Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava

Biologically inspired computation models

(Dissertation thesis)

Mgr. Michal Kováč

Advisor: doc. RNDr. Damas Gruska, PhD.

Bratislava, 2015

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Mgr. Michal Kováč
Študijný program: informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: dizertačná
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Biologicky inšpirované výpočtové modely

Školiteľ: doc. RNDr. Damas Gruska, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 01.09.2011

Dátum schválenia: 14.04.2011 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
školiteľ

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

. . . . . .

Acknowledgments

I am very grateful to all the people I had the privilege to work with, especially I would like to thank my advisor RNDr. Damas Gruska PhD. for his constant interest, encouragement and help that I deeply appreciate. The present work was possible due to many beneficial consultations and intensive cooperation.

The major thanks also goes to those who provided valuable feedback to my work, specifically classmates Mgr. Martin Králik and Mgr. Andrej Borsuk and the editors and referees of conferences I have contributed to: Hybrid Systems and Biology 2013, Computational Models for Cell Processes 2013 and Computability in Europe 2014, 2015.

I would also like to thank to the members of Laboratory of Comparative and Functional Genomics of Eukaryotic Organelles and their head, Prof. Ľubomír Tomáška for familiarization with the work of biologists.

Last, but not least, I thank to my family, colleagues, teammates and all the friends who supported me especially last weeks before the deadlines for their patience, support and understanding.

Abstract

Author: Mgr. Michal Kováč
Title: Biologically inspired computation models
University: Comenius University in Bratislava
Faculty: Faculty of Mathematics, Physics and Informatics
Department: Department of Applied Informatics
Supervisor: doc. RNDr. Damas Gruska, PhD.

This work discusses the research in the membrane systems, an emerging field of natural computing. Many variants of membranes systems have already been studied, most of them uses parallel rewriting and are computationally complete. Various sequential models have been proposed, however, in many cases they are weaker than their parallel variant. We investigated several variants in terms of computational power and decidability of behavioral properties. We have also proposed own variants and suggested many topics for future study.

Keywords: Computation models inspired by biology, Membrane systems, Sequential P systems, Maximal parallelism, Computational power

Abstrakt

Autor: Mgr. Michal Kováč
Názov dizertačnej práce: Biologically inspired computation models
Škola: Univerzita Komenského v Bratislave
Fakulta: Fakulta matematiky, fyziky a informatiky
Katedra: Katedra aplikovanej informatiky
Vedúci dizertačnej práce: doc. RNDr. Damas Gruska, PhD.
Bratislava, 2015

V tejto práci sa zaoberáme výskumom v oblasti membránových systémov. Veľa variantov už bolo preskúmaných, väčšinou pri výpočte používajú paralelizmus a sú Turingovsky úplné. Navrhlo sa aj veľa sekvenčných modelov, ale väčšina z nich má slabšiu výpočtovú silu ako ich paralelný variant. Preskúmali sme niekoľko variantov z hľadiska výpočtovej sily, ako aj z hľadiska rozhodnuteľnosti behaviorálnych vlastností. Navrhli sme aj vlastné varianty a naznačili smery, akými sa môže rozvíjať tento výskum.

Kľúčové slová: Výpočtové modely inšpirované biológiou, Membránové systémy, Sekvenčné P systémy, Maximálny paralelizmus, Výpočtová sila

Contents

List of Figures

List of Algorithms

Introduction

There are a lot of areas in the theoretical computer science that are motivated by other science fields. Computation models motivated by biology forms a large group of them. They include neural networks, computational models based on DNA evolutionary algorithms, which have already found their use in computer science and proved that it is worth to be inspired by biology. L-systems are specialized for describing the growth of plants, but they have also found the applications in computer graphics, especially in fractal geometry. Other emerging areas are still awaiting for their more significant uses.

One of them is the membrane computing [79]. It is relatively young field of natural computing - in comparison: neural networks have been researched since 1943 and membrane systems since 1998 [77].

Membrane systems (P systems) are distributed parallel computing devices inspired by the structure and functionality of cells. Recently, many P system variants have been developed in order to simulate the cells more realistically or just to improve the computational power.

We will start by an introduction of some natural computing areas including models inspired by biology in Chapter 1. In Chapter 2 we recall some computer science basic notions that we will use through the work. P systems are formally presented in Chapter 3, with the current state of the research in their variants, overview of software simulator MeCoSym and various case studies.

In Chapter 4 we will present the current state of our work, mainly from theoretic viewpoint (computational power, decidability of behavioral properties), including the published results in sections 4.1, 4.2 and 4.4.

Chapter 1

Natural computing

Recently, an interdisciplinary research between the fields of Computer Science and Biology has been rapidly growing. Natural computing consists of three classes of methods:

- Those that are based on the use of computers to simulate natural phenomena (Section 1.1).
- Those that employ natural materials (e.g., molecules) to compute (Section 1.2).
- Those that take inspiration from nature for the development of novel problem-solving techniques (Section 1.3).

1.1 Bioinformatics

Bioinformatics has undergone a fast evolving process, especially the areas of genomics and proteomics. Bioinformatics can be seen as the application of computing tools and techniques for the management of biological data. Just to mention a few:

- the design of efficient algorithms for DNA sequence alignment,
- the investigation of methods for prediction of the three-dimensional structure of molecules and proteins,
- the development of data structures to effectively store huge amount of structured data.

1.2 Biomolecular computing

Biomolecular computing make use of molecules such as DNA and proteins to perform computations involving storing, retrieving and processing data. It takes advantage of the many different molecules to try many different possibilities at one, so it is somewhat similar to parallel computing.

Adleman in his 1994 report [3] demonstrated a proof of concept use of DNA as a form of computation which solved the Hamiltonian path problem. Since then, various Turing machines have been proven to be constructible with DNA [54].

DNA can also be used as a digital data storage with 5.5 petabits per cubic millimeter of DNA (see [21]). The information retrieval is, however, a slow process, as the DNA needs to be sequenced in order to retrieve data, so this method is intended mainly for a long-term archival of large amounts of scientific data.

1.3 Biologically inspired computing models

The birth of biologically inspired frameworks started the investigation of complex natural phenomena in terms of computational processes and enhanced our understanding of both nature and the essence of computation. Those frameworks are inspired by the nature in the way it "computes", and has gone through the evolution for billions of years. Characteristic for human-designed computing inspired by nature is the metaphorical use of concepts, principles and mechanisms of underlying natural systems.

Neural networks, evolutionary algorithms and cellular automata are already well established research fields. More recent computational systems abstracted from natural processes include swarm intelligence and membrane computing only

1.3.1 Neural networks

Inspired by the human brain, which contains on average 86 billions neurons [8], neurophysiologist Warren McCulloch in 1943 proposed a mathematical model of artificial neural network.

A single perceptron computes a function $f(x)$, where x is an input - a vector of real values.

The perceptron can learn itself by modifying parameters used to compute the function. This learning can be performed in various ways, we will mention only the supervised learning. Imagine a function with

- input: perceptron's parameters
- output: error of the computed result $f(x)$

If the perceptron receives a feedback in form of error (from the supervision, e.g. dataset used to train the perceptron), it can modify its parameters such the error will be minimized in the future. This can be done through the gradient descent method, which is used to find a local minimum of a function.

A single perceptron can only compute linear functions, so they are often connected with other perceptron to form a neural network. Often it is practically unusable to say what is the purpose of a single neuron in a more complex neural networks.

Neural networks have broad applicability to real world problems and are best if the modeled system has some tolerance to error such as:

- Image recognition: OCR, web search by image
- Music recognition by voice sample
- Speech recognition
- Time series forecast: weather, stock
- Diagnosing of illnesses
- Natural language processing

Besides real world problems, Graves et al. [42] extended the capabilities of neural networks by coupling them to external memory resources, allowing them to infer simple algorithms such as copying and sorting.

1.3.2 Evolutionary algorithms

Evolutionary algorithms are inspired by Darwin's theory of evolution. Basically, an algorithm starts with a population of random individuals (solutions to the problem). In each generation, the fitness of every individual is evaluated and the more fit individuals are replicated and mutated to form a new generation. The less fit individuals die.

Idea of evolutionary computing was introduced in the 1960's by I. Rechenberg. Nowadays they have been applied to find solution of many optimization problems. They also can be used to design or to train a neural network [71].

1.3.3 L systems

In 1968, a Hungarian botanist and theoretical biologist Aristid Lindenmayer introduced a new string rewriting algorithm named Lindenmayer systems (or L-systems for short) [64, 91]. Inspired by the growth of plants, they have proven to be among the most beautiful examples of interdisciplinary science, where work in one area induces fruitful ideas and results in other areas:

- Computer graphics - simulated evolution of plants, where L-systems are used as genetic encoding. The phenotypes are the branching structures resulting from the derivation and graphic interpretation of the genotypes [74].
- Abstract framework for music composers to allow generation of musical structure [66].
- Generating nonlinear missions in PC games, where multiple possible paths can lead to the goal of the mission [101].

1.3.4 Cellular automaton

A cellular automaton (CA) is a discrete dynamical system that consists of an infinite array of cells. Each cell has a state from a finite state set. The cells change their states according to a local update rule that provides the new state based on the old states of the cell and its neighbors. All states use the same update rule, and the updating happens simultaneously at all cells. The updating is repeated over and over again at discrete time steps, leading to a time evolution of the system.

The first CA studies by John von Neumann in the late 1940s were biologically motivated, related to self-replication in universal systems [73]. John von Neumann considered cells to be arranged in a two-dimensional grid.

Widespread popular interest for cellular automaton was spread by Game of Life introduced by John Conway in 1970 [40]. Conway's criterion was that the rule should neither lead to populations which quickly died out, nor which quickly expanded without end from random finite initial configuration. The state of a cell in a two-dimensional grid is determined by the state of 8 neighbouring cells.

- A live cell survives if it has two or three live neighbors.
- A new cell is born whenever there are three live neighbors.
- All other cells either die or remain inactive.

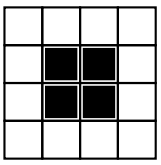


Figure 1.1: Block

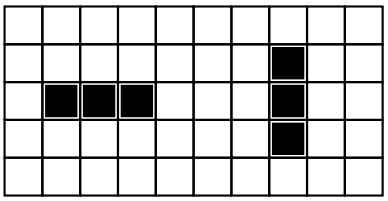


Figure 1.2: Blinker

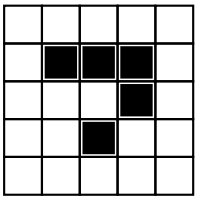


Figure 1.3: Glider

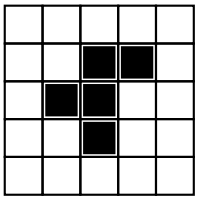


Figure 1.4: R-pentomino

These simple rules which resembles behavior of bacteria has quickly become popular and various real-world phenomena now have corresponding behavior with the same notion in Game of Life. Comprehensive glossary and new discoveries are summarized in the LifeWiki [1]. **Still life** is a configuration which does not change in time. The simplest still life is a **block**, which is a 2×2 square in figure 1.1 where each cell has exactly three neighbors. The feature of the game that probably caused intensive interest was undoubtedly the discovery of oscillators (periodic forms) and gliders (translating oscillators) **Alternators** are oscillators with period 2. The simplest oscillator is a **blinker** (figure 1.2), which is a structure consisting of three live cells alternating in form between three horizontal cells and three vertical cells. A **glider** (figure 1.3) is a pattern that travels across the board. It was discovered while attempting to track the evolution of the **R-pentomino** which is by far the most active polyomino with fewer than six cells. All of the others stabilize in at most 10 steps, but the R-pentomino does not do so until step 1103, by which time it has a population of 116 live cells. The glider it releases in generation 69, noticed by Richard K. Guy, was the first glider ever observed. Gliders are important because they can be used to transmit information over long distances and collided with each other to form more complicated objects. Gliders can be generated with a **glider gun**. It is possible to construct logic gates such as AND, OR and NOT using gliders. Rendell [87] even constructs a universal Turing machine.

Physicists, biologists, economists, mathematicians, philosophers, generative scientists and others are interested in Conway's Game of Life in order to observe the way that complex patterns can emerge from the implementation

of very simple rules.

Elementary one dimensional version of Game of Life known as Rule 110 is also capable of universal computation [24]. It is arguably the simplest known Turing complete system.

1.3.5 Swarm Intelligence

Swarm intelligence systems conists of a population of simple agents, which are following simple rules. They coordinate in decentralized manner and self-organize to achieve a common goal. The swarm intelligence focuses on the collective behaviors that result from the local interactions of the individuals with each other and with their environment. Examples of such systems are colonies of ants and termites, schools of fish, flocks of birds, herds of land animals and bees.

The solutions designed by nature can often be used in other real-world problems.

Ant colony optimization

Ant colony optimization introduced by Dorigo [31] is a class of optimization algorithms for finding better paths through graphs. Ants wander randomly to find some food and return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but instead to follow the trail, returning and reinforcing it if they eventually find food. Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. The influence of pheromone evaporation in real ant systems is unclear, but it is very important in artificial systems [32]. It has been used to produce near-optimal solutions to the travelling salesman problem. They have an advantage over simulated annealing and genetic algorithm approaches of similar problems when the graph may change dynamically, the ant colony algorithm can be run continuously and adapt to changes in real time. Among other real-world uses are the vehicle routing problem [90] and improving the efficiency of an electric motor in dry vacuum cleaner [58].

Bird flocking

In the natural world, organisms exhibit certain behaviors when traveling in groups. This phenomenon, also known as flocking, occurs at both microscopic scales (bacteria) and macroscopic scales (birds, fish, insects). Using computers, these patterns can be simulated by creating simple rules for interaction with neighbors and combining them to a complex collective behavior:

- Alignment is a behavior that causes a particular agent to line up with agents close by.
- Cohesion is a behavior that causes agents to steer towards the center of mass of neighbors - that is, the average position of the agents within a certain radius.
- Separation is the behavior that causes an agent to steer away from all of its neighbors to avoid clashes.

Reynolds in 1987 first used such rules in a simulation [88]. Solutions to several problems in other fields have been inspired by flocking. It has been applied to the problem of managing the flight of a number of autonomous unmanned air vehicles [26]. They also took into account aerodynamic features of the flock. Another usages are in films to generate crowds which move more realistically, or spatial representation of time-varying information [70].

Artificial bee colony

Artificial Bee Colony Algorithm is one of the most recent algorithms in the domain of the collective intelligence. It was created by Dervis Karaboga in 2005, who was motivated by the intelligent behavior observed in the domestic bees to take the process of foraging [53]. Although, the performance of different optimization algorithm is dependent on applications, some recent works demonstrate that the artificial bee colony is more rapid than either genetic algorithm or particle swarm optimization solving certain problems, especially those with a lot of variables (high-dimensional problems), e.g. protein folding [63] and magnetic resonance brain image classification [102].

1.3.6 Membrane systems

Nature computes not only at the neural or genetic level, but also at the cellular level. In general, any non-trivial biological system has a hierarchical structure where objects and information flows between regions, what can be interpreted as a computation process.

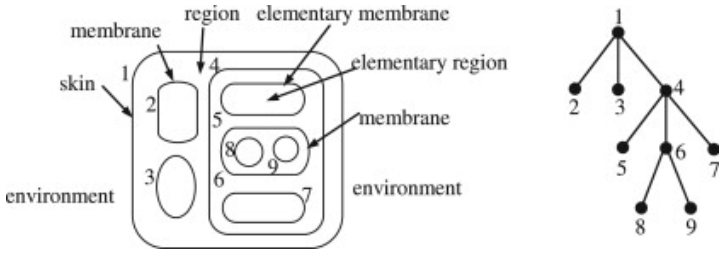


Figure 1.5: The membrane structure of a P system and its associated tree [104]

The regions are typically delimited by various types of membranes at different levels from cell membranes, through skin membrane to virtual membranes which delimits different parts of an ecosystem. This hierarchical system can be seen in other field such as distributed computing, where again well delimited computing units coexist and are hierarchically arranged in complex systems from single processors to the internet.

Membranes keep together certain chemicals or information and selectively determines which of them may pass through.

From these observations, Păun [77] introduces the notion of a membrane structure as a mathematical representation of hierarchical architectures composed of membranes. It is usually represented as a Venn diagram with all the considered sets being subsets of a unique set and not allowed to be intersected. Every two sets are either one the subset of the other, or disjoint. Outermost membrane (also called skin membrane) delimits the finite "inside" and the infinite "outside".

Hierarchical structures are usually represented by trees, but membrane structures are visually better viewed as Venn diagram as seen in the figure 1.5.

Recently, several computational models based on the membrane structure have been created.

The Calculus of Looping Sequences

In the last few years many formalisms originally developed by computer scientists to model systems of interacting components have been applied to biology. Here, we can mention Petri nets (see section 2.9). Others, such as P systems (see Chapter 3), have been proposed as biologically inspired computational models and have been later applied to the description of biological systems.

Many of these models either offer only very low-level interaction primitives or they are specialized to the description of some particular kinds of

phenomena such as membrane interactions or protein interactions making the model be not so flexible to allow describing easily new activities observed on membranes without extending the formalism to model such activities.

Barbuti in [11] concluded that there is a need for a formalism having a simple notation, having the ability of describing biological systems at different levels of abstractions, having some notions of compositionality and being flexible enough to allow describing new kinds of phenomena as they are discovered, without being specialized to the description of a particular class of systems. The Calculus of Looping Sequences (CLS) was introduced in [11].

A CLS model consists of a term and a set of rewriting rules. The term is intended to represent the structure of the modeled system and the rewriting rules to represent the events that may cause the system to evolve.

The membrane structure in CLS is defined recursively, consisting of terms and sequences.

We start with defining the syntax of terms. We assume a possibly infinite alphabet Σ of symbols.

Definition 1.3.1. *Terms T and sequences S of CLS are given by the following grammar:*

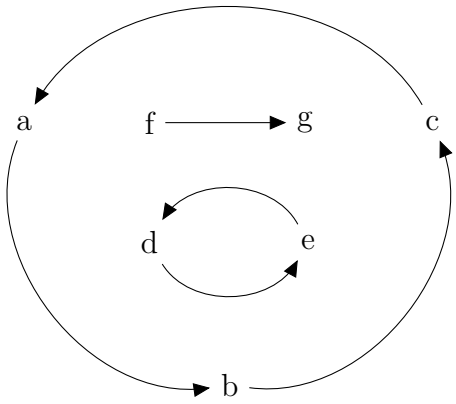
$$T ::= S \;\Big|\; (S)^L \rfloor T \;\Big|\; T|T$$
$$S ::= \varepsilon \;\Big|\; a \;\Big|\; S \cdot S$$

where $a \in \Sigma$ and ε represents the empty sequence. We denote with $\mathcal{T}$ the infinite set of terms and with $\mathcal{S}$ the infinite set of sequences.

In CLS we have a sequencing operator $\_ \cdot \_$, a looping operator $(\_)^L$, a parallel composition operator $\_|\_$ and a containment operator $\_\rfloor\_$. Sequencing can be used to concatenate elements of the alphabet Σ. The empty sequence ε denotes the concatenation of zero symbols. A term can be either a sequence or a looping sequence (that is the application of the looping operator to a sequence) containing another term, or the parallel composition of two terms. By definition, looping and containment are always applied together, hence we can consider them as a single binary operator $(\_)^L\rfloor\_$ which applies to one sequence and one term.

Example 1.3.1. In the Figure 1.6 we show an example of CLS and its visual representation. The same structure may be represented by syntactically different terms, e.g. $(b \cdot c \cdot a)^L\rfloor(f \cdot g|(e \cdot d)^L)$. We introduce a structural congruence relation to identify such terms.

Definition 1.3.2. *The structural congruence relations $\equiv_S$ and $\equiv_T$ are the least congruence relations on sequences and on terms, respectively, satisfying the following rules:*



A representation of the term $(a \cdot b \cdot c)^L \rfloor ((d \cdot e)^L | f \cdot g)$.

Figure 1.6: Example CLS

- $S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3$
- $S \cdot \varepsilon \equiv_S \varepsilon \cdot S \equiv_S S$
- $S_1 \equiv_S S_2$ *implies* $S_1 \equiv_T S_2$ *and* $(S_1)^L \rfloor T \equiv_T (S_2)^L \rfloor T$
- $T_1 | T_2 \equiv_T T_2 | T_1$
- $T_1 | (T_2 | T_3) \equiv_T (T_1 | T_2) | T_3$
- $T | \varepsilon \equiv_T T$
- $(\varepsilon)^L \rfloor \varepsilon \equiv_T \varepsilon$
- $(S_1 \cdot S_2)^L \rfloor T \equiv_T (S_2 \cdot S_1)^L \rfloor T$

Note that the last rule does not introduce the commutativity of sequences, but only says that looping sequences can rotate.

What could look strange in CLS is the use of looping sequences for the description of membranes, as sequencing is not a commutative operation and this does not correspond to the usual fluid representation of membrane surface in which objects can move freely. What one would expect is to have a multiset or a parallel composition of objects on a membrane. For this reason, a variant called CLS+ was introduced in [68], in which the looping operator can be applied to a parallel composition of sequences.

