ulates 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,
- $\Gamma = \{0, \ldots, 9\}$ is the work-tape alphabet,
- $Q = \{q_0\} \cup \{[t, u_1, \ldots, u_n] \mid t \in Tr,\ u_i \in V^k,\ 1 \leq i \leq n\}$ is the set of internal states with q_0 being the initial state, and $F' = \{q_f\}$ the set of accepting states,
- $\delta_M : V \cup \{\varepsilon\} \times Q \times \Gamma^m \to 2^{Q \times \Gamma^m \times \{left, right, none\}^m}$ 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 non-deterministically 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 non-deterministically 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 $|\text{import}(t)|_a$ 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 $i \in \text{blocked}(t)$, 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 export(t) and subtracting the multiset 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 communication rules are tape rules, 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. ■

Corollary 4.1. *If we restrict all-tape mode by excluding the possibility to pump objects into the environment without reading anything, that is, excluding "erasing" tape rules of the form* $a \xrightarrow{T} e$*, then again the number of objects present in any configuration of* Π *can be bounded by a linear*

function of the number of executed computational steps, so we get

$$\mathcal{L}_{perm}(\textit{genPCol}, \textit{all-tape}'(k)) \subseteq \textit{r1LOGSPACE for any } k \geq 1.$$

4.2 A more general approach for input mappings

In the previous chapter, it was shown that with unrestricted programs systems of capacity *o*ne generate any recursively enumerable language, that is,

$$\mathcal{L}_{perm}(\text{genPCol}, *(k)) = \mathcal{L}(\text{RE}),\ k \geq 1.$$

The following theorem form [25] shows that this holds also for the class of mappings $\mathcal{F}_{\text{TRANS}}$.

Theorem 4.4.

$$\mathcal{L}_{\textit{TRANS}}(\textit{genPCol}, *(k)) = \mathcal{L}(\textit{RE}),\ k \geq 1.$$

Proof. Let $L \subseteq \Sigma^*$ be an arbitrary recursively enumerable language and let Π be the genPCol automaton constructed in Theorem 4.1 with $L(\Pi, f_{perm}) = L$. The idea of the simulation is to have an object in the environment corresponding to the label of the instruction which is to be simulated next. The cells of the system "process" the instruction

label in such a way that the necessary modifications of the configuration are implemented, and the label of the next instruction is sent to the environment. By observing the components which are responsible for the simulation of the tape reading instructions, we may notice that there is one simulating component constructed for each such instruction $l_i : (\text{READ}(a), l_j)$ with label l_i

$$P_{l_i} = \{\langle e \leftrightarrow l_i \rangle,\ \langle l_i \overset{T}{\rightarrow} a \rangle,\ \langle a \rightarrow l_j \rangle, \langle l_j \leftrightarrow e \rangle\}.$$

These programs can be applied when l_i appears in the environment. They read an input symbol a while exchanging l_i for l_j in the environment. Notice that the actual "reading" of the input symbol is realized by the tape program $\langle l_i \overset{T}{\rightarrow} a \rangle$, and note also, that the system is constructed in such a way that at most one such program might be applied in any computational step. This means, that the set of input multisets of Π correspond to the singleton multisets consisting of the symbols of the input alphabet of Π. Thus, if we define $f_1 \in \mathcal{F}_{\text{TRANS}}$ as $f_1(a) = \{a\}$ for any $a \in \Sigma$, then we have $L = L(\Pi, f_{perm}) = L(\Pi, f_1)$. This implies that $L \in \mathcal{L}_{TRANS}(\text{genPCol}, *(1))$, and considering the third inclusion of Proposition 4.1, our statement holds. ■

A similar result holds for all-tape systems with capacity at least two. From Proposition 4.2 we have that

$$\mathcal{L}_{perm}(\text{genPCol}, \text{all-tape}(k)) = \mathcal{L}(\text{RE}) \text{ for } k \geq 2,$$

and we can show the same for systems with input mappings from $\mathcal{F}_{\text{TRANS}}$.

The following theorem is from [25].

Theorem 4.5.

$$\mathcal{L}_{\textit{TRANS}}(\textit{genPCol}, \textit{all-tape}(k)) = \mathcal{L}(\textit{RE}\,) \textit{ for } k \geq 2.$$

Proof. The proof is based on the construction of the proof of Theorem 4.1 and Theorem 4.4 above. For any recursively enumerable language $L \subseteq \Sigma^*$, we can obtain a genPCol automaton of capacity *two* accepting L with all-tape type of programs by simply putting one more e object into each cell, and adding the dummy tape rules $e \xrightarrow{T} e$ or $e \xleftrightarrow{T} e$ to every program. This way we get that $\mathcal{L}_{\text{TRANS}}(\text{genPCol}, \text{all-tape}(2)) = \mathcal{L}(\text{RE})$. Since the third inclusion of Proposition 4.1, the statement also holds for any $k > 2$. ■

Next, we are going to show that in case of capacity *one*, all regular languages, and also some non-regular ones, can be characterized not only with all-tape, but also with com-tape programs.

The following theorem is also from [25].

Theorem 4.6.

$$\mathcal{L}(\textit{REG}\,) \subset \mathcal{L}_{\textit{TRANS}}(\textit{genPCol}, X(1)),$$

$$\textit{for } X \in \{\textit{all-tape}, \textit{com-tape}\}.$$

Proof. To prove that any regular language can be described by genPCol automata of capacity one having all-tape or com-tape type of programs, consider an arbitrary regular language L, and the finite automaton $M = (Q, \Sigma, q_0, A, \delta)$ with $L(M) = L$, where

- Q is the set of internal states,
- Σ is the input alphabet,
- $q_0 \in Q$ is the initial state, A is the set of final states, and
- $\delta : Q \times \Sigma \to Q$ is the transition function.