Definition 1.3.3. *Terms T, branes B and sequences S of CLS+ are given by the following grammar:*

$$T ::= S \mid (B)^L \rfloor T \mid T|T$$
$$B ::= S \mid B|B$$
$$S ::= \varepsilon \mid a \mid S \cdot S$$

The structural congruence relation of CLS+ is a trivial extension of the one of CLS. The only difference is that commutativity of branes replaces rotation of looping sequences. CLS+ models can be translated into CLS models, while preserving the semantics of the model [11]. Milazzo in his PhD thesis [68] includes also a simulation of a P system using CLS. The major difficulty was the simulation of the maximal parallel rule application.

P systems

P systems [77] were introduced in 1998 as a system with membrane structure that contains multisets of objects and rewriting rules that are executed in a maximally parallel manner. Since then, a huge amount of variants have been created with various computation powers. We have investigated the computational power and other attributes of several such variants. Additionally, we propose a new variants various notions of emptyness detection, where the rules can describe what will happen to an object that arrives to an empty membrane.

P systems with existing and newly proposed variants are more in depth discussed in Chapter 3.

Chapter 2

Preliminaries

This Chapter is about some basic notions of computer science which will be used through the work. We start by defining formal languages and basic models (grammars, machines) that define language families. We introduce some behaviors of the models and methods to analyze them.

2.1 Formal languages

Our study is based on the classical theory of formal languages. We will recall some definitions:

Definition 2.1.1. *An* **alphabet** *is a finite nonempty set of symbols.*

Definition 2.1.2. *A* **string** *over an alphabet is a finite sequence of symbols from the alphabet.*

The length of the string s is denoted by $|s|$. We denote by ε a string with zero length. It is also called an empty string. We denote by Σ^* the set of all strings over an alphabet Σ. By $\Sigma^+ = \Sigma^* - \{\varepsilon\}$ we denote the set of all nonempty strings over Σ.

Definition 2.1.3. *A* **language** *over the alphabet Σ is any subset of Σ^*.*

Definition 2.1.4. *A* **family of languages** *is a set of languages.*

2.2 Formal grammars

We are now ready to define the basic type of computation model inspired by the study of real languages - formal grammars. Realizing the parts of sentences could be labeled to indicate their function, it seems natural that

one might notice that there appear to be rules dictating how these parts of speech are used.

Definition 2.2.1. *A* **formal grammar** *is a tuple* $G = (N, T, P, \sigma)$*, where*

- N, T *are disjoint alphabets of non-terminal and terminal symbols,*
- $\sigma \in N$ *is the initial non-terminal,*
- P *is a finite set of rewriting rules of the form* $u \to v$*, with* $u \in (N \cup T)^* N (N \cup T)^*$ *and* $v \in (N \cup T)^*$*.*

Definition 2.2.2. *A* **rewriting step** *in the grammar* G *is a binary relation* $\Rightarrow$ *on* $(N \cup T)^*$*, where* $x \Rightarrow y$ *iff* $\exists w_1, w_2 \in (N \cup T)^*$ *and a rule* $u \to v \in P$ *such that* $x = w_1 u w_2$ *and* $y = w_1 v w_2$*.*

Definition 2.2.3. *Language defined by a grammar* G *is a set* $L(G) = \{w \in T^* | \sigma \Rightarrow w\}$*.*

Set of languages that can be defined by a formal grammar are called recursively enumerable languages RE.

2.3 Chomsky hierarchy

In this section we introduce several well-known families of languages, introduced by Chomsky [20].

Definition 2.3.1. *A* **regular grammar** *is a formal grammar, where the rewriting rules are of the form* $u \to v$*, where* $u \in N$ *and* $v \in T^*(N \cup \{\varepsilon\})$*.*

Definition 2.3.2. *A* **regular language** *is a language generated by a regular grammar. The family of regular languages is denoted* R*.*

Definition 2.3.3. *A* **context-free grammar** *is a formal grammar, where rewriting rules are of the form* $u \to v$*, where* $u \in N$ *and* $v \in (N \cup T)^*$*.*

Definition 2.3.4. *A* **context-free language** *is a language generated by a context-free grammar. The family of context-free languages is denoted* CF*.*

Definition 2.3.5. *A* **context-sensitive grammar** *is a formal grammar, where rewriting rules are of the form* $u \to v$*, where* $u \in (N \cup T)^* N (N \cup T)^*$*,* $v \in (N \cup T)^*$ *and* $|u| < |v|$*.*

Definition 2.3.6. *A* **context-sensitive language** *is a language generated by a context-sensitive grammar. The family of context-sensitive languages is denoted* CS*.*

These families of languages forms the Chomsky hierarchy by means of inclusions: $R \subset CF \subset CS \subset RE$.

2.4 Matrix grammars

Context-free grammars are well-studied and well-behaved in terms of decidability, but many real-world problems cannot be described with context-free grammars. Grammars with regulated rewriting are grammars with mechanisms to regulate the application of rules, so that certain derivations are avoided. They often have a higher generative capacity than context-free grammars.

Matrix grammars ([25]), as introduced by Abraham (1965) fall into this category. Rewriting rules cannot be applied separately, but they are grouped together into finite sequences, and they must be applied in sequence.

Definition 2.4.1. *Let $G = (N, T, P, \sigma)$ be a context-free grammar. Let M be a finite set of matrices, which are sequences of context-free rules from P. These sequences are called matrices and the pair $G_m = (G, M)$ is called a context-free* **matrix grammar**.

Definition 2.4.2. *A* **rewriting step** *$x \Rightarrow y$ holds only if there is a matrix $(u_1 \to v_1, u_2 \to v_2, \ldots, u_n \to v_n) \in M$ such that for each $1 \leq i \leq n$ the following holds: $x_i = x'_i u_i x''_i$ and $x_{i+1} = x'_i v_i x''_i$, where $x_i, x'_i, x''_i \in (N \cup T)^*$ and $x_1 = x$ and $x_{n+1} = y$.*

Example 2.4.1. Consider the following matrix grammar:

$$G_m = ((\{\sigma, X, Y\}, \{a, b, c\}, P, \sigma), M),$$

where P contains rules

$$\sigma \to XY, X \to aXb, Y \to cY, X \to ab, Y \to c$$

and M contains three matrices:

$$[\sigma \to XY], [X \to aXb, Y \to cY], [X \to ab, Y \to c].$$

The following example computation of G_m uses the first matrix, then three times the second matrix and finally the last matrix:

$$\sigma \Rightarrow XY \Rightarrow aXbcY \Rightarrow aaXbbccY \Rightarrow aaaXbbbcccY \Rightarrow aaabbbccc$$

Although there are only context-free rules, yet the grammar generate the context-sensitive language $\{a^n b^n c^n | n \geq 1\}$.

The family of matrix grammars is denoted MAT. It is known that $CF \subset MAT \subset RE$. Interestingly, $MAT \cap a^* \subset R$ (see [12]).

2.5 Lindenmayer systems

Lindermeyer system (or L-system for short) [64] is a parallel string rewriting algorithm. The simplest version of L-systems assumes that the development of a cell is free of influence of other cells. This type of L-systems is called $0L$ systems, where "0" stands for zero-sided communication between cells.

Definition 2.5.1. *A $0L$ system is a triple (Σ, P, ω), where Σ is an alphabet, ω is a word over Σ and P is a finite set of rewriting rules of the form $a \to x$, where $a \in \Sigma, x \in \Sigma^*$.*

It is assumed there is at least one rewriting rule for each letter of Σ. $0L$ system works in parallel way, so all the symbols are rewritten in each step.

Example 2.5.1. Consider a $0L$ system with alphabet $\Sigma = \{a, b\}$, initial word $\omega = a$ and rewriting rules $P = \{a \to b, b \to ab\}$. Since in this system there is exactly one rule for every letter of the alphabet, the rewriting is thus deterministic and the generated words will be $\{a, b, ab, bab, abbab, \ldots\}$.

L-systems with tables (T) have several sets of rewriting rules instead of just one set. At one step of the rewriting process, rules belonging to the same set have to be applied. The biological motivation for introducing tables is that one may want different rules to take care of different environmental conditions (heat, light, etc.) or of different stages of development.

Definition 2.5.2. *An extended ($E0L$) system is a pair $G_1 = (G, \Sigma_T)$, where $G = (\Sigma, P, \omega)$ is an $0L$ system, where $\Sigma_T \subseteq \Sigma$, referred to as the terminal alphabet. The language generated by G_1 is defined by $L(G_1) = L(G) \cap \Sigma_T^*$.*

Such languages are called $E0L$ languages. $E0L$ languages with tables are called $ET0L$ languages.

It is known that $CF \subset E0L \subset ET0L \subset CS$ (see section 2.3 for definitions of CF and CS).

2.6 Multisets

Definition 2.6.1. *A multiset over a set X is a mapping $M : X \to \mathbb{N}$.*

We denote by $M(x), x \in X$ the multiplicity of x in the multiset M.

Definition 2.6.2. *The* **support** *of a multiset M is the set $supp(M) = \{x \in X | M(x) \geq 1\}$.*

It is the set of items with at least one occurrence.

Definition 2.6.3. *A multiset is* **empty** *when its support is empty.*

A multiset M with finite support $X = \{x_1, x_2, \ldots, x_n\}$ can be represented by the string $x_1^{M(x_1)} x_2^{M(x_2)} \ldots x_n^{M(x_n)}$. As elements of a multiset can also be strings, we separate them with the pipe symbol, e.g.

$$element|element|other\_element.$$

Definition 2.6.4. *Multiset inclusion. We say that multiset M_1 is included in multiset M_2 if $\forall x \in X : M_1(x) \leq M_2(x)$. We denote it by $M_1 \subseteq M_2$.*

Definition 2.6.5. *The* **union** *of two multisets $M_1 \cup M_2$ is a multiset where $\forall x \in X : (M_1 \cup M_2)(x) = M_1(x) + M_2(x)$.*

Definition 2.6.6. *The* **difference** *of two multisets $M_1 - M_2$ is a multiset where $\forall x \in X : (M_1 - M_2)(x) = \max(M_2(x) - M_1(x), 0)$.*

Definition 2.6.7. *Product of multiset M with natural number $n \in \mathbb{N}$ is a multiset where $\forall x \in X : (n \cdot M)(x) = n \cdot M(x)$.*

2.7 Semilinear sets

Some results in Chapter 3 refer to semilinear sets [49], so we will define them in this section.

Definition 2.7.1. *A* **linear set** *$L(v_0, v_1, \ldots, v_r)$ is a subset $\{v_0 + \sum_{i=1}^{r} a_i v_i | a_i \in \mathbb{N}\}$ of $\mathbb{N}^n$, where $v_0, v_1, \ldots, v_r \in \mathbb{N}^n$.*

We call v_0 the initial vector and $v_1, \ldots, v_r$ the generator vectors of the linear set. Basically, a linear set is a set of n-dimensional vectors given by integral linear combinations (with integer coefficients) of generator vectors added to the initial vector.

Example 2.7.1. For $n = 1$ the linear set is a subset of $\mathbb{N}$. For $n = 1$ and $r = 1$ we get an arithmetic progression.

Example 2.7.2. $L((0,0),(0,1),(1,0))$ defines all pairs of natural numbers.

Definition 2.7.2. *A subset of $\mathbb{N}^n$ is called* **semilinear** *if it is a finite union of linear sets.*

2.7.1 Parikh's mapping

In this subsection we show how semilinear sets and multisets relate to the formal language theory. We will start with several basic definitions.

The number of occurrences of a given symbol $a \in \Sigma$ in the string $w \in \Sigma^*$ is denoted by $|w|_a$.

Definition 2.7.3. $\Psi_\Sigma(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$ *is called a* **Parikh image of the string** $w \in \Sigma^*$, *where* $\Sigma = \{a_1, a_2, \ldots a_n\}$.

When referring to the Parikh mapping for a language, this should be taken to mean the mapping applied to all the words on the language. This idea is expressed in the next definition.

Definition 2.7.4. *For a language* $L \subseteq \Sigma^*$, $\Psi_\Sigma(L) = \{\Psi_\Sigma(w) | w \in L\}$ *is the* **Parikh image of the language** L.

Definition 2.7.5. *If* FL *is a family of languages, by* $PsFL$ *we denote the family of Parikh images of languages in* FL, *e.g.* $PsCF, PsRE$.

Example 2.7.3. Consider an alphabet $V = \{a, b\}$ and a language $L = \{a, ab, ba\}$. $\Psi_\Sigma(L) = \{(1, 0), (1, 1)\}$. Notice that Parikh image of L has only 2 element while L has 3 elements.

Example 2.7.4. Consider the context-free grammar $G = (N, T, P, \sigma)$, where $N = \{A, B\}$, $T = \{a, b\}$ and with rules $P = \{\sigma \to A\sigma B | B\sigma A | ab, A \to a, B \to b\}$.

This means that the language generated by G contains strings where as and bs can occur intermixed. But, as is clear from the language definition, the same number of as and bs will always be present. Moreover, there will always be at least one of each letters.

Consider the words $w_1 = aaaabbbb \in L(G)$ and $w_2 = babababa \in L(G)$. Both of these words have the same Parikh image: $\Psi_\Sigma(w_i) = (4, 4)$. It is interesting to note that most information embedded in a word generated with a context-free grammar is thrown away by the Parikh mapping.

This loss of information is expressed in the Parikh's theorem [75], stating that the Parikh image of a context-free language is semilinear. The biggest implication of the theorem though, is that it shows that if the order of the symbols is ignored, then it is impossible to distinguish between a regular set and a context-free language [62].

Another interesting result appeared in [52] that every semilinear set is a finite union of disjoint linear sets.

2.8 Register machines

As a referential universal language acceptor we will use Minsky's register machine [51]. Such a machine runs a program consisting of numbered instructions of several simple types. Several variants of register machines with different number of registers and different instructions sets were shown to be computationally universal (see [50] for some original definitions and [56] for the definition used in this thesis).

Definition 2.8.1. *A* n**-register machine** *is a tuple* $M = (n, P, i, h, Lab)$*, where:*

- $m \in \mathbb{N}$ *is the number of registers,*
- P *is a set of labeled instructions of the form* $j : (op(r), k, l)$*, where* $op(r)$ *is an operation on register* $r \leq n$*, and* j*,* k*,* l *are labels from the set* Lab *such that there are no two instructions with the same label* j*,*
- $i \in Lab$ *is the initial instruction label,*
- $h \in Lab$ *is the final instruction label, and*
- Lab *is the finite set of instruction labels.*

We will use the term instruction j to denote the instruction with label j. The machine is capable of the following instructions:

- $(add(r), k, l)$: Add one to the contents of register r and proceed to instruction k or to instruction l; in the deterministic variants usually considered in the literature we demand $k = l$.
- $(sub(r), k, l)$: If register r is not empty, then subtract one from its contents and go to instruction k, otherwise proceed to instruction l.
- *halt* : This instruction stops the machine. This additional instruction can only be assigned to the final label h.

Definition 2.8.2. *A configuration of an* n*-register machine* $M = (n, P, i, h, Lab)$ *is a tuple* $(j, m_1, \ldots, m_n)$*, where* $j \in Lab$ *is a label of the current instruction and* $(m_1, \ldots, m_n) \in N_0^n$ *are contents of registers 1 to* m*.*

An n-register machine can analyze an input $(n_1, \ldots, n_m) \in N_0^n$ in registers 1 to n, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty (this last requirement is not necessary). If the machine does not halt, the analysis was not successful.

2.9 Petri nets

Another well-studied formalism often referred to (e.g. in [27, 37]) are Petri nets. Petri nets [82, 103, 43] were introduced by Carl Adam Petri in 1962 in his PhD thesis. A Petri net is a graphical and mathematical tool for the modeling of concurrent processes and analysis of system behavior. A Petri net is usually drawn as a directed bipartite graph with two kind of nodes. Places are represented by circles within which each small black dot denotes a token. Transitions are represented by bars. Each edge is either from a place to a transition or vice versa.

Definition 2.9.1. *A* **Petri net** *is a tuple* (P, T, φ)*, where:*

- P *is a finite set of places,*
- T *is a finite set of transitions,*
- $\varphi : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is a flow function.*

The edges of the bipartite graph are annotated by either $\varphi(p, t)$ or $\varphi(t, p)$, where $p \in P$ and $t \in T$ are two endpoints of the arc. If $\varphi(p, t) = 1$ or $\varphi(t, p) = 1$, we usually omit the label.

Definition 2.9.2. *A* **marking** *is a mapping* $\mu : P \to \mathbb{N}$.

The mapping μ assigns certain number of tokens to each place of the net. For places $P = \{p_1, p_2, \ldots, p_n\}$ we can view a marking μ as an n-dimensional column vector in which the ith component is $\mu(p_i)$. Examplar usage of such vectors is in the Figure 2.2.

Definition 2.9.3. *A marking* μ_1 **covers** *marking* μ_2*, when* $\forall p \in P : \mu_1(p) \geq \mu_2(p)$ *- in each place there is no less tokens in* μ_1 *than in* μ_2*. We denote it by* $\mu_1 \geq \mu_2$.

Definition 2.9.4. *A transition* $t \in T$ *is* **enabled** *at a marking* μ *iff* $\forall p \in P, \varphi(p, t) \leq \mu(p)$.

If a transition t is enabled, it may fire by removing $\varphi(p, t)$ tokens from each input place p and putting $\varphi(t, p')$ tokens in each output place p'. We then write $\mu \xrightarrow{t} \mu'$, where $\forall p \in P : \mu'(p) = \mu(p) - \varphi(p, t) + \varphi(t, p)$.

Example 2.9.1. In the Figure 2.1 the Petri net has four places and two transitions. At the current marking the transition t_1 is enabled and the transition t_2 is not enabled. Firing the transition t_1 takes one token from the place p_1 and produces one token to places p_2, p_3 and p_4. In the resulting marking both transitions t_1 and t_2 are enabled.

Definition 2.9.5. *By establishing an ordering on the elements of P and T (i.e. $P = \{p_1, p_2, \dots p_k\}$ and $T = \{t_1, t_2, \dots t_m\}$), we define the $k \times m$* **incidence matrix** *$[T]$ so that $[T](i,j) = \varphi(t_j, p_i) - \varphi(p_i, t_j)$.*

Note that $\varphi(t_j, p_i), \varphi(p_i, t_j), [T](i,j)$ represents the number of tokens removed, added, and changed in place i when transition j fires once. Thus, if we view a marking μ as a k-dimensional column vector in which the ith component is $\mu(p_i)$, jth column of $[T]$ is then a k-dimensional vector such that if $\mu \xrightarrow{t_j} \mu'$, then the following state equation holds: $\mu + [T](j) = \mu'$. We call this column vector a **displacement** of transaction t_j and denote it with $\Delta(t_j)$.

Definition 2.9.6. *A* **marked Petri net** *is a tuple (P, T, φ, μ_0), where (P, T, φ) is a Petri net and μ_0 is called the initial marking.*

Definition 2.9.7. *A sequence of transitions $\sigma = t_1 \dots t_n$ is a* **firing sequence** *from μ_0 iff $\mu_0 \xrightarrow{t_1} \mu_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} \mu_n$ for some markings $\mu_1, \dots, \mu_n$. We also write $\mu_0 \xrightarrow{\sigma} \mu_n$.*

We write $\mu_0 \xrightarrow{\sigma}$ to denote that σ is enabled and can be fired from μ_0, i.e., $\mu_0 \xrightarrow{\sigma}$ iff there exists a marking μ such that $\mu_0 \xrightarrow{\sigma} \mu$. The notation $\mu_0 \xrightarrow{*} \mu$ is used to denote the existence of a firing sequence σ such that $\mu_0 \xrightarrow{\sigma} \mu$.

Definition 2.9.8. *A marking μ is reachable for a marked Petri net $\mathcal{P} = (P, T, \varphi, \mu_0)$ iff $\mu_0 \xrightarrow{*} \mu$.*

Definition 2.9.9. *Let $\mathcal{P} = (P, T, \varphi, \mu_0)$ be a marked Petri net. The* **reachability set** *of $\mathcal{P}$ is $R((P)) = \{\mu | \mu_0 \xrightarrow{*} \mu\}$.*

A notion of reachability graph is helpful for analyzing the behavior of a Petri net as a tool for visualisation of the structure of the reachability set.

Definition 2.9.10. *Let $\mathcal{P} = (P, T, \varphi, \mu_0)$ be a marked Petri net. The* **reachability graph** *of $\mathcal{P}$ is a labelled graph whose nodes are the reachable markings and edge from μ_1 to μ_2 is labeled with a transition $t \in T$ iff $\mu_1 \xrightarrow{t} \mu_2$.*

Example 2.9.2. Consider a Petri net $\mathcal{P}$ from the Figure 2.1. Its reachability graph is in the Figure 2.2. $\mathcal{P}$ is not bounded because by alternately firing transitions t_1 and t_2 we can reach infinitely many different markings. We can also easily see that it is live, because in every marking for every transition $t \in \{t_1, t_2\}$ there is a firing sequence ending with t.