Consider the genPCol automaton $\Pi_1 = (V, e, \emptyset, (w_0, P), F)$ with

- $V = \{(q, a, s) \mid \delta(q, a) = s\} \cup \{(q_0)\}$ alphabet,
- $w_0 = (q_0)$ starting contents of the $0.$ cell, where q_0 is the initial state of M, and
- $F = \{(\emptyset, (s, a, q)), (\emptyset, (q)) \mid q \in A\}$ set of accepting configurations.

The set of all-tape programs is as follows

$$P = \left\{ \langle (q, a, s) \xrightarrow{T} (s, b, r) \rangle \;\mid\; \delta(q, a) = s, \delta(s, b) = r \right\} \cup \left\{ \langle (q_0) \xrightarrow{T} (q_0, a, s) \rangle \;\middle|\; \delta(q_0, a) = s \right\}.$$

The sets of accepted multiset sequences of Π_1 are

$$A(\Pi_1) = \{(q_0, a_1, q_1) \dots (q_n, a_{n+1}, q_{n+1}) \mid q_0, q_1, \dots, q_{n+1} \text{ is a sequence of states ending in an accepting state while reading the string } a_1 \dots a_{n+1}\}.$$

Now, if we define f_1 by $f_1((q, a, s)) = \{a\}$, then we have $f_1 \in \mathcal{F}_{\text{TRANS}}$ and $L(\Pi_1, f_1) = L(M) = L$.

For the case of com-tape programs, we construct the following system, $\Pi_2 = (V, e, (q_0), (e, P), F)$ with

- $V = \{(q, a, s) \mid \delta(q, a) = s\} \cup \{(q_0)\}$ alphabet,
- $w_0 = (q_0)$ starting contents of the $0.$ cell, where q_0 is the initial state of M, and
- $F = \{((s, a, q), u), ((q), u) \mid q \in A\}$ set of accepting configurations.

The set of com-tape programs is as follows

$$P = \{\langle e \to (q, a, s)\rangle,\ \langle (s, b, r) \overset{T}{\leftrightarrow} (q, a, s)\rangle,\ \langle (q_0, a, s) \overset{T}{\leftrightarrow} (q_0)\rangle,$$
$$\langle (q_0) \to (q, a, s)\rangle \mid \delta(q, a) = s,\ \delta(q_0, a) = s,\ \delta(s, b) = r\}.$$

The sets of accepted multiset sequences of Π_2 are again

$$A(\Pi_2) = \{(q_0, a_1, q_1) \dots (q_n, a_{n+1}, q_{n+1}) \mid q_0, q_1, \dots, q_{n+1} \text{ is a sequence of states ending in an accepting configuration while reading the string } a_1 \dots a_{n+1}\}.$$

Taking the same f_1 input mapping as above, we have $L(\Pi_2, f_1) = L(M) = L$. As Π_1 and Π_2 are systems with all-tape and com-tape programs, respectively, we have shown that genPCol automata of capacity one with both of these types of programs and input mappings from $\mathcal{F}_{\text{TRANS}}$ are able to characterize any regular language.

To see that the inclusion is strict, let us construct

$$\Pi_3 = (\Sigma \cup \{e\}, e, w_E, (e, P_1), (e, P_2), (e, P_3), F),$$

with the sets of programs as

$$\begin{aligned} P_1 &= \{\langle e \overset{\alpha}{\to} \$\rangle, \langle \$ \overset{\beta}{\leftrightarrow} e\rangle\}, \\ P_2 &= \{\langle e \overset{\alpha}{\to} a\rangle, \langle a \overset{\beta}{\leftrightarrow} e\rangle, \langle e \overset{\alpha}{\to} b\rangle, \langle b \overset{\beta}{\leftrightarrow} \$\rangle\}, \\ P_3 &= \{\langle e \overset{\beta}{\leftrightarrow} b\rangle, \langle b \overset{\beta}{\leftrightarrow} a\rangle, \langle a \overset{\alpha}{\to} b\rangle\}, \end{aligned}$$

where α, β can be empty, or they also can be T, that is, the rules are either non-tape rules or tape rules, and set of accepting configurations

$$F = \{(u, \$, \$, b) \mid u \in \mathcal{M}(\Sigma - \{a\}),\ u \neq \emptyset, \text{and } |u|_{\$} \geq 2\}.$$

If we have com-tape programs, that is, if α is empty and $\beta = T$, then let the resulting system be denoted by Π_1. It accepts the sequences of multisets

$$A(\Pi_1) = \left\{ \{\$a\}^n \{\$b\}^{n+1} \;\middle|\; n \geq 1 \right\}.$$

Thus, if we take $f_1 : \{\$a, \$b\} \to 2^{\{c,d\}^*}$ with $f_1(\$a) = c$ and $f_1(\$b) = d$,

then we obtain the language

$$L_{com-tape}(\Pi_1, f_1) = \left\{ c^n d^{n+1} \;\middle|\; n \geq 1 \right\},$$

which is not a regular language.

If $\alpha = \beta = T$, that is, if we have all-tape programs, then let the resulting system be denoted by Π_2. The accepted sequences of multisets are

$$A(\Pi_2) = \left\{ \{\$a\}^{2n} \{\$b\}^{2n+1} \;\middle|\; n \geq 1 \right\}.$$

In this case, taking the same f_1 as above, the accepted language is

$$L_{all-tape}(\Pi_2, f_1) = \left\{ c^{2n} d^{2n+1} \;\middle|\; n \geq 1 \right\}.$$

The accepted language is non-regular, in any of the cases, so the strictness of the inclusion in the statement follows. ■

4.3 Generalized P colony automata and their relation to P automata

In this chapter, we are going to compare the power of genPCol automata and P automata. A P automaton is an antiport P system where, similarly to P colony automata, the accepted string language is defined by mapping the accepted multiset sequence (that is, the sequence of multisets entering the skin membrane during the steps of an accepting computation) to symbols of a given alphabet.