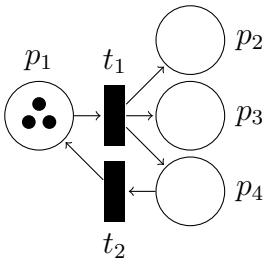


Figure 2.1: An example Petri net

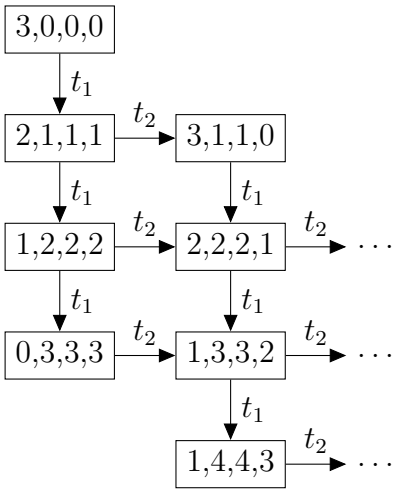


Figure 2.2: An example reachability graph

In spite of its simplicity, the applicability of the technique of reachability graph analysis is rather limited in the sense that it suffers from the state explosion phenomenon as the sizes of the reachability sets grow beyond any primitive recursive function in the worst case [103].

Coverability graph analysis offers an alternative to the techinque of reachability graph analysis by abstracting out certain details to make the graph finite. To understand the intuition behind coverability graphs, consider the figure 2.2 which shows a part of the reachability graph of the Petri net in the figure 2.1. Consider the path $(3,0,0,0) \xrightarrow{t_1} (2,1,1,1) \xrightarrow{t_2} (3,1,1,0)$ along which the places p_2 and p_3 both gain an extra token in the end, i.e. $(3,0,0,0) < (3,1,1,0)$. Clearly they can be made to contain arbitrary large number of tokens by repeating the firing sequence t_1t_2 for a sufficient number of times, as $(3,0,0,0) \xrightarrow{t_1t_2} (3,1,1,0) \xrightarrow{t_1t_2} (3,2,2,0) \xrightarrow{t_1t_2} \ldots \xrightarrow{t_1t_2} (3,n,n,0)$, for arbitrary n. In order to capture the notion of a place being unbounded, we short-circuit the above infinite sequence of computation as $(3,0,0,0) \xrightarrow{t_1} (2,1,1,1) \xrightarrow{t_2} (3,\omega,\omega,0)$, where ω is a symbol denoting something being arbitrarily large. As it turns out, the coverability graph of a Petri net is always finite [55, 43]. The corresponding coverability graph of the example Petri net in the figure 2.1 is in the figure 2.3. The algorithm for generating the coverability graph of a Petri net [103] is shown on the following page.

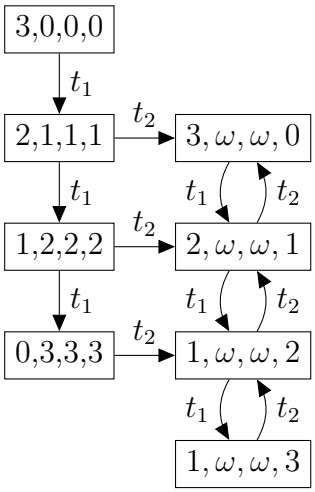


Figure 2.3: An example coverability graph

Algorithm 1 Coverability graph algorithm

procedure CoverabilityGraph(marked Petri net $\mathcal{P} = (P, T, \varphi, \mu_0)$)
  create a node μ_{init} such that $\mu_{init} = \mu_0$ and mark it as "new"
  while there is a "new" node μ **do**
    for each transition t enabled at μ **do**
      if there is a node $\mu' = \mu + \Delta t$ **then**
        add an edge $\mu \xrightarrow{t} \mu'$
      else if there is a path $\mu_{init} \xrightarrow{*} \mu'' \xrightarrow{*} \mu$ such that $\mu'' < \mu + \Delta t$ **then**
        add a "new" node x with
          $x(p) = \omega$ if $\mu''(p) < (\mu + \Delta t)(p)$
          $x(p) = \mu''(p)$ otherwise
        add an edge $\mu \xrightarrow{t} x$
      else
        add a "new" node x with $x = \mu + \Delta t$ and an edge $\mu \xrightarrow{t} x$
    mark μ with "old"

2.9.1 Analysis of behavioral properties

Analysis of several behavioral properties is studied and following decidability problems are of special importance:

The boundedness problem

The boundedness problem is, given a marked Petri net $\mathcal{P}$, deciding whether $|R(\mathcal{P})|$ is finite. This problem was first considered by Karp and Miller [55], where it was shown to be decidable using the technique of coverability graph analysis. A Petri net is unbounded iff an ω occurs in the corresponding coverability graph. The algorithm presented there was basically an unbounded search and consequently no complexity analysis was shown. Subsequently, a lower bound of $O(2^{cm})$ space was shown by Lipton in [65], where m is the number of places in the Petri net and c is a constant. Finally, an upper bound of $O(2^{cn \log n})$ space was given by Rackoff in [86]. Here, however, n represents the size or number of bits in the problem instance and c is a constant.

The covering problem

The covering problem is, given a marked Petri net $\mathcal{P}$ and a marking μ, deciding whether there exists $\mu' \in R(\mathcal{P})$ such that $\mu' \geq \mu$. The complexity (both upper and lower bounds) of the covering problem can be derived along a similar line of that of the boundedness problem [86].

The reachability problem

The reachability problem is, given a marked Petri net $\mathcal{P}$ and a marking μ, deciding whether $\mu \in R(\mathcal{P})$. This problem has attracted the most attention in the Petri net community. One reason is that the problem has many real-world applications; furthermore, it is the key to the solutions of several other Petri net problems. Before the decidability question of the reachability problem for general Petri nets was proven by Mayr in 1981 [67], a number of attempts had been made to investigate the problem for restricted classes of Petri nets, in hope of gaining more insights and developing new tools in order to conquer the general Petri net reachability problem. It should be noted that the technique of the coverability graph analysis does not answer the reachability problem as ω abstracts out the exact number of tokens that a place can accumulate, should the place be potentially unbounded.

The containment problem

The containment problem is, given two marked Petri nets $\mathcal{P}_1$ and $\mathcal{P}_2$, deciding whether $R(\mathcal{P}_1) \subseteq R(\mathcal{P}_2)$. In the late 1960's, Rabin first showed the containment problem for Petri nets to be undecidable. Even though the original work of Rabin have never been published, a new proof based on Hilbert's Tenth Problem [28] was presented at MIT in 1972 [9].

The equivalence problem

The equivalence problem: given two marked Petri nets $\mathcal{P}_1$ and $\mathcal{P}_2$, deciding whether $R(\mathcal{P}_1) = R(\mathcal{P}_2)$. In 1975, Hack [46] extended Rabin's result of the containment problem by showing the equivalence problem to be undecidable as well. The proof was also based on Hilbert's Tenth Problem.

The liveness problem

The liveness problem: given a marked Petri net $\mathcal{P}$, deciding whether for every $t \in T, \mu \in R(\mathcal{P})$ there exists a sequence of transitions σ such that $\mu \xrightarrow{\sigma t}$, i.e. t is enabled after firing σ from μ. In [44], several variants of the reachability problem were shown to be recursively equivalent. Among them is the single-place zero reachability problem, i.e. the problem of determining whether a marking with no tokens in a designated place can be reached. Hack [46] also showed the single-place zero reachability problem to be recursively equivalent to the liveness problem, which is then as well decidable.

2.9.2 Backwards-compatible extensions

Conventional Petri nets have several limitations such as the inability to test for zero tokens in a place. Models often tend to become large, with no support for structuring. To remedy these weaknesses, a number of extended Petri net models have been proposed in the literature. Some of these extensions only increase the modeling convenience by making the model more expressive while they do not actually increase the power of the basic model and are completely backwards-compatible, e.g. colored Petri nets.

Colored Petri nets

A colored Petri net (CPN) has each token attached with a color, indicating the identity of the token. Moreover, each place and each transition has attached a set of colors. A transition can fire with respect to each of its colors. By firing a transition, tokens are removed from the input places and

added to the output places in the same way as that in original Petri nets, except that a functional dependency is specified between the color of the transition firing and the colors of the involved tokens. The color attached to a token may be changed by a transition firing and it often represents a complex data-value. CPNs lead to compact net models by using of the concept of colors.

2.9.3 Turing complete extensions

The computational power of conventional or backwards-compatible Petri nets such as CPNs is strictly weaker than that of Turing machines [103]. From a theoretical viewpoint, the limitation of Petri nets is precisely due to the lack of abilities to test potentially unbounded places for zero and then act accordingly. With zero-testing capabilities, it is fairly easy to show the equivalence to register machines, which are Turing equivalent. We will mention several extensions, which add properties that cannot be modeled in the original Petri net.

Petri nets with inhibitor arcs

The inhibitor arc connects an input place to a transition. The presence of an inhibitor arc connecting an input place to a transition changes the transition enabling conditions. In the presence of the inhibitor arc, a transition is regarded as enabled if each input place, connected to the transition by a normal arc, contains at least the number of tokens equal to the weight of the arc, and no tokens are present on each input place connected to the transition by the inhibitor arc. The transition firing rule is the same for normally connected places. The firing, however, does not change the marking in the inhibitor arc connected places.

Time petri nets

Time Petri nets (TPN) are Petri nets in which each transition t has attached two times $a \leq b$. Assuming that t was last enabled at time c, then t may fire only during the interval $[c + a, c + b]$ and must fire at time $c + b$ at the latest unless it is disabled before by the firing of another transition. Firing a transition takes no time.

Timed Petri nets

Timed Petri nets (TdPN) are obtained from Petri nets by associating a firing time (delay) to each transition of the net. Moreover, each token has an

age property, so the marking of a place p is a finite multiset of ages. The precondition of a transition with input place p is an interval. A transition is enabled if for every input place there exists an appropriate token, i.e. its age is included in the interval.

Prioritized Petri nets

Transitions in prioritized Petri nets have input arcs of two types: normal and prioritized. A place with a token and several transitions enabled from this place will fire the transition with a priority arc first. If there are more than one priority arc outgoing from a place which causes that more than one transition is enabled, then the firing choice is nondeterministic.

Maximal parallel Petri nets

Under the maximal parallel semantics, maximal sets of simultaneously enabled rules are fired. They are of interest due to their close connections with the model of P systems.

2.9.4 Other extensions

While the above extended Petri nets are powerful enough to simulate Turing machines, all nontrivial behavioral properties for such Petri nets become undecidable. A natural and interesting question to ask is: are there Petri nets whose powers lie between conventional Petri nets and Turing machines? As it turns out, the quest for such weaker extensions has attracted considerable attention in recent years.

Reset nets

A reset arc does not impose a precondition on firing, and empties the place when the transition fires. This makes reachability and boundedness undecidable, while some other properties, such as termination, coverability remain decidable.

Transfer nets

Petri nets with transfer arcs (transfer nets) can contain transitions that can also consume all the tokens present in one place regardless of the actual number of tokens and move them to another place. The termination and coverability remain decidable and reachability undecidable, on the other hand, in this case the boundedness is decidable [33].

2.10 Vector addition systems

Vector addition systems were introduced by Karp and Miller [55], and were later shown by Hack [45] to be equivalent to Petri nets.

Definition 2.10.1. *A* **vector addition system** *(VAS) is a pair* $G = (x, W)$*, where* $x \in \mathbb{N}^n$ *is an initial vector and* $W \subseteq \mathbb{Z}^n$ *is a finite set of vectors, where* $n > 0$ *is called the dimension of VAS.*

The initial vector is seen as the initial values of multiple counters and the vectors in W are seen as actions that update the counters. These counters may never drop below zero.

Definition 2.10.2. *The* **reachability set** *of the VAS* $G = (x, W)$ *is the set* $R(G) = \{z | \exists v_1, \ldots, v_j \in W : z = x + v_1 + \ldots + v_j \wedge \forall 1 \leq i \leq j : x + v_1 + \ldots + v_i \geq 0\}$.

Definition 2.10.3. *A* **vector addition system with states** *(VASS) is a tuple* $G = (x, W, Q, T, p_0)$*, where:*

- (x, W) *is a vector addition system,*
- Q *is a finite set of states,*
- T *is a finite set of transitions of the form* $p \to (q, v)$*, where* $v \in W$ *and* $p, q \in Q$ *are states,*
- $p_0 \in Q$ *is the starting state.*

The transition $p \to (q, v)$ can be applied at vector y in state p and yields the vector $y + v$ in state q, provided that $y + v \geq 0$.

Example 2.10.1. For the Petri net in the Figure 2.1, the corresponding VAS (x, W) is:

- $x = (3, 0, 0, 0)$,
- $W = \{(-1, 1, 1, 1), (1, 0, 0, -1)\}$.

It is known [45] that Petri nets, VAS and VASS are computationally equivalent.

2.11 Graph theory

Biological membranes forms a hierarchical structure, which can be represented by a tree. In this section we define several notions from the graph theory taken from [30].

Definition 2.11.1. *An* **undirected graph** *is a pair $G = (V, E)$ of sets such that $E \subseteq V \times V$ and $V \cap E = \emptyset$. The elements of V are the vertices (or nodes) of the graph G, the elements of E are its edges.*

The vertex set of a graph G is denoted by $V(G)$, its edge set as $E(G)$.

Definition 2.11.2. *The* **order of the graph** *G is the number of its vertices, denoted by $|G|$. Graphs are (in)finite iff their order is (in)finite.*

Definition 2.11.3. *A vertex $v \in V$ is* **incident** *with an edge $e \in E$ iff $v \in e$, i.e. either $e = (v, y)$, where $y \in V$ or $e = (x, v)$, where $x \in V$.*

Definition 2.11.4. *Two vertices $x, y \in V(G)$ are* **adjacent** *iff $(x, y) \in E(G)$.*

Definition 2.11.5. *A* **path** *is a non-empty graph $P = (V, E)$, where $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{(x_i, x_{i+1}) | 0 \leq i < k\}$ and the x_i are all distinct.*

Definition 2.11.6. *If $P = (V, E)$ is a path where $V = \{x_0, x_1, \ldots, x_k\}$ and $|V| \geq 3$, then the graph $C = (V, E \cup \{(x_k, x_0)\})$ is called a* **cycle**.

Definition 2.11.7. *A graph $G = (V, E)$ is a* **subgraph** *of a graph $G' = (V', E')$ iff $V \subseteq V'$ and $E \subseteq E'$. We denote it by $G \subseteq G'$ and also say that G' contains G.*

Definition 2.11.8. *A non-empty graph G is called* **connected** *iff any two of its vertices are linked by a path in G.*

Definition 2.11.9. *A graph not containing any cycle is called a* **forest**.

Definition 2.11.10. *A connected forest is called a* **tree**.

Sometimes it is convenient to consider one vertex of a tree as special. Such a vertex is then called the root of this tree.

Definition 2.11.11. *A tree T with a fixed root is called a* **rooted tree**. *The root node of T is denoted by r_T.*

Definition 2.11.12. *Let d be a non-root node of a tree T, i.e. $d \in V(T) \setminus \{r_T\}$. As T is a tree, there is a unique path from d to r_T [30]. The node adjacent to d on that path is also unique and is called a* **parent node** *of d and is denoted by $parent_T(d)$.*

Definition 2.11.13. *Let T_1, T_2 be two rooted trees. A bijection $f : V(T_1) \to V(T_2)$ is an* **isomorphism** *iff $\forall x, y \in V(T) : (x, y) \in E(T_1) \Leftrightarrow (f(x), f(y)) \in E(T_2)$.*

Rooted trees T_1 and T_2 are called isomorphic if there exists an isomorphism on their nodes.

2.12 Bisimulations

For proving equivalence between two computation models, a notion of bisimulation is essential. Specifically, there are multiple notions that represent various equivalences. Definitions in this section are taken from [29].

Definition 2.12.1. *A* **state transition system** *is a pair $(S, \to)$, where S is a set of states and $\to \subseteq S \times S$ is a binary transition relation over S.*

If $p, q \in S$, then $(p, q) \in \to$ is usually written as $p \to q$. This represents the fact that there is a transition from state p to state q.

Definition 2.12.2. *A* **labeled state transition system** *(LTS) is a tuple $(S, A, \to)$, where S is a set of states, A is a set of labels and $\to \subseteq S \times A \times S$ is a ternary transition relation.*

If $p, q \in S$ and $a \in A$, then $(p, a, q) \in \to$ is usually written as $p \xrightarrow{a} q$. This represents the fact that there is a transition from state p to state q with a label a.

Definition 2.12.3. *Let $(S_1, A, \to)$ and $(S_2, A, \to)$ be two labeled transition systems. A* **simulation** *is a binary relation $R \subseteq S_1 \times S_2$ such that if $(s_1, s_2) \in R$ then for each $s_1 \xrightarrow{a} t_1$ there is some $s_2 \xrightarrow{a} t_2$ such that $(t_1, t_2) \in R$.*

Definition 2.12.4. *Let $(S_1, A, \to)$ and $(S_2, A, \to)$ be two labeled transition systems. A* **strong bisimulation** *is a binary relation $R \subseteq S_1 \times S_2$ such that if $(s_1, s_2) \in R$ then:*

1. *for each $s_1 \xrightarrow{a} t_1$ there is some $s_2 \xrightarrow{a} t_2$ such that $(t_1, t_2) \in R$,*
2. *for each $s_2 \xrightarrow{a} t_2$ there is some $s_1 \xrightarrow{a} t_1$ such that $(t_1, t_2) \in R$.*

If there exists a strong bisimulation we say the two systems are strongly bisimilar.

Sometimes, we want to allow systems to perform internal (silent) steps, of which the impact is considered unobservable. That is why there are multiple notions of the bisimulation.

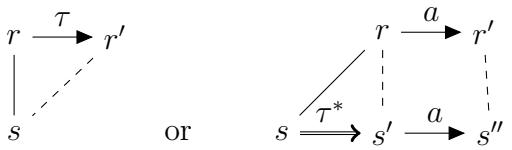


Figure 2.4: Transfer diagram for branching bisimulation

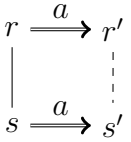


Figure 2.5: Transfer diagram for weak bisimulation

Definition 2.12.5. *A* **labeled transition system with silent actions** *is a triple $(S, A, \rightarrow)$, where S is a set of states, A is a set of actions, also a silent action τ is assumed that is not in A and $\rightarrow \subseteq S \times (A \cup \tau) \times S$ is the transition relation.*

We use A_τ to denote $A \cup \{\tau\}$. By $\stackrel{\tau^*}{\Rightarrow}$ we denote the transitive and reflexive closure of $\stackrel{\tau}{\rightarrow}$. For $a \in A$ we define the relation $\stackrel{a}{\Rightarrow}$ on S by $r \stackrel{a}{\Rightarrow} s$ iff there exists $r', s' \in S$ such that $r \stackrel{\tau^*}{\Rightarrow} r' \stackrel{a}{\rightarrow} s' \stackrel{\tau^*}{\Rightarrow} s$.

Definition 2.12.6. *A* **run** *of a labeled transition system is a finite, non-empty alternating sequence $\rho = s_0 a_0 s_1 a_1 \ldots s_{n-1} a_{n-1} s_n$ of states and actions, beginning and ending with a state such that for $0 \leq i < n : s_i \stackrel{a_i}{\rightarrow} s_{i+1}$.*

We also say that ρ is a run from s_0. We denote $first(\rho) = s_0$ and $last(\rho) = s_n$.

Definition 2.12.7. *A relation $R \subseteq S \times S$ is called a* **branching bisimulation** *if it is symmetric and satisfies the following transfer property: if rRs and $r \stackrel{a}{\rightarrow} r'$ then either $a = \tau$ and $r'Rs$ or there are two states $s', s'' \in S$ such that $s \stackrel{\tau^*}{\Rightarrow} s' \stackrel{a}{\rightarrow} s''$, rRs' and $r'Rs''$.*

If there exists a branching bisimulation we say the two systems are branching bisimilar.

Example 2.12.1. The diagrams shown in the Figure 2.4 summarize the main transfer properties of branching bisimulation. We have used the dashed lines to represent the relations that have to be established in order to conclude that the two states connected by the plain line are bisimilar.

We could have strengthened the above definition of branching bisimulation by requiring all intermediate states in $s \stackrel{\tau^*}{\Rightarrow} s'$ to be related with r. However, that would lead to the same equivalence relation as is proven in [29].

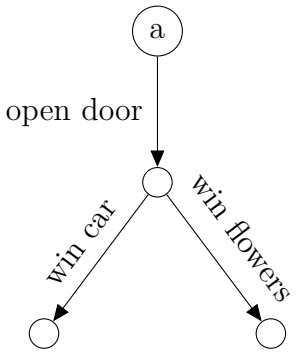


Figure 2.6: Example run of a labeled transition system

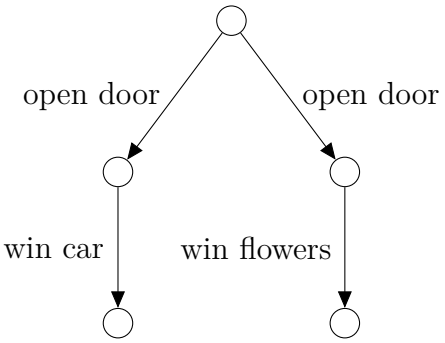


Figure 2.7: Example run of a labeled transition system with hidden branching before the first action

Definition 2.12.8. *A relation $R \subseteq S \times S$ is called a* **weak bisimulation** *if it is symmetric and satisfies the following transfer property: if rRs and $r \stackrel{a}{\Rightarrow} r'$ then there exist $s' \in S$ such that $s \stackrel{a}{\Rightarrow} s'$ and $r'Rs'$.*

If there exists a weak bisimulation we say the two systems are weakly bisimilar. This means that two states are considered equivalent if they lead via the same sequences of visible actions to bisimilar states. The intermediate states are not questioned. The diagram in the Figure 2.5 summarizes the transfer property for the weak bisimulation. We have used the same notational conventions of the Figure 2.4.

Example 2.12.2. In the Figure 2.6 a participant in a contest opens a door and then it is decided if he wins a car or flowers. In the Figure 2.7 the price is decided beforehand so when a participant opens a door, he can directly see his outcome. We can model these events (open door, win car, win flowers) as actions of a labeled transition system.

The contestant is not interested in the inner states of the system, he only sees these events. For him, the two systems are weakly bisimilar.

In the state right after opening the door and before the price is given there is no way for him knowing the outcome. However, in the system from the Figure 2.7 some people who organize the contest already know what is the outcome. For them, the system is not branching bisimilar.

2.13 Rice's theorem

Consider any kind of software testing problem. Its description will typically start as "For a given program decide whether the function it computes is. . . ".

In the setting of Turing machines, we often encounter natural problems of the form "Decide if the language recognized by a given Turing machine...". Rice's theorem [89] proves in one clean sweep that all these problems are undecidable. That is, whenever we have a decision problem in which we are given a Turing machine and we are asked to determine a property of the language recognized by the machine, that decision problem is always undecidable. The only exceptions will be the trivial properties that are always true or always false.

Note that Rice's theorem is applicable only for Turing complete models. It means that the properties analyzed in section 2.9.1 can still be perfectly decidable as they are studied on Petri nets, which are not Turing complete.

We will mention Rice's theorem again in section 4.2.

2.14 Reaction systems

In this section we present reaction systems [92], a formal framework for investigating processes driven by interactions between biochemical reactions in living cells. These interactions are based on the mechanisms of facilitation and inhibition: a reaction can take place if all of its reactants are present and none of its inhibitors is present. If a reaction takes place, then it creates its products. Therefore to specify a reaction one needs to specify its set of reactants, its set of inhibitors, and its set of products. This leads to the following definition.

Definition 2.14.1. *A* **reaction** *is a triplet $a = (R, I, P)$, where R, I, P are finite sets. If S is a set such that $R, I, P \subseteq S$, then a is a reaction in S.*

The sets R, I, P are also denoted R_a, I_a, P_a, and called the reactant set of a, the inhibitor set of a and the product set of a.

Definition 2.14.2. *Let a be a reaction and T a finite set. Then a is enabled by T, denoted by a en T, iff $R_a \subseteq T$ and $I_a \cap T = \emptyset$.*

Definition 2.14.3. *Let a be a reaction and T a finite set. The result of a on T is defined by: $res_a(T) = P_a$ if a en T and $res_a(T) = \emptyset$ otherwise.*

Clearly, if $R_a \cap I_a \neq \emptyset$, then $res_a(T) = \emptyset$ for every T. Therefore, we can assume that, for each reaction a, $R_a \cap I_a = \emptyset$ and we will also assume that $R_a \neq \emptyset$, $I_a \neq \emptyset$ and $P_a \neq \emptyset$.

Example 2.14.1. Consider the reaction a with $R_a = \{c, x_1, x_2\}$, $I_a = \{y_1, y_2\}$, and $P_a = \{c, z\}$. We can interpret c as the catalyzer of a as it

is needed for a to take place, but is not consumed by a. Then a *en* T for $T = \{c, x_1, x_2, z\}$, and a is not enabled on neither $\{c, x_1, x_2, z, y_1\}$ nor on $\{x_1, x_2, z\}$.

A reaction a is enabled on a set T if T separates R_a from I_a, i.e. $R_a \subseteq T$ and $I_a \cap T = \emptyset$. There is no assumption about the relationship of P_a to either R_a or I_a. When a is enabled by a finite set T, then $res_a(T) = P_a$. Thus, the result of a on T is "locally determined" in the sense that it uses only a subset of T. However, the result of the transformation is global as all elements from $T \setminus P_a$ "vanished". This is in great contrast to classical models like Petri nets (see section 2.9), where the firing of a single transition has only a local influence on the global marking which may be changed only on places that are neighbouring the given transition. In reaction systems there is no permanency of elements: an element of a global state vanishes unless it is sustained by a reaction.

Definition 2.14.4. *Let A be a finite set of reactions. The result of A on T is defined by*

$$res_A(T) = \bigcup_{a \in A} res_a(T)$$

.

There are no conditions on the relationship between reactions in A nor the notion of conflict here: if $a, b \in A$ with a *en* T and b *en* T, then, even if $R_a \cap R_b \neq \emptyset$, still both a and b contribute to $res_A(T)$, i.e. $res_a(T) \cup res_b(T) \subseteq res_A(T)$.

There is no counting in reaction systems, it is a qualitative rather than quantitative model. This reflects the assumption about the "threshold supply" of elements: either an element is present, and then there is "enough" of it, or an element is not present.

The notion of conflict is reflected via the notion of consistency.

Definition 2.14.5. *A set of reactions A is* **consistent** *iff $R_A \cap I_A = \emptyset$, i.e. $R_a \cap R_b = \emptyset$ for any two reactions $a, b \in A$.*

Clearly, if $R_a \cap I_b \neq \emptyset$, then a and b can never be together enabled.

Definition 2.14.6. *A* **reaction system** *is an ordered pair $\mathcal{A} = (S, A)$ such that S is a finite set and A is a set of reactions in S.*

There are many situations where one needs to assign quantitative parameters to states (e.g. when dealing with time issues). A numerical value can be assigned to a state T if there is a measurement of T yielding this value. This leads to the notion of reaction systems with measurements [34], where a

finite set of measurement functions is added as a third component to reaction systems.

There are various extensions of reaction systems, but they are based on the core definition with basic notions of "no permanence" and "no counting" of elements. We will mention reaction systems again in section 3.2 as an inspiration for some variants of P systems.

Chapter 3

P systems

In subsection 1.3.6 we briefly introduced the notions of membrane, membrane structure and P system. Chapter 2 introduced some basic definitions useful for better understanding of further reading. In this Chapter we define a P system (section 3.1), its variants with recent advances (section 3.2) and case studies (section 3.3).

A P system [77] is a computing model which abstracts from the way the alive cells process chemical compounds in their compartmental structure. In short, in the regions defined by a membrane structure we have objects which can be described by symbols. Their multiplicity matters, that is, we work with multisets of objects placed in the regions of the membrane structure. Objects evolve according to given rules. Any object, alone or together with another objects, can be transformed in other objects, can pass through a membrane, and can dissolve the membrane in which it is placed. All objects evolve at the same time, in parallel manner across all membranes. The evolution rules are hierarchized by a priority relation, which is a partial order. By using the rules in a nondeterministic, maximally parallel manner, one gets transitions between the system configurations. A sequence of transitions is a computation. With a halting computation we can associate a result, in the form of the objects present in a given membrane in the halting configuration, or expelled from the system during the computation.

These aspects all together forms a P system as introduced in [77]. An example of a P system containing rules for computing the Fibonacci sequence can be seen in the figure 3.1.

In section 3.1 we will provide a formal definition of a P system. We will follow in section 3.2 by presenting most common P system variants along with a survey of known results and in section 3.3 we show how P systems can be of use in practice case studies.

3.1 Definitions

The definition of a P system is based on the notion of membrane structure [12], which is introduced in terms of the language MS over the alphabet $\{[,]\}$, consisting of the strings recurrently defined as follows:

1. $[] \in MS$,
2. if $\mu_1, \ldots, \mu_n \in MS$, then $[\mu_1 \ldots \mu_n] \in MS$,
3. nothing else is in MS.

We assume that each pair of well matching parentheses [] can be labelled in a one-to-one manner by an integer in the set $1 \ldots m$, e.g. $[[]_2[]_3]_1$.

We define now an equivalence relation over the elements of MS: we write $x \sim y$ if and only if:

1. $x = []_i$ and $y = []_i$,
2. $x = [\mu_1 \ldots \mu_n]_j$, $y = [\xi_1 \ldots \xi_n]_j$ and $\mu_i \sim \xi_{p(i)}$ for every i and for some permutation p.

Thus, the order of parentheses is irrelevant as long as they are at the same hierarchical level. The set of equivalence classes with respect to the equivalence relation $\sim$ is denoted by $\overline{MS}$.

Definition 3.1.1. *Any element in $\overline{MS}$ is called a* **membrane structure**.

Definition 3.1.2. *Each pair of matching parentheses [] appearing in a membrane structure μ is called a* **membrane**.

Definition 3.1.3. *The external membrane of a membrane structure is called the* **skin membrane**.

Definition 3.1.4. *Any membrane without any other membrane inside (appearing in a membrane structure in the form []) is called an* **elementary membrane**.

Definition 3.1.5. *The number of membranes in μ is called the* **degree** *of μ and it is denoted by $deg(\mu)$.*

Definition 3.1.6. *The depth of a membrane structure μ, $dep(\mu)$, is recurrently defined as:*

1. *if $\mu = []$, then $dep(\mu) = 1$;*

2. *if* $\mu = [\mu_1 \dots \mu_n]$ *for some* $\mu_1, \dots, \mu_n \in MS$, *then* $dep(\mu) = 1 + max\{dep(\mu_i\}|1 \leq i \leq n)$.

Each membrane determines a compartment, also called region, the space delimited from above by it and from below by the membranes placed directly inside, if any exists. The correspondence membrane-region is one-to-one, that is why we sometimes use interchangeably these terms.

Next step is inserting objects into regions and adding them the possibility to evolve and communicate according to certain specified rules.

Definition 3.1.7. *An* **evolution rule** *over an alphabet* Σ *is a pair* (u, v), *which will be written as* $u \to v$, *where* u *is a string over* Σ *and* $v = v'$ *or* $v = v'\delta$, *where* v' *is a string over* $\Sigma \times (\{here, out\} \cup \{in_j|1 \leq j \leq m\})$ *and* δ *is a special symbol not in* Σ.

An example of such rule is $ab^2 \to (a, here)(b, in_3)(c, out)(c, here)$. An evolution rule specifies for each object on the right side, whether it stays in the current region, moves through a membrane to the parent region or through a membrane to one of the child regions.

Definition 3.1.8. *For an evolution rule* $r : u \to v$, *the size of multiset* u *is called the* **radius** *of* r.

Note that the strings u, v are understood as representations of multisets over Σ, in a sense from section 2.6.

We are now ready to proceed with a formal definition of a P system of degree m.

Definition 3.1.9. *A* **P system** *is a tuple*

$$\Pi = (\Sigma, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m),$$

where:

- Σ *is the alphabet of objects,*
- μ *is a membrane structure consisting of* m *membranes labeled with numbers* $1, 2, \dots, m$,
- $w_1, w_2, \dots w_m$ *are multisets of objects from* Σ *initially present in the regions* $1, 2, \dots, m$ *of the membrane structure,*
- $R_1, R_2, \dots R_m$ *are finite sets of evolution rules associated with the regions* $1, 2, \dots, m$ *of the membrane structure.*

Some more remarks about the definition of a P system are to be given, before explaining its functioning as a computing device.

Any of the initial multisets $w_1, w_2, \ldots, w_m$ can be empty as well as any of the sets of evolution rules $R_1, R_2, \ldots R_m$.

The $(m+1)$-tuple $(\mu, w_1, w_2, \ldots w_m)$ constitutes the initial configuration of the P system Π. In general, the configuration is represented by the membrane structure of $\Pi$and the multisets of objects in the regions.

Definition 3.1.10. *A* **configuration** *of a P system Π is an $(m+1)$-tuple $(\mu', w'_1, w'_2, \ldots w'_m)$, where μ' is a membrane structure and $w'_1, w'_2, \ldots w'_m$ are multisets over Σ present in the regions $1, 2, \ldots, m$ of μ'.*

Note that the membrane structure can be different from the initial membrane structure, according to the dissolving action, which will be explained later.

Definition 3.1.11. *A rule $r : u \to v$ is* **applicable** *in the configuration $C = (\mu', w'_1, w'_2, \ldots w'_m)$ of the P system Π, iff $\exists i$ such that $r \in R_i$ and the following conditions are fulfilled:*

1. *$u \subseteq w'_i$,*
2. *for any object, which appears in v in the form (a, in_j) there is a membrane j directly inside i,*
3. *if $v = v'\delta$, then i is not the skin membrane.*

Definition 3.1.12. *Suppose a rule $r : u \to v$ that belongs to a set of rules R_i. The* **result of the application of the rule** *r in the configuration $C = (\mu', w'_1, w'_2, \ldots w'_m)$ of the P system Π is determined by the following prescription:*

- *the multiset u is subtracted from w_i;*
- *if an object appears in v in the form $(a, here)$, then it remains in the same region i (the indication here will often be omitted);*
- *if an object appears in v in the form (a, out), then it exits the membrane i and becomes an element of the region immediately outside it. If the membrane i is the skin membrane, then the object leaves the system and it will never come back;*
- *if an object appears in v in the form (a, in_j) and j a membrane immediately inside i, then the object a is added to the multiset w'_j. If it is not a membrane immediately inside i, then the application of the rule is not allowed;*

- *if the symbol δ appears in v, then the membrane i is removed (we say it is dissolved) from μ' and, at the same time, the multiset w'_i is added to the region immediately outside of the membrane i. The dissolving of the skin membrane is not allowed.*

For better understanding of this process, we present an algorithm for the rule application below.

Algorithm 2 Application of a single rule in a P system

1: **procedure** RULEAPLICATION(applicable rule $u \to v \in R_i$, configuration $C = (\mu', w'_1, w'_2, \dots w'_m)$)
2: $w_i := w_i - u$
3: **for all** $(a, here) \in v$ **do**
4: $w_i := w_i + a$
5: **for all** $(a, out) \in v$ **do**
6: $w_{parent(i)} := w_{parent(i)} + a$
7: **for all** $(a, in_j) \in v$ **do**
8: $w_j := w_j + a$
9: **if** $v = v'\delta$ **then**
10: $w_{parent(i)} := w_{parent(i)} + w_i$
11: $w_i :=$ empty multiset

All these operations are performed in a maximal parallel mode: inside each region all applicable rules $u \to v$ are used over all occurrences of multisets u, and all regions are processed at the same time. Hence, the parallelism is maximal at both a global and a local level: the global level involves all membranes at the same time, the local level involves all evolution rules and all objects inside each region.

Definition 3.1.13. *A multiset of rules*

$$\rho : \bigcup_{1 \le i \le m} R_i \to \mathbb{N}$$

is **simultaneously applicable** *in the configuration $C = (\mu', w'_1, w'_2, \dots w'_m)$ of the P system Π, iff the following conditions are fulfilled:*

1. *for any membrane with label $1 \le i \le m$, there is enough objects to allow simultaneous application of rules:*

$$\bigcup_{(r:u \to v) \in supp(\rho) \cap R_i} \rho(r) \cdot u \subseteq w'_i$$

,

2. *for any membrane with label* $1 \leq i \leq m$, *any rule* $u \to v \in supp(\rho) \cap R_i$, *any object, which appears in* v *in the form* (a, in_j) *there is a membrane* j *directly inside* i,

3. *for the skin membrane* i, *any rule* $r : u \to v'\delta \in R_i$ *has no occurence in the multiset:* $\rho(r) = 0$.

The multiset of simultaneously applicable rules is maximal iff adding more rules to the multiset make it not simultaneously applicable. Simultaneous application of a multiset of rules arises no contradictions:

- if there is both (a, out) and δ present in v, the object a ends up in the parent region,
- if there is a rule from R_i with v containing (a, in_j) and a rule from R_j with v containing δ, the object a ends up in the region i

Definition 3.1.14. *A* **computation step** *of a P system is a relation* $\Rightarrow$ *on the set of configurations such that* $C_1 \Rightarrow C_2$ *iff there is a nonempty maximal multiset of simultaneously applicable rules, which, applied to* C_1, *can result in* C_2.

Because there can be more than one maximal multiset, for a configuration C_1 there can be multiple possible results C_2, so one of them is nondeterministically chosen.

Starting from the initial configuration and applying the rules in the way described above, we obtain a sequence of transition among configurations; such a sequence is called a computation of Π.

Definition 3.1.15. *An* **infinite computation** *of a P system is an infinite sequence of configurations* $\{C_i\}_0^\infty$ *such that* C_0 *is the initial configuration and* $C_i \Rightarrow C_{i+1}$ *for all* $0 \leq i$.

Definition 3.1.16. *A* **finite computation** *of a P system is a sequence of configurations* $\{C_i\}_0^n$ *such that* C_0 *is the initial configuration and* $C_i \Rightarrow C_{i+1}$ *for all* $0 \leq i < n$.

Definition 3.1.17. *A* **halting configuration** C *of a P system is a configuration with no applicable rule.*

When no rule is applicable, there is only one maximal multiset of simultaneously applicable rules: an empty multiset, so there is no configuration C' such that $C \Rightarrow C'$.

Definition 3.1.18. *A* **halting computation** *of a P system is a finite computation ending with a halting configuration.*

There are multiple possible ways of assigning a result of a computation [12]. We present two of them:

1. By considering the multiplicity of objects present in a designated membrane in a halting configuration. In this case we obtain a vector of natural numbers. We can also represent this vector as a multiset of objects or as Parikh image of a string.

2. By concatenating the symbols which leave the system, in the order they are sent out of the skin membrane. If several symbols are expelled at the same time, then any ordering of them is assumed. In this case we generate a string.

The result of a computation is clearly only one multiset or a string, but for one initial configuration there can be multiple possible computations. It follows from the fact that there exist more than one maximal multiset of rules that can be applied in each step. Aggregating the results of possible computations a P system defines a language or a Parikh image of a language.

Example 3.1.1. The P system in figure 3.1 computes a Fibonacci sequence. Maximal parallelism enforces the simultaneous movement of objects between regions 1 and 2. Every second step the number of occurences of the object e in the skin membrane constitutes the following item in Fibonacci sequence. The P system has only one computation, which is infinite.

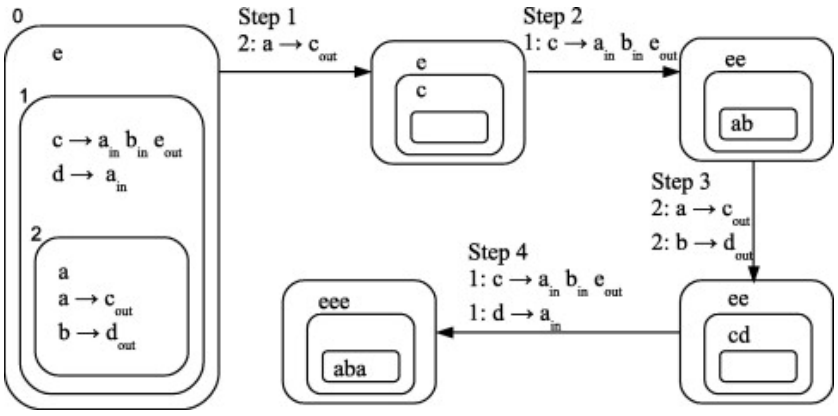


Figure 3.1: P system computing a Fibonacci sequence [15]

3.2 P system variants

Besozzi in his PhD thesis (see [12]) formulates three criteria that a good P system variant should satisfy:

1. It should be as much realistic as possible from the biological point of view, in order not to widen the distance between the inspiring cellular reality and the idealized theory.

2. It should result in computational completeness and efficiency, which would mean to obtain universal (and hence, programmable) computing devices, with a powerful and useful intrinsic parallelism.

3. It should present mathematical minimality and elegance, to the aim of proposing an alternative framework for the analysis of computational models.

In membrane computing, many models are equal in power with Turing machines. To be correct, we should speak about Turing completeness (or computational completeness). Because the proofs are always constructive, using the constructions from these proofs on a universal Turing machine or from a equivalent device, we obtain universal P systems, which are able to simulate any other P system of the given type. That is why we often speak about universality results, and not about computational completeness.

3.2.1 Accepting vs generating

In the Chomsky hierarchy, there are language acceptors (finite automata, Turing machines) and language generators (formal grammars).