More formally, a *P automaton* (see [14]) is a membrane system

$$\Pi = (V, \mu, w_1, \ldots, w_k, P_1, \ldots, P_k, F)$$

with object alphabet V, membrane structure μ, initial contents (multisets) of the ith region $w_i \in V^*$, $1 \leq i \leq k$, sets of antiport rules P_i, $1 \leq i \leq k$, and a set of accepting configurations F. An antiport rule is of the form $(u, in; v, out)$, where $u, v \in \mathcal{M}(V)$ are finite multisets over V. If such a rule is applied in a region, then the objects of u enter from the parent region and, in the same step, objects of v leave to the parent region.

The configurations of the P automaton can be changed by transitions where the rules are applied in the sequential mode (seq) or in the maximally parallel mode (par). In the first case one rule is applied in each region in every step, in the second case as many rules are applied simultaneously in the regions at the same step as possible. Thus, a transition in the P automaton Π is $(v_1, \ldots, v_k) \in \delta_\Pi(u_0, u_1, \ldots, u_k)$, where δ_Π denotes the transition relation, $u_1, \ldots, u_k$ are the contents of the k regions, u_0 is the multiset entering the system from the environment, and $v_1, \ldots, v_k$, respectively, are the contents of the k regions after performing the transition.

In this way, there is a sequence of multisets which enter the system from the environment during the steps of its computation. If the computation is accepting, that is, if it reaches a configuration from F, the

set of accepting configurations, then this multiset sequence is called an accepted multiset sequence, and denoted by $A_X(\Pi)$, $X \in \{seq, par\}$ for a P automaton Π.

The language accepted by a P automaton Π is defined with the use of input mappings in the same way as in the case of genPCol automata, see Definition 3.3. Let Π be a P automaton and let $f : in(\Pi) \to 2^{\Sigma^*}$ be a mapping, such that $f(u) = \{\varepsilon\}$ if and only if u is the empty multiset, where $in(\Pi)$ is the set of possible input multisets. The accepted language is defined as

$$L_X(\Pi, f) = \{f(u_1) \dots f(u_s) \in \Sigma^* \mid u_1 \dots u_s \in A_X(\Pi)\},$$

for $X \in \{par, seq\}$ where par and seq denotes maximally parallel or sequential application of the antiport rules, respectively.

The class of languages belonging to r1LOGSPACE is important from the point of view of P automata, as the following results are known. If we denote by $\mathcal{L}_X(PA, f_{perm})$, $X \in \{par, seq\}$ the class of languages characterized by P automata with input mapping f_{perm} (using maximal parallel or sequential rule application), then we have from [16]

$$\mathcal{L}_X(PA, f_{perm}) \subset \text{r1LOGSPACE}, \ X \in \{par, seq\}.$$

Now we show that genPCol automata even with capacity two are able to accept languages that P automata cannot.

The following theorem is from [25].

Theorem 4.7.

$$\mathcal{L}_{perm}(\textit{genPCol}, \textit{com-tape}(2)) \setminus \mathcal{L}_X(PA, f_{perm}) \neq \emptyset,\ X \in \{par, seq\}.$$

Proof. Consider the language $L = \{(ab)^n(cd)^n \mid n \geq 1\}$ which, according to [18] cannot be accepted by any P automaton using the mapping f_{perm}. The following genPCol automaton accepts L with f_{perm}.

Let $\Pi = (\Sigma, e, \varepsilon, (ea, P), F)$ with $\Sigma = \{a, b, c, d\}$, $F = \{(u, ec) \mid u \in \{\Sigma \setminus \{b\}\}^+\}$ and

$$P = \Big\{p_1 : \langle e \to b, a \overset{T}{\leftrightarrow} e\rangle,\ p_2 : \langle b \overset{T}{\leftrightarrow} a, e \to e\rangle,\ p_3 : \langle e \to c, a \to e\rangle,$$
$$p_4 : \langle c \overset{T}{\leftrightarrow} b, e \to d\rangle,\ p_5 : \langle d \overset{T}{\leftrightarrow} e, b \to c\rangle\Big\}.$$

Let us show two example computations:

$$(\varepsilon, ea) \overset{p_1}{\Longrightarrow} (a, be) \overset{p_2}{\Longrightarrow} (b, ae) \overset{p_3}{\Longrightarrow} (b, ce) \overset{p_4}{\Longrightarrow} (c, bd) \overset{p_5}{\Longrightarrow} (cd, ce),$$

and

$$(\varepsilon, ea) \overset{p_1}{\Longrightarrow} (a, be) \overset{p_2}{\Longrightarrow} (b, ae) \overset{p_1}{\Longrightarrow} (ab, eb) \overset{p_2}{\Longrightarrow} (bb, ae) \overset{p_3}{\Longrightarrow}$$
$$(bb, ce) \overset{p_4}{\Longrightarrow} (cb, bd) \overset{p_5}{\Longrightarrow} (cbd, ce) \overset{p_4}{\Longrightarrow} (ccd, bd) \overset{p_5}{\Longrightarrow} (ccdd, ce).$$

The computations accept the strings $abcd$ and $(ab)^2(cd)^2$ respectively. In general, the system above first reads a string $(ab)^n$ while sending n copies of b into the environment. Then in the second phase, as many cds are read, as the number of bs that can be found in the environment. ■

Now we examine the case of mappings from $\mathcal{F}_{\text{TRANS}}$. We show that the class of languages that characterized by genPCol automata with com-tape rules is also included in r1LOGSPACE.

The following theorem is also from [25].

Theorem 4.8.

$$\mathcal{L}_{TRANS}(\textit{genPCol}, \textit{com-tape}(k)) \subseteq \textit{r1LOGSPACE}), \textit{ for } k \geq 1.$$

Proof. We have shown in Section 4.1.1, that a similar inclusion holds for systems with permutation mappings.