Bordhin in [13] extends grammars to allow for accepting languages by interchanging the left side with the right side of a rule. The mode will apply rewriting rules to an input word and accept it when it reaches the starting nonterminal. However, the input word consists of terminal symbols, which could not be rewritten when using original definition, hence they consider the pure version of various grammar types where they give up the distinction between terminal and nonterminal symbols.

The regular, context-free, context-sensitive and recursively enumerable languages were shown to have equal power in accepting and generating mode. Some other grammars (programmed grammars with appearance checking) are shown to be more powerful in accepting mode than in generating mode. For deterministic Lindenmayer systems, the generating and accepting mode are incomparable.

It can be interesting to investigate accepting and generating mode also in P system variants. The original P system from section 3.1 defines the language as a generator. The accepting mode is usually defined as in [12]. A fixed membrane is specified as the input membrane. The system is said to accept a multiset (the initial contents of the input membrane), a number (the size of the input membrane) or a relation (various combinations of the occurrences of certain symbols in the input membrane) iff there is a computation that reaches a given halting configuration (or just a configuration).

In many variants the generating and accepting mode are equivalent. Barbuti in [10] showed some variants where the P system in accepting mode is strictly stronger than in generating mode.

3.2.2 Active vs passive membranes

Most of the studied P system variants assume that the number of membranes can only decrease during a computation, by dissolving membranes as a result of applying evolution rules to the objects present in the system [84]. A natural possibility is to let the number of membranes also to increase during a computation, for instance, by division, as it is well-known in biology. Actually, the membranes from biochemistry are not at all passive, like those in the models briefly described above. For example, the passing of a chemical compound through a membrane is often done by a direct interaction with the membrane itself (with the so-called protein channels or protein gates present in the membrane); during this interaction, the chemical compound which passes through membrane can be modified, while the membrane itself can in this way be modified (at least locally).

In [84] Păun considers P systems with active membranes where the central role in the computation is played by the membranes: evolution rules are associated both with objects and membranes, while the communication through membranes is performed with the direct participation of the membranes; moreover, the membranes can not only be dissolved, but they also can multiply by division. An elementary membrane can be divided by means of an interaction with an object from that membrane.

There are also other ways to define the creation of new membranes. In section 4.2 we present a variant with a special evolution rule which creates a new membrane when a given multiset of objects is present in the region.

P systems with polarized membranes

Polarizing membranes [84] is a special extension of active P systems, where each membrane is supposed to have an electrical polarization (we will say

charge), one of the three possible: positive, negative, or neutral. If in a membrane we have two immediately lower membranes of opposite polarizations, one positive and one negative, then that membrane can also divide in such a way that the two membranes of opposite charge are separated; all membranes of neutral charge and all objects are duplicated and a copy of each of them is introduced in each of the two new membranes. The skin membrane is never divided. If at the same time a membrane is divided and there are objects in this membrane which are being rewritten in the same step, then in the new copies of the membrane the result of the evolution is included.

In this way, the number of membranes can grow, even exponentially. As expected, by making use of this increased parallelism we can compute faster. For example, the SAT problem, which is NP complete, can be solved in linear time, when we consider the steps of computation as the time units. Moreover, the model is shown to be computationally universal.

3.2.3 Cooperative vs non-cooperative

If a P system contains evolution rules whose radius is greater than 1, then it is said to be cooperative, otherwise it is a non-cooperative system. Non-cooperative P systems have all rules restricted to contain only one object on the left side. P systems with cooperative rules are universal [77], while P systems with non-cooperative rules only characterize Parikh image of context-free languages ($PsCF$) [94]. It seems to relate with context-free and context-sensitive grammars in Chomsky's hierarchy mentioned in section 2.3.

Păun [77] also defines catalytic P systems where catalysts are a specified subset of the alphabet $C \subseteq \Sigma$. In a catalytic system the only possible rule of radius greater than 1 is of the form $ca \to cv$, where $c \in C, a \in \Sigma \setminus C$ and $v \in (\Sigma \setminus C)^*$ and that is the only form of a rule that can contain a catalyst. If $C = \emptyset$ then the system is called non-catalytic. Catalysts are not modified by applying the catalytic rule. Surprisingly, P systems with catalytic rules are universal, even two membranes in the P system are sufficient to achieve universality.

Freund in [38] showed that two catalysts and one membrane are enough for universality and raised an open problem whether one catalyst is sufficient. He conjectured that for computationally universal P systems the results obtained in this paper are optimal not only with respect to the number of membranes (2), but also with respect to the number of catalysts.

From some point of view, catalysts are way too powerful in restricting the parallelism - they directly participate in the rules, hence the number of catalytic rules that can be applied in one step, is bounded by number of

catalysts.

A variant with promoters and inhibitors have been proposed (see [51]).

In the case of promoters, the rules are possible only in the presence of certain symbols. An object p is a promoter for a rule $u \to v$ and we denote this by $u \to v|_p$, if the rule is applicable only in the presence of object p. Note that unlike in the case with catalysts, promoters allow the associated rules to be applied as many times as possible.

If an object i is an inhibitor for a rule $u \to v$, we denote this by $u \to v|_{\neg i}$. Such rule is applicable as long as inhibitor i is not present in the region.

Ionescu in [51] showed that P systems with non-cooperative catalytic rules with only one catalyst and with promoters / inhibitors are universal. One of our results (see section 4.1) uses inhibitors to achieve universality for sequential P systems.

Non-cooperative rules with no catalysts and with inhibitors were studied in [95], the equivalence with the Parikh image of Lindenmayer systems ($ET0L$ as defined in section 2.5) was proved.

Dang [48] proposes a simple cooperative system (SCO) as a P system where the only rules allowed are of the form $a \to v$ or of the form $aa \to v$, where a is an object and v is a multiset of objects not containing a. This variant is investigated with various modes of parallelism, so their results will be mentioned in the subsection 3.2.7

Another interesting variant is where the only rules are the catalytic rules of the form $ca \to cv$, so there are no rules of radius 1. This variant is called purely catalytic systems and was introduced by Ibarra in [49]. The computational completeness was shown to hold with just three catalysts [38], while when initialized with just one catalyst, the equivalence with Petri nets has been proven [47].

3.2.4 Rules with priorities

In the original definition of a P system [77], a partial order relation over set of rewriting rules have been specified. The rule can be used only if no rule of a higher priority in the region can be applied at the same time.

Sosík in [97] showed that the priorities may be omitted from the maximal parallel model without loss of computational power, thus maintaining computational completeness.

Sequential P systems (see subsection 3.2.7) with prioritized rules are universal [50], while without the priorities they are not.

As mentioned in subsection 3.2.3, P systems with non-cooperative rules only characterize the Parikh image of context-free languages ($PsCF$) [94], it was an open problem, whether priorities could improve the computational

power. In [93] Sburnan showed that the variant with prioritized rules is more powerful and characterize exactly the Parikh image of $ET0L$ (see section 2.5 and subsection 2.7.1 in preliminaries).

We may conclude here that adding a priority relation over rules is a powerful feature which often boosts the computational power by allowing finer control mechanisms and limiting otherwise uncontrolled nondeterministic selection of rules to be applied.

3.2.5 Energy in P systems

Various notions of energy has been proposed for use in P systems. Păun in [85] considers a P system where each evolution rule "produces" or "consumes" some quantity of energy, in amounts which are expressed as integer numbers. In each moment and in each membrane the total energy involved in an evolution step should be positive, but if "Too much" energy is present in a membrane, then the membrane will be destroyed (dissolved). This variant was investigated in two cases, both were shown to be universal:

1. when using only two membranes and unbounded amount of energy,
2. when using arbitrarily many membranes and a bounded energy associated with rules

Freund in [39] introduced a new variant where the rules are assigned directly to membranes (every rule consume objects on one side of the membrane and produce objects on the other side) and every membrane carries an energy value that can be changed during a computation by objects passing through the membrane.

This variant is universal even in sequential mode if we allow priorities on the objects. When omitting the priority relation, only the family of Parikh sets generated by context-free matrix grammars ($PsMAT$ as defined in section 2.4) is obtained.

3.2.6 Symport / antiport rules

Păun in [76] proposes a new way of communicating between membranes.

Symports allow two chemicals to pass together through a membrane in the same direction using symport rules of type (ab, in) or (ab, out). Antiports allow two chemicals to pass simultaneously through a membrane in opposite directions using antiport rules of type $(a, in; b, out)$.

A P system with symport/antiport rules uses only these two types of rules and nothing else. Surprisingly, this P system variant has been shown

to be computationally complete. Five membranes are enough for this result as shown in [76]. If more than two chemicals may collaborate when passing through membranes, two membranes are sufficient for universality.

Dang in [27] showed some results about the sequential P systems with symport/antiport rules. When working in just one membrane, its reachability set is the same as of vector addition systems. Thus it is not universal in contrast to the maximal parallel variant, which is universal.

3.2.7 Parallelism options

Original definition of P system (see [77]) uses maximal parallelism when doing a step of computation. There is an obvious biological motivation relying on the assumption that "if we wait long enough, then all reaction which may take place will take place". This condition is rather powerful, because it decreases the non-determinism of the system's evolution. For various reasons ranging from looking for more realistic models to just the mathematical challenge, the maximal parallelism was questioned.

Dang in [27] investigates the sequential mode. In each step, from the set of applicable rules across all membrane one is nondeterministically chosen and applied.

Definition 3.2.1. *A* **computation step of a sequential P system** *is a relation $\Rightarrow$ on the set of configurations such that $C_1 \Rightarrow C_2$ holds iff there is an applicable rule in a membrane in C_1 such that applying that rule can result in C_2.*

For purely catalytic systems with 1 membrane, the sequential mode generates only the semilinear sets and thus is strictly weaker than the maximally parallel version. Sequential version of symport / antiport systems are equivalent to vector addition systems making it strictly weaker than the original maximally parallel version.

Investigation of the sequential mode continues in [50]. Sequential P system without priorities with cooperative rules with rules for membrane dissolution are not universal by showing they can be simulated by vector addition systems with states (VASS). This holds even when the membrane creation is allowed for bounded number of created membranes. However, if any number of membranes are allowed to be created, the system becomes universal. This result was shown by simulation of the register machine (see section 2.8).

We have further investigated this universal sequential variant in Chapter 4 where we will show some of our interesting results on the decidability problems of existence of finite / infinite computation.

Even though the variant without rules for creation of new membranes is not universal, we have studied the effect of adding inhibitors, which was shown to be universal.

Dang in [48] proposes several restricted versions of parallelism. n-Max-Parallel version nondeterministically selects a maximal subset of at most n rules to apply. It is proved that 9-Max-Parallel SCO (defined in the subsection 3.2.3) is universal. $\leq n$-Parallel version is similar, but does not require the condition of a maximal subset of rules. It is shown to be weaker than n-Max-Parallel version. n-Parallel version requires the size of the subset of rules to apply to be exactly n. All three versions are equal to the sequential mode when $n = 1$. For non-universality results, Dang showed the variants are able to be simulated by vector addition systems.

Ciobanu in [22] proposes even more restricted version of parallelism: minimal parallelism. For each region, if at least one rule can be applied, then at least one rule will be applied. The symport / antiport rules variant has been shown to be universal. In [23] the results are extended to a variant with active membranes with unbounded membrane creation and polarisations, which was shown to be universal.

Freund in [37] studied the asynchronous mode of P systems, where in each step, arbitrary many rules can be applied. The application of rules is hence done in parallel way, but are not synchronized or somewhat controlled.

Obviously, for P systems without priorities this is equivalent with just letting them work in the sequential mode, but for P systems with priorities this observation would not be true any more, because the outcome of performing one rule might affect the applicability of another rule.

3.2.8 Toxic objects

Some of the P system variants are aimed at increasing modeling power. Introducting toxic objects [6] does not increase computational power nor modify the decidability of behavioral properties, but such P systems allow for smaller descriptional complexity, especially for smaller number of rules. Moreover, they have a clear direct biological motivation. Toxic objects are a specified subset of alphabet. They must not stay idle as otherwise the computation is abandoned without yielding a result. Alhazov in [6] used toxic objects to improve the known results for catalytic and purely catalytic P systems by significantly reducing the number of rules needed for the constructions in proofs.

3.2.9 Variants inspired by reaction systems

Reaction systems (section 2.14) deal with two assumptions made about the chemistry of a cell.

1. We assume that we have the "threshold" supply of elements. Either an element is present and then we have "enough" of it, or an element is not present. Therefore we deal with a qualitative rather than quantitative calculus.

2. We do not have the "permanence" feature in our model. If nothing happens to an element in P systems, then it remains / survives (status quo approach). On the contrary, in reaction systems, an element remains / survives only if there is a reaction sustaining it.

We will present variants of P systems inspired by these assumptions in following subsubsections.

Set membrane systems

Standard models of membrane systems have configurations, where any given compartment is represented as a multiset of objects and each computational action is represented by a multiset of simultaneously executed (multiple copies of) individual evolution rules.

Such strong reliance on counting (through multiple copies of objects and rules) may lead to potential problems in two respects. First, one may wonder how realistic is the counting (multiset) mechanism if one needs to represent huge numbers of molecules and instances of biochemical reactions. Second, a membrane system would normally have an infinite state space, making the application of formal verifiation techniques impratical or indeed impossible (there exists a rich body of results proving Turing completeness of even very simple kinds of membrane systems).

A radical solution to the state space problems can be provided by reaction systems (section 2.14), which, however, model biochemial reactions in living cells using qualitative (based on presence and absence of entities) rather than quantitative rewriting rules.

Set membrane systems [57] are based on sets (of objects or rules) together with the associated set operations, rather than on multisets and multiset operations.

Definition 3.2.2. *A* **set membrane system** *is a tuple*

$$\Pi = (\Sigma, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$$

, such that Σ is a finite set of objects, μ is a membrane structure with m membranes, $w_i \subseteq \Sigma$ is a set of objects initially present in region i and R_i is a finite set of evolution rules of the form $u \to v$, where $u \subseteq \Sigma$ is a nonempty set of objects and $v \subseteq \Sigma_i$ is a set of indexed objects with Σ_i being defined as:

$$\Sigma_i = \Sigma \cup \{a_{out} | a \in \Sigma\} \cup \{a_{in_j} | a \in \Sigma \text{ and } i = parent(j)\}$$

.

The difference between the "qualitative" and the "quantitative" interpretation of the evolution rules is twofold. First, there may be two enabled evolution rules in a compartment with a common object in their left hand sides while there is only a single representant of that object in the current state in the compartment. In the current qualitative setup, the two rules can be executed together. That is, objects are charaterized by their presence rather than their quantity. Second, if two simultaneously executed rules produce the same object in the same compartment, instead of adding two instances of this object, only one is added, so that we never have more than a single representant of an object in any given compartment.

Kleijn in [57] describes different ways to define a computation step. An interesting property of maximal parallel steps is that there is always exactly one maximal parallel set of enabled rules, thus such system is deterministic.

Alhazov in [5] proposed P systems, where the multiplicities of objects are ignored, which is essentially the same as set membrane systems. He proved that with bounded number of membranes they have a very limited computing power, exactly the Parikh images of regular languages. On the other hand, allowing membrane creation or division implies the computational completeness.

P systems with no permanence

The second assumption of the reaction systems theory is much easier to handle in terms of membrane computing. If we want to preserve an object a which is not evolving, we may provide a dummy rule for it, of the type $a \to a$, changing nothing. Still, many technical problems appear in this framework. The presence of such dummy rules makes the computation endless, while halting is the "standard" way to define successful computations in membrane computing. Moreover, the rules are nondeterministically chosen, hence the dummy rules can interfere with the "computing rules".

While the second difficulty is a purely technical one, the first one can be overpassed by considering other ways of defining the result of a computation in a P system, and there are many suggestions in the literature. Three of them are assumed in [78].

1. The local halting (the computation stops when at least one membrane in the system cannot use any rule).

2. Signal-objects (the result consists of the number of objects in a specified membrane at the moment when a distinguished object appears in the system).

3. Signal-events (the result consists of the number of objects in a specified membrane at the moment when a distinguished rule is used in the system).

The paper [78] includes a proof of computational completeness for the variant with cooperative rules along with an open problem for the variant with catalytic rules.

3.3 Case studies

Although P systems were inspired by the functioning of a biological cell, the applications has already far exceeded the scope of a cell. P systems are already being used in many problems ranging from modeling biological processes and population dynamics through social network problems to image processing and effectively solving hard problems.

3.3.1 Membrane computing simulator

MeCoSim (Membrane Computing Simulator) [83] is a software that offers the users a General Purpose Application to model, design, simulate, analyze and verify different types of models based on P systems. Some of the main features of MeCoSim are the following:

- Simulation of models of P systems under different initial conditions. It enables the load of P-Lingua based models, parsing, edition, debugging, and different simulation types.

- Visualization capabilities for analyzing P systems: alphabet, membrane structure, multisets and graphs viewers.

- Highly customizable platform for defining inputs, outputs, parameters and graphs for each model of a family of P systems.

- Repositories system for the visual management of available online resources, including plugins, custom applications, models and scenarios.

- Export option for releasing end-user applications for solving practical problems, abstracting P system functionalities.
- Plugins architecture, permitting the extension of the functionality with Java jars or external non-Java programs. MeCoSim is written in Java.
- Auto-update capability, using the latest release of the program whenever it runs.

There is an online collection of various case studies on the MeCoSim webpage [72]. The number of scenarios and the level of detail vary among the different case studies, possibly including detailed descriptions, references to related publications, charts and videos.

MeCoSim development started in 2010, and a number of case studies have been analyzed since then, covering many variants of P systems, different areas of interest and application domains. The examples contain, at least, some snippets of code, along with the minimal needed files to run some scenario in MeCoSim.

3.3.2 Population dynamics

The Bearded Vulture (Gypaetus barbatus) is an endangered species in Europe that feeds almost exclusively on bone remains of wild and domestic ungulates. Spanish researchers in [17] presented a model of an ecosystem related to the Bearded Vulture in the Pyrenees (NE Spain), by using P systems. The evolution of six species is studied: the Bearded Vulture and five subfamilies of domestic and wild ungulates upon which the vulture feeds. P systems provide a high level computational modeling framework which integrates the structural and dynamic aspects of ecosystems in a comprehensive and relevant way. P systems explicitly represent the discrete character of the components of an ecosystem by using rewriting rules on multisets of objects which represent individuals of the population and bones. The inherent stochasticity and uncertainty in ecosystems is captured by using probabilistic strategies.

This research was extended in [16] where they also proposed model for exotic invasive species of Zebra Mussel for the design of a plan for the management of the reservoir of Ribarroja in the area of Ebro River basin because this species create significant environmental problems and economic costs due to its ability to block all types of infrastructure and water pipes.

3.3.3 Cellular Signalling Pathways

Perez in [80] proposed a model for EGFR Signalling Cascade. Cellular signalling pathways are fundamental to the control and regulation of cell behaviour. More than 60 proteins and 160 chemical reactions were included in the model. Membrane structure consists of 3 regions: the environment, the cell surface and the cytoplasm.

3.3.4 Solving SAT in linear time

The discovery of the NP-Complete problems has created a vigorous industry producing proofs of NP-Completeness [2]. But does NP-Completeness really imply intractability of interesting instances? This question seems to divide the Computer Science community into two camps, the pessimists who believe that NP-Complete problems require exponential resources and are therefore intractable and the optimists who expect a subexponential algorithm to emerge eventually. There is however a viable third belief, namely that the resources required do rise exponentially but slowly enough for large practical problems to be tractable.

A number of algorithms have been reported, using an approach more sophisticated than the obvious search of all possible solutions to achieve run times significantly less than the naive approach would suggest, but on a sequential machine these do not yet seem to solve real large problems in reasonable time. On the other hand the naive approach to most NP-Complete problems parallelises easily but yields an algorithm which is too slow even on large parallel machines. However, where the sophisticated algorithms can be parallelised, the result may well be that very large problems which have appeared intractable can be solved quite easily with machines which exist today and will be cheap tomorrow.

Polynomial time solutions to NP-complete problems are usually achieved by trading time for space [98].

P systems with their inherent parallelism seem to be a natural choice due to the capabililty of cells to produce an exponential number of new membranes in polynomial time. However, many simulators of P system are inefficient since they cannot handle the parallelism of these devices. Nowadays, we are witnessing the consolidation of the GPUs as a parallel framework to compute general purpose applications such as bitcoin mining.

The simulation of P systems with active membranes using GPUs is analyzed in [19] and an efficient linear solution to the SAT problem is illustrated. They compared it to the classical simulator and reported up to 94x of speedup for 256 literals in the formula. The major constraint of this parallel simula-

tion is the GPU memory size, which can be overcome with a data partition on a cluster of GPUs.

As time passes, GPU mining of bitcoins is largely dead these days. The majority is now mining with Application Specific Integrated Circuits (ASIC) [96]. The proposal of ASIC for P systems could be the possible aim of future research of P system simulators.

3.3.5 Implementation of P systems in vitro

"In vitro" are studies in experimental biology that uses components of an organism that have been isolated from their usual biological surroundings in order ot provide a more convenient analysis.