Note that the number of different possible states of the component cells is $(m^k)^l$ where we have an alphabet of cardinality m, and l cells having the capacity k. To take into account also the different states of the environment, we need to count how many objects have been exported from the component cells, since all symbols different from e that are present in the environment during any configuration must have been exported from the cells with the use of communication rules. Since all communication rules are tape rules, the number of objects inside the multisets of the multiset sequence read by the system cannot be less than the number of non-e objects present in the environment.

Let us examine now the relationship of the length of the multiset sequence read and the corresponding string obtained by applying the input mapping to the multiset sequence. Since the cardinality of the

read multisets is at most $l \cdot k$, the input mapping is from $\mathcal{F}_{\text{TRANS}}$, and each nonempty multiset is mapped to a nonempty string, the number of objects in the environment is at most $c_1 \cdot n$ where n is the length of the string obtained by the input mapping from the multiset sequence that is read so far, and $c_1 \in \mathbb{N}$ is an appropriate constant. Thus, after reading a string of length n, there can be at most $c_1 \cdot n$ non-e symbols in the environment.

Such a system can be simulated by a Turing machine in restricted logarithmic space by creating a Turing machine program that is able to store the encodings of the genPCol automaton configurations on the work-tapes, and based on the currently stored configuration, is able to simulate its computation by computing and storing the next configuration and reading the necessary input symbols from the input tape. The number of these symbols can be recorded on the work-tapes of the Turing machine in a way described in the proof of Theorem 4.3 using restricted logarithmic space. ■

As P automata with sequential rule application and mappings from $\mathcal{F}_{\text{TRANS}}$ characterize exactly the class r1LOGSPACE (see [13]), we also have the following.

Corollary 4.2.

$$\mathcal{L}_{\textit{TRANS}}(\textit{genPCol}, \textit{com-tape}(k)) \subseteq \mathcal{L}_{\textit{seq}}(PA, \mathcal{F}_{\textit{TRANS}}),\ k \geq 1.$$

5 An application of generalized P colony automata

5.1 Deterministic parsing

Lewis and Stearns [31] and Knuth [29] were the first ones to define the theory of top-down parsing and the class of LL(k) grammars. A context-free grammar is said to be in the class of LL(k) grammars, if it can be parsed top-down, in a deterministic way using k lookahead symbols, where the parsing constructs the leftmost derivation. In other words, an LL(k) parser is a one state deterministic push-down transducer, that produces the left parse of an input string. A language is said to be an LL(k) language, if it can be generated by an LL(k) grammar. LL(k) languages stand in a hierarchy based on their k variable. Not every context-free

language is an LL(k) language, for instance, consider the following example language: $\{a^n b^n \cup a^n c^n \mid n \in \mathbb{N}\}$, and a grammar G, that generates this language:

$$\begin{aligned} S &\rightarrow A \mid B, \\ A &\rightarrow a\ A\ b \mid \varepsilon, \\ B &\rightarrow a\ B\ c \mid \varepsilon. \end{aligned}$$

It is easy to see, that no matter how big k is, we cannot construct a deterministic LL(k) parser based on G. There will always be an $n > k$ that makes deterministic parsing impossible. Not only G is a non LL(k) grammar, but it is impossible to construct any LL(k) grammar that could generate the given example language.

In this chapter we examine the parsing properties of genPCol automata in terms of programs and rules and we study the possibility of deterministically parsing the languages characterized by these devices. We define the so-called LL(k) condition for these types of automata, which enables deterministic parsing with k lookahead symbol, as in the case of context-free LL(k) languages, and present that by using generalized P colony automata we can deterministically parse context-free languages that are not LL(k) in the "original" sense.

Let $U \subset \Sigma^*$ be a finite set of strings over some alphabet Σ. Let $\text{FIRST}_k(U)$ denote the set of length k prefixes of the elements of U for

some $k \geq 1$, that is, let

$$\text{FIRST}_k(U) = \{pref_k(u) \in \Sigma^* \mid u \in U\}$$

where $pref_k(u)$ denotes the string of the first k symbols of u if $|u| \geq k$, or $pref_k(u) = u$ otherwise.

Definition 5.1. *Let $\Pi = (V, e, w_E, (w_1, P_1), \ldots, (w_n, P_n), F)$ be a genPCol automaton, let $f : (V \setminus \{e\})^* \to 2^{\Sigma^*}$ be a mapping, and let $c_0, c_1, \ldots, c_s$ be a sequence of configurations with $c_i \Longrightarrow c_{i+1}$ for all $0 \leq i \leq s-1$.*

We say that the P colony Π is LL(k) for some $k \geq 1$ with respect to the mapping f, if for any two distinct sets of programs applicable in configuration c_s, $P_{c_s}, P'_{c_s} \in Acc_{c_s}$ with $P_{c_s} \neq P'_{c_s}$, the next k symbols of the input string that is being read determines which of the two sequences are to be applied in the next computational step, that is, the following holds.

Consider two computations

$$c_s \overset{P_{c_s}}{\Longrightarrow} c_{s+1} \overset{P_{c_{s+1}}}{\Longrightarrow} \ldots \overset{P_{c_{s+m}}}{\Longrightarrow} c_{s+m+1},$$

and

$$c_s \overset{P'_{c_s}}{\Longrightarrow} c'_{s+1} \overset{P'_{c_{s+1}}}{\Longrightarrow} \ldots \overset{P'_{c_{s+m'}}}{\Longrightarrow} c'_{s+m'+1}$$

where $u_{c_s} = read(P_{c_s})$ and $u_{c_s+i} = read(P_{c_s+i})$ for $1 \leq i \leq m$, and similarly $u'_{c_s} = read(P'_{c_s})$ and $u'_{c_s+i} = read(P_{c_s+i'})$ for $1 \leq i \leq m'$, thus, the two sequences of input multisets are

$$u_{c_s} u_{c_s+1} \ldots u_{c_s+m} \text{ and } u'_{c_s} u'_{c_s+1} \ldots u'_{c_s+m'}.$$

Assume that these sequences are long enough to "consume" the next k symbols of the input string, that is, for w and w' with

$$w \in f(u_{c_s})f(u_{c_s+1})\dots f(u_{c_s+m}) \text{ and } w' \in f(u'_{c_s})f(u'_{c_s+1})\dots f(u'_{c_s+m'}),$$

either $|w| \geq k$ and $|w'| \geq k$, or if $|w| < k$ (or $|w'| < k$), then c_{s+m+1} (or $c_{s+m'+1}$) is a halting configuration.

Now, the P colony Π is LL(k), if for any two computations as above,

$$FIRST_k(w) \cap FIRST_k(w') = \emptyset.$$

The class of context-free LL(k) languages will be denoted by $\mathcal{L}$(CF,LL(k)) (see for example the monograph [1] for more details), while the languages characterized by genPCol automata satisfying the above defined condition, with input mapping of type f_{perm} or $f \in TRANS$, will be denoted by $\mathcal{L}_X$(genPCol,LL(k)), $X \in \{perm, TRANS\}$.

Now we show that there are context-free languages which are not in $\mathcal{L}$(CF,LL(k)), but can be accepted by genPCol automata satisfying the $LL(1)$ condition.

The following theorem is from [10].

Theorem 5.1.

$\mathcal{L}_X(genPCol, LL(1)) \setminus \mathcal{L}(CF, LL(k)) \neq \emptyset, X \in \{perm, TRANS\}$,

for any $k \geq 1$.

Proof. Let $\Pi = (\{a, b, c, d, f, g, e\}, e, \emptyset, (ea, P_1), F)$ be a genPCol automaton, where

$$\begin{aligned} P_1 = \Big\{ & \langle e \to b, a \overset{T}{\leftrightarrow} e \rangle,\ \langle e \to e, b \overset{T}{\leftrightarrow} a \rangle,\ \langle e \to c, a \overset{T}{\leftrightarrow} e \rangle, \\ & \langle e \to f, a \overset{T}{\leftrightarrow} e \rangle,\ \langle e \to d, c \overset{T}{\leftrightarrow} b \rangle,\ \langle b \to c, d \overset{T}{\leftrightarrow} e \rangle, \\ & \langle e \to g, f \overset{T}{\leftrightarrow} b \rangle,\ \langle b \to f, g \overset{T}{\leftrightarrow} e \rangle \Big\}, and \\ F = \{ & (v, ce), (v, fe) \mid v \in V^*, b \notin v \}. \end{aligned}$$

The language accepted by Π is

$$L(\Pi, f_{perm}) = \{a\} \cup \{(ab)^n a (cd)^n \mid n \geq 1\} \cup \{(ab)^n a (fg)^n \mid n \geq 1\}.$$

To see this, consider the possible computations of Π. The initial configuration is $(\emptyset, ea)$ and there are three possible configurations that can be reached, namely (we denote by $\overset{u}{\Rightarrow}$ a configuration change during which the multiset of symbols u was read by the automaton)

1. $(\emptyset, ea) \overset{a}{\Rightarrow} (a, ce)$,
2. $(\emptyset, ea) \overset{a}{\Rightarrow} (a, fe)$,
3. $(\emptyset, ea) \overset{a}{\Rightarrow} (a, be)$.

The first two cases are non-accepting states, but the derivations cannot be continued, so let us consider the third one.

$$(a, be) \overset{b}{\Rightarrow} (b, ea) \overset{a}{\Rightarrow} (ba, be) \overset{b}{\Rightarrow} (bb, ea) \overset{a}{\Rightarrow} \ldots \overset{b}{\Rightarrow} (b^i, ea).$$

At this point, the computation can follow two different paths again, either

$$(b^i, ae) \overset{a}{\Rightarrow} (b^i a, ec) \overset{c}{\Rightarrow} (b^{i-1}ac, db) \overset{d}{\Rightarrow} (b^{i-1}acd, ce) \overset{c}{\Rightarrow} \ldots \overset{d}{\Rightarrow} (ac^i d^i, ce),$$

or

$$(b^i, ae) \overset{a}{\Rightarrow} (b^i a, ef) \overset{f}{\Rightarrow} (b^{i-1}af, gb) \overset{g}{\Rightarrow} (b^{i-1}afg, fe) \overset{f}{\Rightarrow} \ldots \overset{g}{\Rightarrow} (af^i g^i, fe).$$

In the first phase of the computation, the system produces copies of b and sends them to the environment, then in the second phase these copies of b are exchanged to copies of cd or copies of fg. The system can reach an accepting state when all the copies of b are used, that is, when an equal number of copies of ab and either of cd or of fg were produced.

Note that the system satisfies the LL(1) property, the symbol that has to be read in order to accept a desired input word, determines the set of programs that has to be used in the next computational step.

The language $L(\Pi, f_{perm}) \in \mathcal{L}_{perm}$(genPCol,LL(1)) is not in $\mathcal{L}$(CF,LL(k)) for any $k \geq 1$. If we consider the mapping $f_1 \in TRANS$, $f_1 : \{a, b, c, d, f, g\} \to 2^{\{a,b,c,d,f,g\}}$ with $f_1(x) = \{x\}$ for all $x \in \{a, b, c, d, f, g\}$, then $L(\Pi, f_1) = L(\Pi, f_{perm})$, thus, $\mathcal{L}_{TRANS}$(genPCol,LL(1)) also contains the non-LL(k) context-free language. ■

6 Summary

The last decade gave birth to a number of unconventional models of computation. Several unconventional models are inspired by biology and chemistry. They describe molecules, cells and processes that occur naturally in living systems that have been observed by biologists and chemists. Membrane systems provide a theoretical approach to formulate a model of computation motivated by the structure and functioning of a living cell. A very special membrane system, namely generalized P colony automata was introduced and examined thoroughly in the dissertation.