This topic could let to the construction of computers based on P systems, and this kind of experiments could be for P systems what are DNA experiments for DNA computing: in vitro use of molecules to calculate. The speed of these processes occurring at membranes (both artificial and natural) is much higher as compared with DNA computing experiments, providing us with a faster computation [7].

There was a planned experiment of computing the Fibonacci sequence using P systems in vitro (see [41]) using test tubes as membranes and DNA molecules as objects, evolving under the control of enzymes. Number of objects in a multiset was represented by a pre-defined "mole" of the substance and synchronization was obtained by "waiting enough", such that all reactions that can take place in a test tube actually take place. Hence, they reuqired a variant where reactions do not cycle and proposes the Local loop-free P systems, which were shown to be universal. Communication was done by moving all the relevant objects to the next tube in a mechanical way. The final result was read by spectrometry.

Many questions and open problems arised from this planned experiment, e.g. if there is any chance to effectively solve NP-Complete problems in this framework.

There are various opinions on the possibility of computer made of real biological cells. As for now it seems too difficult to grasp. Probably, there are also physical constrains with respect to the physical stability in time of a such complicated proteic structure, as compared with the physical stability of a silicon based component now commonly used in computers. Here, we should not to forget that first bulbs made by Edison had a rather short working life too. The ability of scientists to design chemicals, proteins, with changed desired chemical and physical characteristics has improved significantly in the last decade, and the hardware of a P systems based computer probably needs even more progress in this demiurgic (thus dangerous) activity.

Chapter 4

On the edge of universality of sequential P systems

In previous Chapter we defined P systems and presented the current state of the research of the P system variants. Especially the parallelism options have been investigated more thoroughly, but there are still some gaps that should be filled. For many variants the computational completeness has been proven only when working in the maximally parallel mode. In most cases the sequential mode is strictly weaker, but this is not always true.

In this Chapter we investigate several variants of sequential P systems in terms of computational power and decision power of behavioral properties.

Inhibitors in section 4.1 increase the computational power of sequential P systems. Our proof uses a simulation which shows a way how to transform any maximal parallel P system to a sequential one with inhibitors.

Allowing creation of new membranes increases the computational power of seqeuential P systems [50] and we inspect some behavioral properties in section 4.2.

In section 4.3 we propose new variants aiming to fill the gap of computational power between sequential P systems and computational completeness.

Inspired by reaction systems we propose new semantics for membrane creation and study their effect on computational power in section 4.4.

Three of these results were published in conference proceedings [59], [60] and [61].

4.1 Inhibitors

A sequential variant without priorities and with cooperative rules is not universal (see [48]). They have tried modifying the variant to increase the com-

putational power and showed that with rules for membrane creation with unbounded number of membranes it became universal.

We have tried another approach using rules with inhibitors. We show that this variant is computationally complete in both generating and accepting case. For the generative case we present a proof by a simulation of maximal parallel P system and in the accepting case we prove it can simulate a register machine. These results were published in [59].

In the literature there are two variants of rules with inhibitors.

1. Some of them ([51], [94]) allow to use only one inhibitor per rule, e.g. $u \to v|_{\neg i}$.

2. Others ([4], [95]) are more expressive allowing to use a set of inhibitors per rule, e.g. $u \to v|_{\neg B}$, where B is a set of objects. Such a rule can be applied only if no element of B is present in the region, where the rule is applied.

Our proof uses variant of rules with a set of inhibitors, but they could also be implemented with rules with a single inhibitor, while the complexity of the simulation would be increased. Dissolution could be also solved with increased complexity by introducing additional phases of the simulation. But because P systems without dissolution are still computationally complete [4], and for the sake of simplicity, we simulate only rules without dissolution

Definition 4.1.1. *An* **evolution rule without dissolution** *over an alphabet Σ in a P system with m membranes is a pair (u, v), which is usually written as $u \to v$, where u is a string over Σ and v is a string over $\Sigma \times (\{here, out\} \cup \{in_j | 1 \leq j \leq m\})$.*

Definition 4.1.2. *An* **evolution rule with inhibitors without dissolution** *is a triple (u, v, B), which is usually written as $u \to v|_{\neg B}$, where (u, v) is an evolution rule without dissolution by definition 4.1.1 and $B \subseteq \Sigma$ is a set of objects called inhibitors of the rule.*

Theorem 4.1.1. *Any maximal parallel P system with cooperative rules without dissolution can be simulated by a sequential P system with cooperative rules and inhibitors.*

Proof 4.1.1. We show that we can simulate a step of the maximal parallel P system with several steps of a sequential P system with inhibitors.

It is important to note that in the maximal parallel step the evolution occurs simultaneously in all membranes, so we need to synchronize this process. The steps of the sequential P system simulating the maximal parallel step are

divided into several phases. Every membrane will have a phase, represented as an object. There is exactly one phase object in each membrane.

The *RUN* phase represents that the rules of the maximal parallel P system are being applied, one-by-one. When there are no more rules to apply, the membrane has done its maximal parallel step and proceeds to the phase *SYNCHRONIZE*. Other phases are just technical - we need to implement sending objects between membranes and preparing for the next maximal parallel step by unmarking newly created objects in the current maximal parallel step, which have been marked to prevent double evolution in one step.

- **RUN**: Rules of the maximal parallel P system are being applied. Products are marked (a' instead of a) in order to prevent further evolution in the same maximal parallel step. Objects that are to be sent to the parent membrane are directly sent because the parent membrane is in *RUN* or *SYNCHRONIZE* phase (see figure 4.1), so the a' symbols that are sent will not be used in any rule until phase *RESTORE*. But objects that are to be sent down, cannot be sent immediately because child membranes can be in the previous phase waiting to restore symbols from previous step. Objects a' could interfere with them resulting in double application in current maximal parallel step. Such objects are only marked as $a^{\downarrow'}$ to be sent down in the phase *SENDDOWN*. When the phase *RUN* ends, next phase *SYNCHRONIZE* takes place, while emitting an object *SYNCTOKEN* to the parent membrane with a purpose to deliver it to the skin membrane.

- **SYNCHRONIZE**: There was no applicable rule left so the membrane is waiting to get signal *SYNCED* from the parent membrane to proceed to the next phase. And also resending all *SYNCTOKEN* objects to the parent membrane.

- **SENDDOWN**: Signal *SYNCED* was caught, which means that every membrane has already been in the *SYNCHRONIZE* phase. And all descendant membranes are in *SYNCHRONIZE* phase. In this phase all the objects $a^{\downarrow'}$ can be sent down.

- **RESTORE**: All a' symbols are being restored to a, preparing for the next maximal parallel step.

We will now introduce the objects used in the simulation.

- The alphabet Σ of the maximal parallel P system.

- a' for $a \in \Sigma$ used as products of a rule application waiting for the synchronization.
- $a^{\downarrow\prime}$ for $a \in \Sigma$ used as products of a rule application which should be sent down.
- $\dot{a}$ for $a \in \Sigma$ can be used normally in rewriting, with a constraint to have at most one instance of $\dot{a}$ in a membrane. It helps with deciding presence of an applicable rule.
- objects representing phases *RUN*, *SYNCHRONIZE*, *SENDDOWN*, *RESTORE*.
- $SYNCTOKEN_i$ for each membrane with label i to let the skin membrane know the membrane i is waiting in synchronization phase.
- $SYNCED$ used as a broadcast from the skin membrane to let each membrane know that each other membrane has already been synchronized.
- NA_j meaning the rule r_j is not applicable. There is at most one occurence of NA_j in a membrane.

At the start of the simulation the configuration of simulating sequential P system is the same as the configuration of the simulated maximal parallel P system, with a *RUN* symbol in each membrane.

We will now define the evolution rules in a membrane with label i of the simulating P system.

- For every rule $r_j \in R_i$ such that

  $$r_j = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \rightarrow a_1^{N(a_1)} a_2^{N(a_2)} \dots a_n^{N(a_n)}$$

  we will have the following rules:

  $$a_1^{M(a_1)-m_1} \dot{a}_1^{m_1} a_2^{M(a_2)-m_2} \dot{a}_2^{m_2} \dots a_n^{M(a_n)-m_n} \dot{a}_n^{m_n} | RUN$$
  $$\rightarrow a_1'^{N(a_1)} a_2'^{N(a_2)} \dots a_n'^{N(a_n)} | RUN$$

  There will be such rule for each $0 \leq m_k \leq min(M(a_k), 1)$. It represents the idea that $\dot{a_k}$ can be used in rule application in the same way as a_k. The number of occurences of $\dot{a_k}$ is 0 or 1. If $M(a_k) = 0$, the rule r_j has no presence of a_k on the left side so in this case we will have just rules where $m_k = 0$. Right side of the rules contains symbols a', that prevents the symbols to be evolved again.

- For every object $a \in \Sigma$, in order to guarantee at most one occurence of $\dot{a}$ in the membrane, we will have the following rule:

$$a|RUN \rightarrow \dot{a}|RUN|_{\neg\dot{a}}$$

- For every rule $r_j \in R_i$ there will be a rule that detects if the rule r_j is not applicable. If there is not enough objects present in the membrane to satisfy left side of the rule r_j, object NA_i is created.

  If the left side of the rule r_i is of type:

  - a: It is a context free rule. This rule cannot be applied iff there is no occurrence of a nor $\dot{a}$.

$$RUN \rightarrow NA_j|RUN|_{\neg\{NA_j,a,\dot{a}\}}$$

  - ab: It is a cooperative rule with two distinct objects on the left side. This rule cannot be applied iff there is one of them missing.

$$RUN \rightarrow NA_j|RUN|_{\neg\{NA_j,a,\dot{a}\}}$$
$$RUN \rightarrow NA_j|RUN|_{\neg\{NA_j,b,\dot{b}\}}$$

  - a^2: It is a cooperative rule requiring presence of two instances of an object. This rule cannot be applied iff there is at most one occurrence of the object. That happens iff there is no occurrence of a. There can still be $\dot{a}$, but at most one occurrence.

$$RUN \rightarrow NA_j|RUN|_{\neg\{NA_j,a\}}$$

- A rule detecting the maximal set of rules has been applied and no further rule is applicable:

$$NA_1|NA_2|\ldots|NA_{|R_i|}|RUN$$
$$\rightarrow SYNCHRONIZE|SYNCTOKEN_i$$

  If no additional rule can be applied, the simulation of a maximal parallel step in the membrane is complete and it waits for other membranes to complete their maximal parallel step.

- Membrane $i > 1$ (different from the skin membrane) resends synchronization token originating from any membrane l to the parent membrane with a purpose of deliver it to the skin membrane:

$$SYNCHRONIZE|SYNCTOKEN_l$$
$$\rightarrow SYNCHRONIZE|SYNCTOKEN_l \uparrow$$

- In the skin membrane ($i = 1$) there is a rule which collects synchronization tokens from all membranes $1 \dots p$ and then executes the phases *SENDDOWN* and *RESTORE* after which it sends down signal that synchronization is complete.

$$SYNCTOKEN_1 | \dots | SYNCTOKEN_p | SYNCHRONIZE \rightarrow SENDDOWN$$

- Every membrane other than skin membrane ($i > 1$) is waiting for the *SYNCED* signal to go to the senddown phase:

$$SYNCHRONIZE | SYNCED \rightarrow SENDDOWN$$

- Every membrane will have rules to send down all objects prepared to be sent down $a^{\downarrow\prime}$, where $a \in \Sigma$:

$$SENDDOWN | a^{\downarrow\prime} \rightarrow SENDDOWN | a' \downarrow$$

- Every membrane will have a rule for detecting when all such objects have been sent down in order to proceed to the *RESTORE* phase:

$$SENDDOWN \rightarrow RESTORE |_{\neg\{a^{\downarrow\prime} | a \in \Sigma\}}$$

- In the *RESTORE* phase all objects a' will be evolved to a so it can be evolved in the next maximal parallel step:

$$RESTORE | a' \rightarrow RESTORE | a$$

- Descendant membranes are waiting for the signal *SYNCED*, which is sent to child membranes only when the *RESTORE* phase ends. It also means that the next maximal parallel step is already being simulated in the ancestor membranes (*RUN* or *SYNCHRONIZE* phase):

$$RESTORE \rightarrow RUN | SYNCED \downarrow |_{\neg\{a' | a \in \Sigma\}}$$

The pairs of possible phases of the parent and child membrane are shown in the figure 4.1 along with transitions between two consecutive global synchronizations - after the maximal parallel steps i and $i + 1$.

In the figure 4.2 the membrane structure is presented as a hierarchical structure. Every membrane is in one of four phases. It can be seen that the

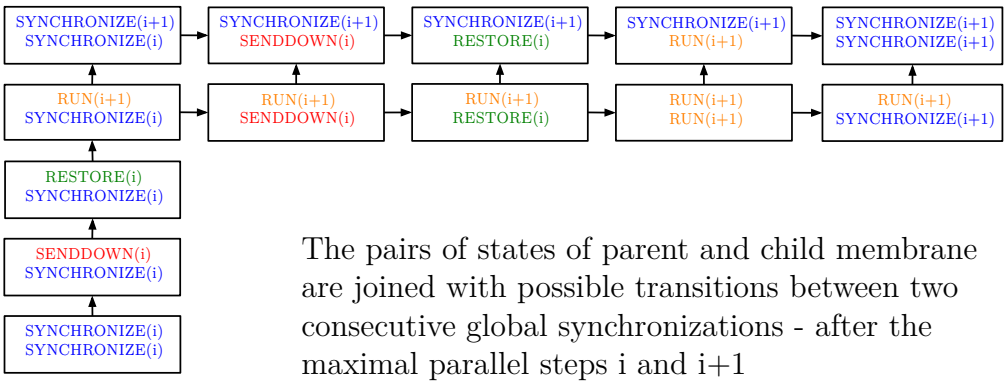


Figure 4.1: Possible pairs of states of parent and child membrane

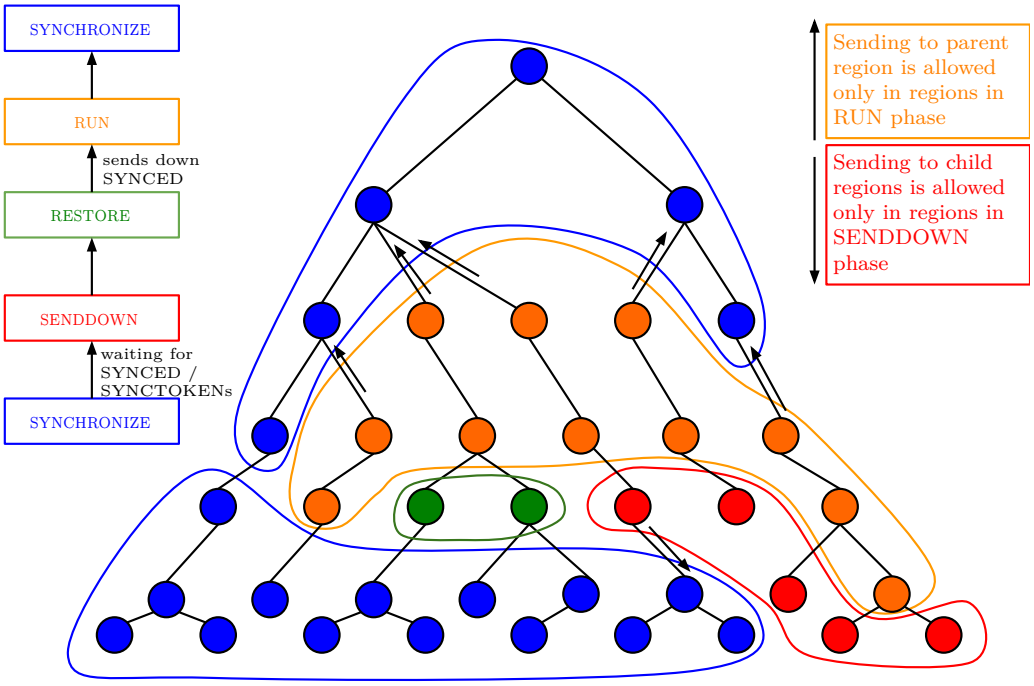


Figure 4.2: Snapshot of all membrane states while simulating

sending of the objects is performed in such phases that the receiving membrane is in either *RUN* or *SYNCHRONIZE* phase, so the received objects (marked a') does not interfere with rewriting.

Another interesting idea can be seen in the figure 4.2 that when a region is in the *SENDDOWN* phase and objects are sent through the child membrane, the receiving region is in the *SYNCHRONIZE* phase waiting for the *SYNCED* signal, which will be sent to it when *SENDDOWN* and *RESTORE* phases finished. □

We have also reached this result in the accepting case by simulation of register machines.

Theorem 4.1.2. *Sequential P systems with cooperative rules and inhibitors can simulate register machines and thus equal $PsRE$.*

Proof 4.1.2. Suppose we have an n-register machine $M = (n, P, i, h, Lab)$. In our simulation we will have a membrane structure consisting of single membrane and the contents of register j will be represented by the multiplicity of the object a_j.

We will simulate the register machine by P system (Σ, μ, w, R), where:

- Σ is an alphabet consisting of instruction labels Lab, n objects that represent registers $(a_1, \ldots a_n)$ and a halting symbol #,
- μ is a membrane structure consisting of one single membrane,
- w is initial contents of the membrane. It contains objects for the input for the machine $a_i^{n_i}$ where n_i is initial state of register with label i and initial instruction label e.
- R is a set of rules in the skin membrane.

For all instructions of type $(e : add(j), k, l)$ we will have rules:

$$e \to a_j|k$$
$$e \to a_j|l.$$

For all instructions of type $(e : sub(j), k, l)$ we will have rules:

$$e|a_j \to k$$
$$e \to l|_{\neg a_j}.$$

And finally halting rules:

$$h|a_j \to h|\# \text{ for all } a \leq j \leq n,$$
$$\# \to \#.$$

For a configuration $(j, m_1, \ldots, m_n)$ of the simulated register machine M the skin membrane of the simulating P system contains a symbol j and objects representing contents of registers $a_1^{m_1}, \ldots, a_n^{m_n}$.

When the halting instruction is reached, if there is an object present in the membrane, the hash symbol # is created and the rule $\# \to \#$ will be applicable forever as there is no rule to remove the symbol #. If there is no object present, there is no rule to apply and computation will halt. It corresponds to the condition that all registers should be empty when halting. □

4.1.1 Concluding remarks

Although these results are not very surprising as similar results already hold for Petri nets, which are equivalent to sequential P systems, the simulation of maximal parallel P system used in the proof for generating case is valuable not only for the universality, but also can be seen as a method of conversion between P systems in sequential manner and maximally parallel manner, which may be essential for future works on P systems and other multiset rewriting systems. The simulation also shows a method how to synchronize application of multiple rules in a membrane and how to synchronize this parallel rule application across whole membrane structure. Sequential variants are promising alternative to traditional maximal parallel variants and will be good subject for the further research. Future plans include research of other more restricted variants such as omitting cooperation in the rules or restricting the power of inhibitors.

4.2 Active membranes

In this section we study a variant of sequential P systems where universality can be achieved without checking for zero by allowing membranes to be created unlimited number of times [50]. Such P systems are called active P systems. Contrary, if we place a limit on the number of times a membrane is created, we get a class of P systems which is only equivalent to vector addition systems, hence not universal. There results were published in [60].

In subsection 4.2.1 we will introduce membrane structure and formally define membrane configuration and active P system, because standard definitions are not convenient for our formal proofs.

The subsection 4.2.2 contains two main results. The existence of an infinite computation is surprisingly shown to be decidable. On the other hand, the existence of a halting computation is shown to be undecidable. Why

is the first result surprising? At first sight it seems to relate to the Rice's theorem (see section 2.13), because it is a decidability problem for a Turing complete computation model. But the existence of an infinite computation is not a property of the language generated or accepted by the active P system. It is the property of the computation itself, therefore the Rice's theorem is not applicable here.

4.2.1 Active P systems

Definition 4.2.1. *Let Σ be a set of objects. We denote by $\mathbb{N}^\Sigma$ a set of all mappings from Σ to $\mathbb{N}$, which represents all multisets of objects from Σ. A* **membrane configuration** *is a tuple (T, l, c), where:*

- *T is a rooted tree,*
- *$l \in \mathbb{N}^{V(T)}$ is a mapping that assigns for each node of T a number (label), where $l(r_T) = 1$, so the skin membrane is always labeled with 1,*
- *$c \in (\mathbb{N}^\Sigma)^{V(T)}$ is a mapping that assigns for each node of T a multiset of objects from Σ, so it represents the contents of the membrane.*

Definition 4.2.2. *An* **active P system** *is a tuple $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, where:*

- *Σ is a set of objects,*
- *C_0 is initial membrane configuration,*
- *$R_1, R_2, \ldots, R_m$ are finite sets of rewriting rules associated with the labels $1, 2, \ldots, m$ and can be of forms:*
  - *$u \to w$, where $u \in \Sigma^+$, $w \in (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})^*$ and $1 \leq j \leq m$,*
  - *$u \to w\delta$, where $u \in \Sigma^+$, $w \in (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})^*$ and $1 \leq j \leq m$,*
  - *$u \to [_j v]_j$, where $u \in \Sigma^+, v \in \Sigma^*$ and $1 \leq j \leq m$.*

Although rewriting rules are defined as strings, u, v and w represent multisets of objects from Σ. For the first two forms, each rewriting rule may specify for each object on the right side, whether it stays in the current region (we will omit the symbol $\cdot$), moves through the membrane to the parent region ($\uparrow$) or to a specific child region ($\downarrow_j$, where j is a label of a membrane). If there are more child membranes with the same label, one is chosen nondeterministically. We denote these transfers with an arrow immediately after the symbol. An example of such rule is the following: $abb \to ab \downarrow_2 c \uparrow c\delta$.