In Chapter 3, the model itself was defined. We defined the set of input sequences accepted by generalized P colony automata and we defined how a mapping function can describe the language that is accepted. Three base subtypes of genPCol automata were introduced, namely the *unrestricted* type, where we have no syntactical or semantical restrictions towards the programs and rules of a system, the *com-tape* case,

where we restrict the rules by not allowing non-tape variants of communication rules, and the *all-tape* type, where all programs must contain at least one tape rule. Lastly, we demonstrated how this model works by constructing a genPCol automaton accepting non-regular language $\{a^n b^n c^n \mid n \geq 1\}$.

In Chapter 4, the results regarding the computational power of genPCol automata were stated. Starting with permutation mapping, it was proven that in case where we allow any types of rules and programs (thus we are talking about unrestricted case), genPCol automata characterize exactly the class of recursively enumerable languages even with *capacity one*. The proof was constructed by simulating register machines with input tape, that is known to be equivalent with $\mathcal{L}(\mathrm{RE})$.

Continuing our research regarding capacity one, we have shown that in case when all programs have at least one tape rule and in case when all communications are tape rules, it is possible to characterize non-regular languages. Moving on by examining upper bounds and higher capacities, we have shown interesting results. The upper bound of the computational power of com-tape subtype can be described by Turing machines with restricted logarithmic space using a one way input tape. It was shown that a logarithmic function dependant on the length of already read input multiset sequences describes the maximum number of possible configurations that can be stored in the environment at any time, concluding the result mentioned above. For all-tape subtype the exact

same theorem holds if and only if we exclude erasing tape rules, that is, rules of the form of $a \xrightarrow{T} e$. If we do not have this extra restriction, all-tape mode characterizes the class of recursively enumerable languages with a minimum capacity of two.

Moving on, we studied a more general approach to map input sequences to accepted strings, by using functions from the class $\mathcal{F}_{\text{TRANS}}$, the class of mappings between multisets and strings that can be realized through the application of finite transducers to the string representations of multisets. We have shown that they characterize the class of recursively enumerable languages as well, when considering systems with capacity one working in unrestricted mode. The next result shows that the computational power of capacity one systems with restricted set of rules (all-tape, com-tape) are strictly more powerful than regular languages. It is proven, that they can simulate finite automata and accept non-regular languages as well.

The last subsection describes the relation of the model to P automata. We show that in case of permutation mapping, there are languages that P automata cannot accept, but genPCol automata with capacity two and with com-tape restriction can. Then it is shown, that considering mappings from the class $\mathcal{F}_{\text{TRANS}}$, P automata is greater or equal in terms of computational power to genPCol automata using com-tape restriction with capacity $k,\ k \geq 1$.

The last chapter deals with an application of genPCol automata. We have shown, that considering deterministic parsing, genPCol automata can be defined in a way that work in LL(k) mode. It is then shown, that even with a lookahead of one, genPCol automata can characterize languages that cannot be accepted by any context free LL(k) grammars ($k \geq 1$).

Regarding the dissertation, we provided constructive proofs for most of the statements. The proofs were based on the publications of the author and his supervisor [21, 22, 23, 24, 25] and [10].

7 Összefoglalás

Az elmúlt évtizedben rengeteg nem hagyományos számítási modell született, melyek döntő többségét a biológia és a kémia ihlette. Ezen számítási modellek a biológusok és a kémikusok által megfigyelt, élő rendszerekben előforduló molekulákat, sejteket és folyamatokat írják le. A membránrendszerek egy tipikus megvalósulása a kémiai paradigmának, melyek elméleti megközelítést biztosítanak az élő sejt struktúrája és működése által motivált számítási modell kialakítására. Az értekezésben egy speciális membránrendszert, nevezetesen a generalizált P-kolónia automatát mutattuk be, melyet alaposan megvizsgáltunk és összehasonlítottunk más klasszikus és nemklasszikus számítási modellekkel számítási erő tekintetében.

A 3. fejezetben a modellel kapcsolatos definíciókat ismertettük. Meghatároztuk a generalizált P-kolónia automaták által elfogadott input szekvenciák halmazát és ismertettük, hogy miként tud nyelveket felismerni az automata. Három főbb működési módot írtunk le, melyek

a szabályok, illetve a belőlük épülő programok alakját és viselkedését szabják meg: a *megszorításmentes* altípust, a *com-tape* esetet, ahol nem fordulnak elő kommunikációs szabályok nem szalagos változatai, és az *all-tape* altípust, ahol minden programnak legalább egy szalagszabályt kell tartalmaznia. Végül konstruáltunk egy genPCol automatát, szemléltetve a működés alapgondolatát.

A 4. fejezetben a genPCol automata számítási erejére vonatkozó eredményeket ismertettük. Az eredményeket az elfogadott input multihalmaz sorozatok és karakterizált nyelvek közötti leképzések alapján csoportosítottuk. Először a permutációs leképezéseket vettük figyelembe. Megmutattuk, hogy megszorításmentes altípusú generalizált P kolónia automaták pontosan a rekurzívan felsorolható nyelvek osztályát karakterizálják, még akkor is, ha a rendszer kapacitását redukáljuk egyre. A bizonyítást a bemeneti szalaggal rendelkező regisztergépek szimulálásával készítettük el, amelyről ismert, hogy az általa elfogadott nyelvek osztálya a rekurzívan felsorolható nyelvek osztályával ekvivalens. Továbbá megmutattuk, hogy a com-tape és all-tape altípusokkal nemreguláris nyelveket is el lehet fogadni.