Symbol δ at the end of the rule means that after the application of the rule, the membrane is dissolved and its contents (objects, child membranes) are propagated to the parent membrane. Active P systems differ from classic (passive) P systems in ability to create new membranes by rules of the third form. Such rule will create new child membrane with a given label j and a given multiset of objects v as its contents.

Definition 4.2.3. *For an active P system* $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$*, configuration* $C = (T, l, c)$*, membrane* $d \in V(T)$ *the rule* $r \in R_{l(d)}$ *is* **applicable** *iff:*

- $r = u \to w$ *and* $u \subseteq c(d)$ *and for all* $(a, \downarrow_k) \in w$ *there exists* $d_2 \in V(T)$ *such that* $l(d_2) = k \wedge parent(d_2) = d$*,*
- $r = u \to w\delta$ *and* $u \subseteq c(d)$ *and for all* $(a, \downarrow_k) \in w$ *there exists* $d_2 \in V(T)$ *such that* $l(d_2) = k \wedge parent(d_2) = d$ *and* $d \neq r_T$*,*
- $r = u \to [_j v]_j$ *and* $u \subseteq c(d)$*.*

In this section we assume only sequential systems, so in each step of the computation, there is one rule nondeterministically chosen among all applicable rules in all membranes to be applied as already stated in the definition 3.2.1.

4.2.2 Termination problems

In this subsection we recall the halting problem for Turing machines. The problem is to determine, given a deterministic Turing machine and an input, whether the Turing machine running on that input will halt. It is one of the first known undecidable problems. On the other hand, for non-deterministic machines, there are two possible meanings for halting. We could be interested either in:

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

We can ask the same questions for non-deterministic P systems. For example we can look at the P system from the figure 4.3. Its computation tree (figure 4.4) contains an infinite branch, so there exists an infinite computation with rule r_1 applied in each step. There is also a finite branch, so there exists a finite computation, e.g. applying rule r_2 results in a halting configuration.

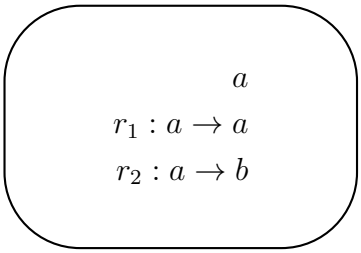


Figure 4.3: A single-membrane sample P system

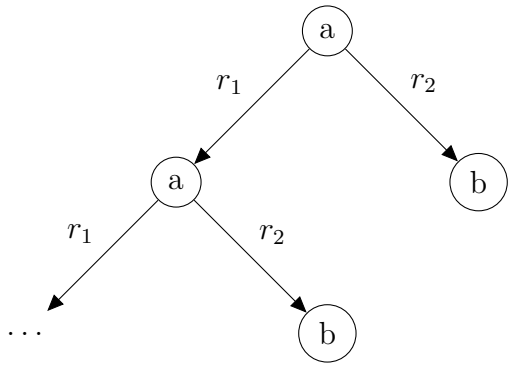


Figure 4.4: The computation tree of the P system from the figure 4.3

4.2.3 Existence of halting computation

In this subsection we will focus on the problem of deciding whether there is a halting computation for sequential active P systems. Recall that halting computation has no applicable rule in the last configuration. We will prove our theorem by two reductions.

Lemma 4.2.1. *Existence of halting computation for nondeterministic register machines is undecidable.*

Proof 4.2.1. Deterministic register machines are a special case of nondeterministic register machines. The only difference is a constraint $k = l$ on the instructions $(add(j), k, l)$. Therefore if we could decide existence of halting computation for nondeterministic register machines, we could also decide it for deterministic case, which is not possible ([69]). □

Theorem 4.2.2. *Existence of halting computation for sequential active P systems is undecidable.*

Proof 4.2.2. According to [50] for a given nondeterministic register machine M there is a sequential active P system P which has a halting computation iff M has. If we could decide existence of halting computation of P, using [50] we could decide existence of halting computation of M, which is undecidable (lemma 4.2.1). □

4.2.4 Existence of infinite computation

In this subsection we will focus on the opposite problem: whether there is a computation that is infinite. Although for register machines the existence of

infinite computation can be reduced from deterministic to nondeterministic mode as in 4.2.1, it cannot be reduced for sequential active P systems as in 4.2.2. In fact, the result will be quite the opposite. Due to technical difficulties we will provide a proof only for active P systems with limit on the total number of membranes.

Active membranes with a limit on the total number of membranes

We will now introduce a variant with a global limit upon the membrane structure. We achieve this by restricting the rule application such that if the rule would result in a structure exceeding the limit, the rule will not be applicable.

Definition 4.2.4. *An* **active P system with a limit on the total number of membranes** *is a tuple* $\Pi = (\Sigma, L, C_0, R_1, R_2, \ldots, R_m)$*, where:*

- $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$ *is an active P system from the definition 4.2.2,*
- $L \in \mathbb{N}$ *is a limit on thetotal number of membranes.*

Anytime during the computation, a configuration (T, l, c) is not allowed to have more than L membranes, so the following invariant holds: $|V(T)| \leq L$. This is achieved by adding a constraint for rule of the form $r = u \rightarrow [_k v]_k$, which is defined to be applicable iff $u \subseteq c(d)$ and $|V(T)| < L$. If the number of membranes is equal to L, there is no space for newly created membrane, so in that case such rule is not applicable.

Definition 4.2.5. *For active P system with a limit on the total number of membranes* $(\Sigma, L, C_0, R_1, R_2, \ldots, R_m)$*, configuration* $C = (T, l, c)$*, membrane* $d \in V(T)$ *the rule* $r \in R_{l(d)}$ *is* **applicable** *iff:*

- $r = u \rightarrow w$ *and* $u \subseteq c(d)$ *and* $\forall (a, \downarrow_k) \in w \exists d_2 \in V(T) : l(d_2) = k \wedge parent(d_2) = d$*,*
- $r = u \rightarrow w\delta$ *and* $u \subseteq c(d)$ *and* $\forall (a, \downarrow_k) \in w \exists d_2 \in V(T) : l(d_2) = k \wedge parent(d_2) = d$ *and* $d \neq r_T$*,*
- $r = u \rightarrow [_j v]_j$ *and* $u \subseteq c(d)$ *and* $|V(T)| < L$*.*

Although we believe the results (theorem 4.2.7) also hold for the variant without limit on the total number of membranes, due to technical difficulties it remains an open problem to determine existence of infinite computation in unbounded case. The variant with limit on the total number of membranes is not very realistic from biological point of view. Assuming membranes to know the number of membranes in the whole system is simply not plausible.

But the results are still quite interesting, because:

Theorem 4.2.3. *Sequential active P systems with limit on the total number of membranes are universal.*

Proof 4.2.3. The proof of this theorem for sequential active P systems in [50] uses simulation of register machines and during the simulation, every configuration has at most three membranes. Hence a universal active P system with limit on the total number of membranes exists (e.g. with $L = 3$). □

We will now propose an algorithm for deciding existence of infinite computation. Basic idea is to consider the minimal coverability graph ([36]), where nodes are configurations and an edge leads from the configuration C_1 to the configuration C_2, whenever there is a rule applicable in C_1, which results in C_2. The construction in [36] is performed on Petri nets, where the configuration consists just of a vector of natural numbers. The situation is the same for single-membrane sequential P systems. We need to modify the construction for active P systems.

Definition 4.2.6. *A configuration* $C_2 = (T_2, l_2, c_2)$ **covers** *configuration* $C_1 = (T_1, l_1, c_1)$ *iff* $\exists$ *isomorphism* $f : T_1 \to T_2$ *preserving membrane labels and contents:* $\forall d \in T_1$ *the following properties hold:* $l_1(d) = l_2(f(d)) \wedge c_1(d) \subseteq c_2(f(d))$. *We will denote this with* $C_1 \leq C_2$.

Example 4.2.1. In the figure 4.5 there are four membrane configuration. $C_1 \leq C_2$, because the membrane structures consists of one membrane, so the corresponding trees are isomorphic. The label is the same and the contents of the membrane in C_1 is a subset of the contents of the membrane in C_2. It does not have to be a proper subset, i.e. $C_1 \leq C_1.C_1$ and C_3 are incomparable, because the label is different, so neither $C_1 \leq C_3$ nor $C_3 \leq C_1$ holds. C_1 and C_4 are also incomparable, because the trees of their membrane structure are not isomorphic.

We will now follow with a proof of an important property of the covering relation - that it maintains rule applicability.

Lemma 4.2.4. *If configuration* $C_2 = (T_2, l_2, c_2)$ **covers** *configuration* $C_1 = (T_1, l_1, c_1)$, *then there is an isomorphism* $f : T_1 \to T_2$ *such that if a rule* r *is applicable in membrane* $d \in T_1$, *then* r *is applicable in* $f(d)$.

Proof 4.2.4. Suppose r is applicable in d. Then the left side u of the rule r is contained within the contents of the membrane $u \subseteq c_1(d)$. Because $C_1 \leq C_2$, then there is an isomorphism $f : T_1 \to T_2$ such that $c_1(d) \subseteq c_2(f(d))$ and then $u \subseteq c_2(f(d))$.

There are three possible forms of the rule r.

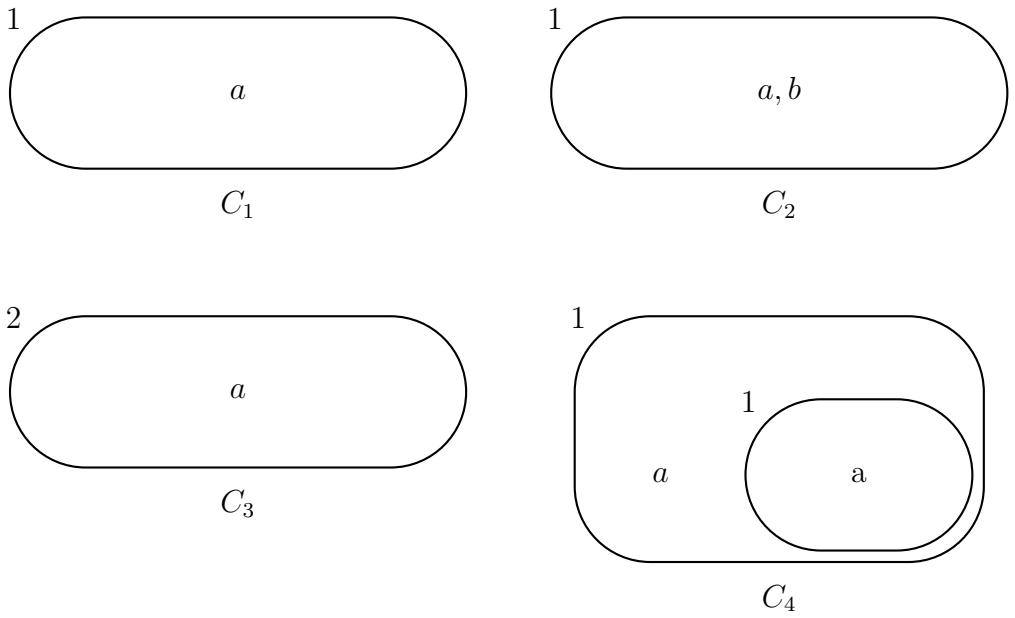


Figure 4.5: Sample membrane configurations

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

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

A membrane $d \in T$ will be encoded as $(n + m)$-tuple $enc(d) \in \mathbb{N}^{(n+m)}$, where first n numbers will be actual counts of objects and next m numbers will encode the membrane label:

$$enc(d)_i = \begin{cases} c(d)(a_i) & \text{if } i \leq n \\ 0 & \text{if } n < i \leq m \wedge i - n \neq l(d) \\ 1 & \text{if } n < i \leq m \wedge i - n = l(d) \end{cases}$$

.

The entire tree will be encoded into concatenated sequences of encoded nodes in the preorder traversal order. This sequence is then padded with

zeroes to have length $(n+m)L$ as that is the maximal length of encoded tree.

Since there are only finitely many non-isomorphic trees with at most L nodes ([18]), there is a constant z such that we can uniquely assign the tree an order number $o(T) \leq z$.

The entire configuration will be encoded in tuple which consists of z parts. All parts except the part with index $o(T)$ will contain just zeros. The part with index $o(T)$ will contain the encoding of the tree. This property ensures that different tree structures are encoded into tuples with nonzero values in different parts making them incomparable.

Example 4.2.2. Consider $L = 2$. There are just two rooted trees with at most 2 nodes. We can define $o(T) = 1$ for the single-node tree and $o(T) = 2$ for the tree with a root and one child. So the encodings of configurations from figure 4.5 contain two parts. Configurations C_1, C_2, and C_3 consists of one membrane, so their $o(T) = 1$ and the second part is filled with zeroes. Configuration C_4 contains two membranes, so its $o(T) = 2$ and the first part of its encoding will be filled with zeroes.

- $enc(C_1) = \overbrace{\underbrace{1}_{c(a)}\underbrace{0}_{c(b)}\underbrace{1}_{l=1?}\underbrace{0}_{l=2?}}^{\text{skin membrane encoded}}\underbrace{0000}_{\text{padding to fit } (n+m)L}\overbrace{00000000}^{\text{second part filled with zeroes}},$
- $end(C_2) = 1110000000000000,$
- $enc(C_3) = 1001000000000000,$
- $enc(C_4) = 00000000\overbrace{1010}^{\text{skin membrane encoded}}\underbrace{1010}_{\text{child membrane encoded}}$

We will now show that comparing two encodings corresponds to covering of two configurations. Recall that configurations are encoded into tuples of integers, so the comparison is performed position by position.

Lemma 4.2.5. *For configurations* $C_1 = (T_1, l_1, c_1)$ *and* $C_2 = (T_2, l_2, c_2)$, $enc(C_1) \leq enc(C_2) \Rightarrow C_1 \leq C_2$.

Proof 4.2.5. Both $enc(C_1)$ and $enc(C_2)$ contain z parts and exactly one part which contains non-zero values. The non-zero part of $enc(C_1)$ must be non-zero also in $enc(C_2)$, because $enc(C_1) \leq enc(C_2)$. Then $o(T_1) = o(T_2)$, so the trees are isomorphic. Suppose there is an isomorphism $f : T_1 \to T_2$. For every membrane $d \in T_1$, $l_1(d) = l_2(f(d))$ and $c_1(d) \subseteq c_2(f(d))$. Hence, $C_1 \leq C_2$. □

Lemma 4.2.6. *For sequential active P system with limit on the total number of membranes L for every infinite sequence of configurations* $\{C_i\}_{i=0}^{\infty} \exists i < j : C_i \leq C_j$.

Proof 4.2.6. Suppose an infinite sequence $\{enc(C_i)\}_{i=0}^{\infty}$. We use a variation of Dickson's lemma ([35]): Every infinite sequence of tuples from $\mathbb{N}^k$ contains an increasing pair. Applied to our sequence, there are two positions $i < j$: $enc(C_i) \leq enc(C_j)$. From lemma 4.2.5, $C_i \leq C_j$. □

Note that the infinite sequence does not have to correspond with a computation, lemma 4.2.6 holds for any infinite sequence of configurations.

Theorem 4.2.7. *Existence of infinite computation for active P systems with limit on the total number of membranes is decidable.*

Proof 4.2.7. The algorithm for deciding the problem will traverse the reachability graph. When it encounters a configuration that covers another configuration, from lemma 4.2.4 follows that the same rules can be applied repeatedly, so the algorithm will halt with the answer YES. Otherwise, the algorithm will answer NO. Algorithm will always halt, because if there was an infinite computation, from lemma 4.2.6 there would be two increasing configurations which is already covered in the YES case. □

4.2.5 Concluding remarks

We have studied the termination problems for active sequential P systems. Unlike deterministic systems, the termination problems cannot be simply reduced to the halting problem. We have shown that active P systems have undecidable existence of halting computation and active P systems with limit on the number of membranes have decidable existence of infinite computation. It is currently unknown whether the same result holds also for a variant without the limit on the number of membranes, so it could be a subject for the future study.

Regarding the open problem stated in [50] about sequential active P systems with hard membranes (without communication between membranes), it could be interesting to find a connection between the universality and decidability of these termination problems.

4.3 Emptiness detection

"In general, it seems that any extension which does not allow zero testing will not actually increase the modeling power (or decrease the decision power) of

Petri nets but merely result in another equivalent formulation of the basic Petri net model. (Modeling convenience may be increased)" [81], page 203.

The above quote from [81] was a fair summary of current beliefs in the Petri net community regarding extensions of the basic Petri net mode: extensions are either Turing-powerful of they are not real extensions.

It is not the case as shown in [33]. They show extensions of Petri nets which do not allow zero testing but that will actually increase the computational power and decrease the decision power (e.g. boundedness becomes undecidable).

In this Chapter we investigate several "weak" extensions of sequential P systems, which allow for zero-testing, aiming to fit in layers between mere reformulations of the basic sequential P system and universal sequential P systems with inhibitors.

We will extend the definition of evolution rules with additional decision option for objects that are being sent through a membrane to another region. Recall the original definition of the evolution rule 3.1.7: $u \to v$, where u is a string over Σ and $v = v'$ or $v = v'\delta$, where v' is a string over $\Sigma \times (\{here, out\} \cup \{in_j | 1 \leq j \leq m\})$ and δ is a special symbol not in Σ. Recall also the algorithm 2, which will be extended in following subsections.

4.3.1 Objects avoiding empty regions

We will have a specific subset of objects, which when occur in a rule in form of (a, out) or (a, in_j), and the target region is empty, they are not sent and stay in the current region instead.

Definition 4.3.1. *P system with objects avoiding empty regions is a tuple*

$$\Pi = (\Sigma, \Gamma, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m),$$

where

$$(\Sigma, \mu, w_1, w_2, \ldots, w_m, R_1, R_2, \ldots, R_m)$$

is a P system and $\Gamma \subseteq \Sigma$ is a subset of objects avoiding empty regions.

We present an algorithm for the rule application on the following page (algorithm 3).

Example 4.3.1. Suppose a P system with objects avoiding empty regions
$(\Sigma = \{a, b\}, \Gamma = \{a\}, \mu = [_1[_2]_2]_1, w_1 = a, w_2 = \varepsilon,$
$R_1 = \{a \to a \downarrow_2 b \downarrow_2\}, R_2 = \{\})$.

The computation starts with an empty region 2 and the object a in the region 1. Application of the rule sends the object b into the region 2 and the

object a stays in the current region because it is avoiding the empty region 2. In the next step, the application of the rule sends both objects into the region 2, because it already contains the object b. The computation finishes with objects abb in the region 2 and empty region 1.

Algorithm 3 Application of a single rule in a P system with objects avoiding empty regions

procedure RULEAPLICATION(applicable rule $u \to v \in R_i$, configuration $C = (\mu, w_1, w_2, \dots w_m)$, set of objects avoiding empty regions $\Gamma \subseteq \Sigma$)
  $w_i := w_i - u$
  for all $(a, here) \in v$ **do**
    $w_i := w_i + a$
  for all $(a, out) \in v$ **do**
    if $a \in \Gamma$ and $parent(i)$ is empty **then**
      $w_i := w_i + a$
    else
      $w_{parent(i)} := w_{parent(i)} + a$
  for all $(a, in_j) \in v$ **do**
    if $a \in \Gamma$ and j is empty **then**
      $w_i := w_i + a$
    else
      $w_j := w_j + a$
  if $v = v'\delta$ **then**
    $w_{parent(i)} := w_{parent(i)} + w_i$
    $w_i :=$ empty multiset

4.3.2 Objects altering when entering empty region

We represent altering the objects entering empty regions with a mapping $\Phi : \Sigma \to \Sigma$. The mapping defines for each object what it will become when sent to an empty region.

Definition 4.3.2. *P system with objects altering when entering empty region is a tuple*

$$\Pi = (\Sigma, \Phi, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m),$$

where

$$(\Sigma, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$$

is a P system and $\Phi : \Sigma \to \Sigma$ is a mapping of objects.

We present an algorithm for this variant of the rule application below (algorithm 4).

Example 4.3.2. Suppose a P system with objects altering when entering empty region ($\Sigma = \{a, b\}, \Phi = \{a \to b, b \to b\}, \mu = [_1[_2]_2]_1, w_1 = a, w_2 = \varepsilon$, $R_1 = \{a \to ba \downarrow_2, b \to a \downarrow_2\}, R_2 = \{\}$).