Egynél nagyobb kapacitású rendszereket vizsgálva érdekes eredményeket mutattunk ki. A com-tape altípust felülről korlátoztuk egy speciális, logaritmikus tárral rendelkező Turing géppel, mely input szalagján csak balról jobbra haladhat és a már elolvasott input hosszának logaritmikus függvénye alapján alakulhat a tárhely, amivel a gép rendelke-

zik. A bizonyításban megmutattuk, hogy a már beolvasott multihalmaz szekvenciák hosszától pontosan logaritmikusan függ a környezetben tárolható konfigurációk maximális darabszáma. Az all-tape altípusra ez a tétel akkor és csak akkor mondható ki, ha kizárjuk a törlő szalagszabályok ($a \xrightarrow{T} e$ alakú) meglétét; egyéb esetben legalább kettő kapacitással a rekurzívan felsorolható nyelvek osztályát kapjuk.

Tovább haladva, egy sokkal általánosabb multihalmaz leképzést vettünk figyelembe, név szerint a $\mathcal{F}_{\text{TRANS}}$ függvényosztályba tartozó multihalmaz leképzésekkel foglalkoztunk, amelyek véges transducerek alkalmazásával valósítják meg a multihalmazokhoz leképzendő karakterláncokat. Hasonló módon a permutációs leképzésekkel működő rendszerekhez, itt is meg tudtuk mutatni, hogy a rekurzívan felsorolható nyelvek osztályát írják le megszorításmentes verziók már egy kapacitással. A következő eredményben leírtuk, hogy a korlátozott szabályrendszerrel rendelkező rendszerek számítási ereje szigorúan nagyobb, mint a reguláris nyelveké. Bebizonyítottuk, hogy képesek véges automatákat szimulálni és nem-reguláris nyelveket elfogadni.

A fejezet utolsó részében összehasonlítottuk a genPCol automatákat a P automatákkal. Megmutattuk, hogy permutációs leképezés esetén vannak olyan nyelvek, amelyeket a P automaták nem tudnak elfogadni, de a com-tape altípusú genPCol automaták kettő kapacitással igen. Majd vizsgálva a $\mathcal{F}_{\text{TRANS}}$ függvényosztályba tartozó multihalmaz leképzésekkel operáló genPCol automatákat, kiderül, hogy a P automaták erőseb-

bek vagy egyenlőek.

Az utolsó fejezetben egy lehetséges alkalmazását mutatjuk be genPCol automatáknak. Itt definiáltuk, hogy mit értünk LL(k) módban működő genPCol automata alatt, majd megmutattuk, hogy pusztán egy darab objektum előre látása esetén el tudunk fogadni olyan nyelveket is, melyeket bármennyi objektum előre látásával sem tudunk környezetfüggetlen LL(k) esetben.

Az értekezésben kifejtett állításokhoz és tételekhez általában konstruktív bizonyításokat adtunk. A bizonyítások a szerző és témavezetője által megjelent publikációk alapján kerültek a disszertációba: [21, 22, 23, 24, 25] és [10].

8 Publications of Kristóf Kántor

International journal publications

1. K. Kántor, Gy. Vaszil: Generalized P colony automata. *Journal of Automata, Languages and Combinatorics* 19(1-4), 145–156 (2014)

2. K. Kántor, Gy. aszil: On the classes of languages characterized by generalized P colony automata. *Theoretical Computer Science* 724, 35–44 (2018)

International conference proceedings

3. E. Csuhaj-Varjú, K. Kántor, Gy. Vaszil: Deterministic Parsing with P Colony Automata. In: Graciani C., Riscos-Núñez A., Păun G., Rozenberg G., Salomaa A. (eds.), *Enjoying Natural Comput-*

ing. Volume 11270 of *Lecture Notes in Computer Science*, Springer, Cham, 88–98 (2018)

4. K. Kántor, Gy. Vaszil: On the classes of languages characterized by generalized P colony automata. In: *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 231–246. ISBN: 978-84-946316-1-0 (2016)

5. K. Kántor, Gy. Vaszil: Variants of P Colony Automata. In: *Ninth Workshop on Non-Classical Models of Automata and Applications (NCMA 2017), Short Papers*, 17–24 (2017)

6. K. Kántor, Gy. Vaszil: Generalized P colony automata and their relation to P automata. In: *18th International Conference on Membrane Computing, CMC 2017, Bradford, UK, July 25-28, 2017, Revised Selected Papers*. Volume 10725 of *Lecture Notes in Computer Science*, Springer, 167–182 (2017)

9 Bibliography

[1] A.V. Aho, J.D. Ulmann: *The Theory of Parsing, Translation, and Compiling,* vol. 1. Prentice-Hall, Englewood Cliffs, N.J. (1973)

[2] L. Cienciala, L. Ciencialová: P colonies and their extensions. In: J. Kelemen, A. Kelemenová, eds., *Computation, Cooperation, and Life. Essays Dedicated to Gheorghe Păun on the Occasion of His 60th Birthday*. LNCS 6610 (Springer Berlin Heidelberg, 2011) 158–169.

[3] L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú: Towards P colonies processing strings. In: Macías Ramos, L.F., Martínez Del Amor, M.A., Păun, G., Pérez Hurtado, I., Riscos Nuñez, A., Valencia Cabrera, L. (eds.) *Twelfth Brainstorming Week on Membrane Computing*, 103–118. Fénix Editora (2014)

[4] L. Cienciala, L. Ciencialová, E. Csuhaj-Varjú, G. Vaszil: Pcol automata: Recognizing strings with P colonies. In: Martínez Del Amor, M.A., Păun, G., Pérez Hurtado, I., Riscos Nuñez, A. (eds.) *Eighth Brainstorming Week on Membrane Computing*, Sevilla,

February 1-5, 2010, 65–76. Fénix Editora (2010)

[5] L. Cienciala, L. Ciencialová, A. Kelemenová: Homogeneous P colonies, *Computing and Informatics* 27 (2008) 481–496.

[6] L. Cienciala, L. Ciencialová, A. Kelemenová: On the number of agents in P colonies. In: G. Eleftherakis et. al, eds., *Membrane Computing. 8th International Workshop, WMC 2007. Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers, LNCS 4860* (Springer-Verlag, Berlin-Heidelberg, 2007) 193–208.

[7] L. Ciencialová, L. Cienciala: Variation on the theme: P colonies. In: D. Kolář, A. Meduna, eds., *Proc. 1st International Workshop on Formal Models* (Ostrava, 2006) 27–34.

[8] L. Ciencialová, E. Csuhaj-Varjú, A. Kelemenová, Gy. Vaszil: Variants of P colonies with very simple cell structure, *International Journal of Computers, Communication and Control* 4(3) (2009) 224–233.

[9] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun: *Grammar Systems – A Grammatical Approach to Distribution and Cooperation*, Gordon and Breach, London, 1994.

[10] E. Csuhaj-Varjú, K. Kántor, Gy. Vaszil: Deterministic Parsing with P Colony Automata. In: Graciani C., Riscos-Núñez A., Păun G., Rozenberg G., Salomaa A. (eds) *Enjoying Natural Computing. Lecture Notes in Computer Science, vol 11270*, 88–98. Springer, Cham (2018)

[11] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, Gy. Vaszil: Computing with cells in environment: P colonies, *Journal of Multi-Valued Logic and Soft Computing* 12 (2006) 201–215.

[12] E. Csuhaj-Varjú, M. Margenstern, Gy. Vaszil: P colonies with a bounded number of cells and programs. In: H-J. Hoogeboom et. al, eds., *Membrane Computing. 7th International Worskhop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006. Revised, Selected and Invited Papers. LNCS 4361* (Springer-Verlag, Berlin-Heidelberg, 2007) 352–366.

[13] E. Csuhaj-Varjú, O.H. Ibarra, Gy. Vaszil: On the computational complexity of P automata. In: C. Ferretti, G. Mauri, C. Zandron, eds.,*10th International Workshop on DNA Computing, LNCS 3384*, Springer Berlin Heidelberg, 2005, 76–89.

[14] E. Csuhaj-Varjú, M. Oswald, Gy. Vaszil: P automata, in [36], chapter 6, 144–167.

[15] E. Csuhaj-Varjú, Gy. Vaszil: P automata or purely communicating accepting P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds., *Membrane Computing. International Worskhop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers. LNCS 2597*, Springer Berlin Heidelberg, 2003, 219–233.

[16] E. Csuhaj-Varjú, Gy. Vaszil, P automata with restricted power, *International Journal of Foundations of Computer Science* **2**5 (2014)

391–408.

[17] P.C. Fischer: Turing machines with restricted memory access, *Information and Control* 9 (1966) 364–379.

[18] R. Freund, M. Kogler, Gh. Păun, M.J. Pérez-Jiménez: On the power of P and dP automata, *Annals of Bucharest University, Mathematics-Informatics Series* 63 (2009) 5-22.

[19] R. Freund, M. Oswald: P colonies and prescribed teams, *International Journal of Computer Mathematics* 83 (2006) 569–502.

[20] R. Freund, M. Oswald: P colonies working in the maximally parallel and in the sequential mode. In: G. Ciobanu, Gh. Păun, eds., *1st Intern. Workshop on Theory and Application of P Systems*, Timisoara, Romania, 2005, 49–56.

[21] K. Kántor, Gy.Vaszil: Generalized P colony automata. *Journal of Automata, Languages and Combinatorics* 19(1-4), 145–156 (2014)

[22] K. Kántor, Gy. Vaszil: On the classes of languages characterized by generalized P colony automata. *Theoretical Computer Science* 724, 35–44 (2018)

[23] K. Kántor, Gy. Vaszil: On the classes of languages characterized by generalized P colony automata. In: *Proceedings of the Fourteenth Brainstorming Week on Membrane Computing*, 231-246. ISBN: 978-84-946316-1-0 (2016)

[24] K. Kántor, Gy. Vaszil: Variants of P Colony Automata. In: *Ninth*

Workshop on Non-Classical Models of Automata and Applications (NCMA 2017), Short Papers, 17-24 (2017)

[25] K. Kántor, Gy. Vaszil: Generalized P colony automata and their relation to P automata. In: *18th International Conference on Membrane Computing, CMC 2017, Bradford, UK, July 25-28, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10725*, 167–182. Springer (2017)

[26] J. Kelemen, A. Kelemenová: A grammar-theoretic treatment of multiagent systems, *Cybernetics and Systems* 23 (1992) 621–633.

[27] J. Kelemen, A. Kelemenová, Gh. Păun, Preview of P colonies: A biochemically inspired computing model, in: M. Bedau et al., eds., *Worksop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)* (Boston Mass., 2004) 82–86.

[28] A. Kelemenová: P colonies. In: Paun, G., Rozenberg, G., Salomaa, A. (eds.) *The Oxford Handbook of Membrane Computing*, 584–593. Oxford University Press, Inc. (2010)

[29] D.E. Knuth: On the translation of languages from left to right. *Information and Control*, 8 (1965), 607-639

[30] M. Kutrib, J. Provillard, Gy. Vaszil, M. Wendlandt: Deterministic One-Way Turing Machines with Sublinear Space. *Fundamenta Informaticae* 136, 195-208 (2013)

[31] P.M. Lewis III, R.E. Stearns: Syntax-directed transductions *J. Assoc. Comput. Mach.*, 15 (1968), 465–488

[32] M.L. Minsky, *Computation: Finite and Infinite Machines*, Prentice Hall, 1967.

[33] A. Păun, G. Păun: The power of communication: P systems with symport/antiport. *New Generation Comput.* 20(3), 295–306 (2002)

[34] G. Păun: From cells to computers: Computing with membranes (P systems). *Biosystems* 59., 2001, 139–158.

[35] G Păun: Computing with membranes. *Journal of Computer and System Sciences* 61(1), 2000, 108-143

[36] G. Păun, G. Rozenberg, A. Salomaa, eds., *The Oxford Handbook of Membrane Computing* (Oxford University Press, 2010).