The computation starts with an empty region 2 and the object a in the region 1. Application of the rule $a \to ba \downarrow_2$ produces an object b which stays in the current region and the second product a is sent into the region 2, which is empty, so the produced a is altered to b. Now both regions contains an object b. In the next step, the application of the rule $b \to a \downarrow_2$ in region 1 sends the produced object a into the region 2, but this time it is not altered because the region is nonempty. The computation finishes with objects ab in the region 2 and empty region 1.

Algorithm 4 Application of a single rule in a P system with objects altering when entering empty region

procedure RULEAPLICATION(applicable rule $u \to v \in R_i$, configuration $C = (\mu, w_1, w_2, \ldots w_m)$)
  $w_i := w_i - u$
  for all $(a, here) \in v$ **do**
    $w_i := w_i + a$
  for all $(a, out) \in v$ **do**
    if $parent(i)$ is empty **then**
      $w_{parent(i)} := w_{parent(i)} + \Phi(a)$
    else
      $w_{parent(i)} := w_{parent(i)} + a$
  for all $(a, in_j) \in v$ **do**
    if j is empty **then**
      $w_j := w_j + \Phi(a)$
    else
      $w_j := w_j + a$
  if $v = v'\delta$ **then**
    $w_{parent(i)} := w_{parent(i)} + w_i$
    $w_i :=$ empty multiset

4.3.3 Vacuum objects

In the common sense, vacuum represents a state of space with no or a little matter in it. Using vacuum in modelling frameworks can help express certain

phenomena more easily. We define a new P system variant, which creates a special vacuum object in a region as soon as the region becomes empty. The vacuum object persists in ther region until it is consumed by a rule. If we made the vacuum to be removed automatically when an object enters the region, there would be no difference with the variant without vacuum objects because of no interactions with it. Instead, the vacuum can be removed explicitly by a rule, e.g. with some interaction with other object. However, it is still created automatically, so we allow the vacuum object to occur only on the left side of rules.

One can object that if there is an interaction of the vacuum object with another object in the same region, the region is nonempty at that time and it does not make sense to have nonempty region with vacuum. But the vacuum object can be regarded as a memory footprint of the past event, when the region was empty. It is just waiting there to be processed.

In the following, the vacuum object is denoted by ν.

We will have rules of type $u \to v$, where u is a string over $\Sigma \cup \{\nu\}$ and $v = v'$ or $v = v'\delta$, where v' is a string over $\Sigma \times (\{here, out\} \cup \{in_j | 1 \leq j \leq m\})$ and δ and ν are special symbols not in Σ.

There will be a phase in the computational step before the rewriting takes place. The vacuum object is created in all empty membranes. It can be seen as a special implicit rule with highest priority.

Theorem 4.3.1. *Sequential P systems with vacuum objects are computationally complete.*

Proof 4.3.1. We will prove the universality by simulating the register machine. If there was an object for every register as in the simulation in the proof 4.1.2, the vacuum object would be created only if all registers are empty. But for the simulation of the *sub* operation we need to detect if one concrete register is empty.

In this simulation the skin membrane has a child membrane for each register. The value of register i will be represented by the multiplicity of object a in membrane i.

The alphabet will contain objects representing the instruction labels of the register machine and the object a, the multiplicity of which represents the contents of a register. Initially, the skin membrane contains only the instruction label. There will be rules to send the object representing the instruction label to the corresponding membrane where the instruction is executed. After the execution the object representing the next instruction label is sent back to the skin membrane.

We will have following rules in the skin membrane:

- $e \to e\downarrow_j$ for an instruction of type $e : (add(j), k, l)$ or $e : (sub(j), k, l)$ and
- $h \to h\downarrow$ for a halting instruction h.

And in every inner membrane j:

- $e \to a|k\uparrow$ for instructions of type $e : (add(j), k, l)$,
- $e \to a|l\uparrow$ for instructions of type $e : (add(j), k, l)$,
- $e|a \to k\uparrow$ for instructions of type $e : (sub(j), k, l)$,
- $e|\nu \to l\uparrow$ for instructions of type $e : (sub(j), k, l)$, and
- $h|a \to h|a$ for a halting instruction h.

When halting, if there is an nonempty register, the last rule $h|a \to h|a$ will always be applicable as there is no rule to consume objects h and a. However, if all registers are empty, h will stay in all membranes and the computation will halt. □

4.3.4 Concluding remarks

The above presented variants show possible implementations of zero-check by the feature of emptyness detection which is a specific notion for membrane systems. The aim of the research was to find a variant with zero-check that will not lead to computational completeness and will possibly reveal some unexpected connection with other models of computation. Sequential P systems with objects altering when entering empty region (subsection 4.3.2) and sequential P systems with vacuum objects (subsection 4.3.3) are universal because we can simulate a register machine. On the other hand, the other variant with objects avoiding empty regions (subsection 4.3.1) appears to be more promising for our goal because the standard contruction of register machine do not work. We conjecture this variant is not universal, possibly equivalent with Petri nets or other model of computation weaker than Turing machine.

4.4 Notions from reaction systems

The variants of P systems inspired by reaction systems introduced in subsection 3.2.9 have been inspected only for maximal parallel semantics. The

various proposed semantics turned out to be deterministic and the research continued to define various semantics for computational step [57] and halting conditions [78]. The sequential mode was only mentioned in [57] under the notion of "min-enabled" computational step. As well as the maximal parallel mode, the sequential set membrane systems can only generate the regular languages [5]. The situation is changed with active membranes. It seems that it is not so black-and-white in terms of computational power of the sequential active P systems that are working with:

- sets instead of multisets,
- the assumption of non-permanency of objects.

In following subsections we study variants of P systems with active membranes that are working in the sequential mode with sets instead of multisets. We challenge the original definition of a membrane creation because possible multiplicity of labels of child membranes are in conflict with no multiplicity of objects in reaction systems. We propose more suitable notions of membrane creation and prove computational completeness by simulating a register machine. In the proofs we use $(op(r), \_, \_)$ to denote any of the operations *add* and *sub* operating on the register r having arbitrary following instruction. These results were published in [61].

We leave the second proposal of assuming non-permanency of objects in sequential mode with active membranes as a topic for the future study.

4.4.1 Alternative definition of active P systems

The rules in active P systems can modify the membrane structure by dissolving and creating new membranes. The skin membrane is not allowed to be dissolved, but object can be sent out through the skin membrane (e.g. to be observed by the surroundings). That is why we will define the configuration to include the membrane structure as well as in [61].

Let Σ be a set of objects. A **membrane configuration** is a tuple (T, l, c), where:

- T is a rooted tree,
- $l \in \mathbb{N}^{V(T)}$ is a mapping that assigns for each node of T a number (label),
- $c \in (2^{\Sigma})^{V(T)}$ is a mapping that assigns for each node of T a set of objects from Σ, so it represents the contents of the membrane.

The common representation of a membrane structure in this section is by a string, where a membrane is denoted by a pair of matching square brackets with the subscript indicating the label of the membrane, e.g. $[_1[_2ab]_2ac]_1$.

A **sequential active set P system** is a tuple $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, where:

- Σ is a set of objects,
- C_0 is the initial membrane configuration,
- $R_1, R_2, \ldots, R_m$ are finite sets of rewriting rules associated with the labels $1, 2, \ldots, m$ and can be of forms:
  - $u \to w$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$,
  - a dissolving rule $u \to w\delta$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$,
  - a membrane creation $u \to [_j v_1]_j v_2$, where $u \subseteq \Sigma$, $|u| \geq 1$, $v_1, v_2 \subseteq \Sigma$ and $1 \leq j \leq m$.

For the first two forms, each rewriting rule may specify for each object on the right side, whether it stays in the current region (we will omit the symbol $\cdot$), moves through the membrane to the parent region ($\uparrow$) or to a specific child region ($\downarrow_j$, where j is a label of a membrane). We denote these transfers with an arrow immediately after the symbol. An example of such rule is the following: $ab \to ab \downarrow_2 c \uparrow c\delta$.

By applying the rule we mean the removal of objects specified on the left side and the addition of the objects on the right side with respect to set union semantics. Symbol $\delta \notin \Sigma$ does not represent an object. It may be present only at the end of the rule, which means that after the application of the rule, the membrane is dissolved and its contents (objects, child membranes) are propagated to the parent membrane. The dissolution of the skin membrane is not allowed.

Active P systems differ from classic (passive) P systems in their ability to create new membranes by rules of the third form. Such rule will create new child membrane with a given label j and a given set of objects v_1 as its contents, while the set v_2 is the set of products that stays in the current membrane. If the current membrane already contains a child membrane with label j, then such rule is not applicable.

For a sequential active set P system $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, configuration $C = (T, l, c)$, membrane $d \in V(T)$ the rule $r \in R_{l(d)}$ is **applicable** iff:

- $r = u \to w$ and $u \subseteq c(d)$ and for all $(a, \downarrow_k) \in w$ there exists $d_2 \in V(T)$ such that $l(d_2) = k \wedge parent(d_2) = d$,

- $r = u \to w\delta$ and $u \subseteq c(d)$ and for all $(a, \downarrow_k) \in w$ there exists $d_2 \in V(T)$ such that $l(d_2) = k \wedge parent(d_2) = d$ and $d \neq r_T$,
- $r = u \to [_j v_1]_j v_2$, $u \subseteq c(d)$ and d has no child membrane with label j $(\forall d_2 \in V(T) : parent(d_2) = d \Rightarrow l(d_2) \neq j)$.

In this section we assume only sequential systems, so in each step of the computation, there is one rule nondeterministically chosen among all applicable rules in all membranes to be applied.

A **computation step** of a sequential active P system is a relation $\Rightarrow$ on the set of membrane configurations such that $C_1 \Rightarrow C_2$ holds iff there is an applicable rule in a membrane in C_1, such that applying that rule results in C_2.

The P system can work in generating or in accepting mode. For the generating mode we consider the concatenation of the objects which leave the system, in the order they are sent out of the skin membrane (if several symbols are expelled at the same time, then any ordering of them is considered). In this case we generate a language. The result of a single computation is clearly only one multiset or a string, but for one initial configuration there can be multiple possible computations. It follows from the fact that there can be more than one applicable rule in each configuration and they are chosen nondeterministically.

For the accepting mode the input word is encoded into a membrane structure by a given encoding and it is accepted if and only if a given accepting configuration can be reached[50].

4.4.2 Simulation of register machine

In this section we will show that sequential active set P systems are powerful computing devices as they can simulate the register machine. We start the formulation of the main theorem followed by a proof made by a simulation. At last the efficiency of the simulation is questioned and various improvements are proposed.

Simple simulation

The main theorem is stated as follows:

Theorem 4.4.1. *Sequential active set P systems are computationally complete.*

Proof 4.4.1. Computational completeness is proved by a direct simulation of a register machine, which is also computationally complete.

For a register machine (m, P, i, h, Lab) we will construct a sequential active set P system $(\Sigma, C_0, R_1, \ldots R_{m+1})$, where

$$\Sigma = \{x_j, y_j \text{ for } j \in Lab\} \cup \{t_i \text{ for each register } \ i\}$$

. Skin membrane will be labeled with $m+1$ and labels 1 to m are reserved for membranes representing registers 1 to m. C_0 will be the input word for the register machine encoded into a membrane structure by the following encoding:

For a configuration of register machine $(r_1, r_2, \ldots r_m, ip)$ the membrane structure will consist of a skin membrane, which will contain m chains consisting of r_i membranes embedded one into another like in a Matryoshka doll with label i. The innermost membranes will contain a single object t_i. If $r_i = 0$ then t_i is in the skin membrane and there is no membrane with label i. Object representing the label of the current instruction (x_{ip}) is in the skin membrane.

Example 4.4.1. For a configuration of register machine $(2, 0, 1)$ the membrane structure is $[_3[_1[_1t_1]_1]_1t_2x_1]_3$. The register 1 has value 2, so there are 2 nested membranes with label 1 and the innermost membrane contains an object t_1. The register 2 has value 0 so there is object t_2 is in the skin membrane with no membrane labeled with 2. In the skin membrane the object x_1 represents the current instruction label 1.

We will have following rules in the skin membrane:

1. $y_j \to x_j$,
2. $x_j \to x_j \downarrow_i$ for instruction $j : (op(i), \_, \_)$,
3. $x_j, t_i \to [_i y_k, t_i]_i$ for instruction $j : (add(i), k, k)$,
4. $x_j, t_i \to x_l, t_i$ for instruction $j : (sub(i), \_, l)$

For the membrane i:

5. $x_j \to x_j \downarrow_i$ for instruction $j : (op(i), \_, \_)$,
6. $x_j, t_i \to [_1 y_k, t_i]_1$ for instruction $j : (add(i), k, k)$,
7. $y_j \to y_j \uparrow$ for instruction $j : (op(i), \_, \_)$,
8. $x_j, t_i \to y_k, t_i, \delta$ for instruction $j : (sub(i), k, l)$

Object x_j represents the instruction currently executed. It is sent down the chain of membranes by rules 2 and 5. In the innermost membrane the creation of a new membrane (rule 6), or the dissolution (rule 8) is performed. Then the next instruction represented by object y_j is sent upwards all the way to the skin membrane by the rule 7. The object t_i is always present in the innermost membrane. For a SUB instruction there are two rules in the skin membrane, together they implement the zero-test. The rule 2 is applicable only if the register is nonempty and the rule 4 is applicable only if the register is empty by requiring the presence of t_i, meaning that the value of register i is zero.

Example 4.4.2. Assume a register machine with two registers with values $r_1 = 2$, $r_2 = 0$ and instructions:

- $1 : sub(1), 2, 3$
- $2 : add(2), 1, 1$
- $3 : halt$

The initial configuration is $[_3[_1[_1t_1]_1]_1t_2x_1]_3$. The computation of the P system is deterministic, at each step there is only one applicable rule. Starting with the rule 2 and then the rule 5, x_1 enters the innermost membrane, where the rule 8 dissolves the membrane. The full example computation:

1. $[_3[_1[_1t_1]_1]_1t_2x_1]_3$
2. $[_3[_1[_1t_1]_1x_1]_1t_2]_3$ (used rule 2)
3. $[_3[_1[_1t_1x_1]_1]_1t_2]_3$ (used rule 5)
4. $[_3[_1t_1y_2]_1t_2]_3$ (used rule 8)
5. $[_3[_1t_1]_1t_2y_2]_3$ (used rule 7)
6. $[_3[_1t_1]_1t_2x_2]_3$ (used rule 1)
7. $[_3[_1t_1]_1[_2t_2y_1]_2]_3$ (used rule 3)
8. $[_3[_1t_1]_1[_2t_2]_2y_1]_3$ (used rule 7)
9. $[_3[_1t_1]_1[_2t_2]_2x_1]_3$ (used rule 1)
10. $[_3[_1t_1x_1]_1[_2t_2]_2]_3$ (used rule 2)
11. $[_3[_2t_2]_2t_1y_2]_3$ (used rule 8)

12. $[_3[_2t_2]_2t_1x_2]_3$ (used rule 1)
13. $[_3[_2t_2x_2]_2t_1]_3$ (used rule 2)
14. $[_3[_2[_2t_2y_1]_2]_2t_1]_3$ (used rule 6)
15. $[_3[_2[_2t_2]_2y_1]_2t_1]_3$ (used rule 7)
16. $[_3[_2[_2t_2]_2]_2t_1y_1]_3$ (used rule 7)
17. $[_3[_2[_2t_2]_2]_2t_1x_1]_3$ (used rule 1)
18. $[_3[_2[_2t_2]_2]_2t_1x_3]_3$ (used rule 4)

We have shown an example computation of P system which simulates the register machine which moves contents of register 1 to the register 2.

The simulation of the register machine was quite straightforward. We proved that the model is computationally complete. However, the simulation is not very effective. It uses alphabet of size $2 *$ number of instructions + number of registers, and its number of membranes is linearly dependent on the sum of values of registers. The time needed for executing an instruction on register i is linearly dependent on r_i.

Optimalization of the simulation

In this subsection we address the inefficient usage of membranes in the previous simulation. New, optimized simulation will reduce it to logarithmic dependency.

For a register machine (m, P, i, h, Lab) we will construct a sequential active set P system, where $\Sigma = \{0, 1, p, q, s, t\} \cup \{x_j, y_j, z_j \text{ for } j \in Lab\}$. Skin membrane will be labeled with $m + 1$ and labels 1 to m are reserved for membranes representing registers 1 to m.

Assume configuration of register machine $(r_1, r_2, \ldots r_m, ip)$. For register i, let $b_1 b_2 \ldots b_k$ be a binary representation of r_i. The skin membrane will contain a chain of k membranes embedded one into another like in a Matryoshka doll with label i. The membrane in depth d will contain the object b_{k-d}, which is either 0 or 1. So the highest-order position in the binary number is represented by the innermost membrane and positions which are more often changed by increments are in membranes closer to the skin membrane. Moreover, the innermost membranes contain a single object t. The skin membrane contains the label of the current instruction x_{ip}. Other membranes (not skin and not innermost) contain s. Direct children of the skin membrane

will contain a single object q representing the fact that we don't want the membrane to be dissolved. Object p will be in all membranes except the skin membrane and direct children of the skin membrane. It represents the fact that the membrane can be dissolved, while keeping at least one membrane for binary representation of the register value.

The basic idea is to recursively decide the next action based on lowest position. For incrementing number ending with zero, incrementing the lowest position is enough. Similar simple case is when decrementing number ending with one. For incrementing number ending with one, we decrement the lowest position and recursively call increment on the binary number omitting the lowest position. Similarly, for decrementing number ending with zero, we increment the lowest position and recursively call decrement on the binary number omitting the lowest position. There are some special cases, like incrementing 111 to 1000 or decrementing 1000 to 111. In these cases we should change the number of membranes representing positions. There is also an exception to this - decrementing 1 to 0 does not change the number of bits.

We will have following rules in the skin membrane:

1. $y_j \to x_j$,
2. $x_j \to x_j \downarrow_i$ for instruction $j : op(i), \_, \_$

For the membrane i and instruction j:

3. $y_j \to y_j \uparrow$ (return the next instruction to the skin membrane).

For the membrane i and instruction $j : add(i), k, k$:

4. $x_j 1s \to x_j \downarrow_i 0s$ (we decremented lower position, so we must increment higher position (011 to 100, now at 1 to 0)),
5. $x_j 0 \to y_k \uparrow 1$ (we incremented a position and can return and proceed to the next instruction),
6. $x_j 1t \to [_i 1tp]_i y_k \uparrow 0s$ (incrementing 111 to 1000).

For the membrane i and instruction $j : sub(i), k, l$:

7. $x_j 1s \to y_k \uparrow 0s$ (we found position to decrement, proceed to the next instruction),
8. $x_j 0s \to x_j \downarrow_i 1s$ (1000 is decremented to 0111 and now we encountered a 0),

9. $x_j 1tp \to z_j t\delta$ (decrementing the number of bits),
10. $x_j 1tq \to y_k \uparrow 0tq$ (exception to the rule 9 - decrementing 1 to 0 does not decrement the number of bits),
11. $z_j st \to y_k t$ (after decremented the number of bits, remove s in the new highest-order position),
12. $x_j 0t \to y_l \uparrow 0t$ (trying to decrement a zero)

Example 4.4.3. Assume a register machine with two registers with values $r_1 = 3$, $r_2 = 0$ and the current instruction $j : add(1), k, k$. The corresponding membrane configuration is $[_3[_1[_1 1tp]_1 1sq]_1[_2 0tq]_2 x_j]_3$.

The computation of the P system is deterministic, at each step there is only one applicable rule. Starting with the rule 2, x_j will meet the object 1 in the configuration $[_3[_1[_1 1tp]_1 1x_j sq]_1[_2 0tq]_2]_3$. The only applicable rule then is 4, resulting in $[_3[_1[_1 1tpx_j]_1 0sq]_1[_2 0tq]_2]_3$. x_j now meets objects 1 and t, which means that the only applicable rule is 6, creating a new innermost membrane and resulting in the configuration $[_3[_1[_1[_1 1tp]_1 0p]_1 0y_k sq]_1[_2 0tq]_2]_3$. The object y_k is by the rule 3 propagated to the skin membrane, where it is prepared for the next instruction by the rule 1.

Example 4.4.4. Assume a register machine with two registers with values $r_1 = 2$, $r_2 = 0$ and instructions:

- $1 : sub(1), 2, 3$
- $2 : add(2), 1, 1$
- $3 : halt$

The initial configuration is $[_3[_1[_1 1tp]_1 0sq]_1[_2 0tq]_2 x_1]_3$. The computation of the P system is deterministic, at each step there is only one applicable rule. Starting with the rule 2 and then the rule 8, x_1 enters the innermost membrane, where the rule 9 dissolves the membrane. The full example computation:

1. $[_3[_1[_1 1tp]_1 0sq]_1[_2 0tq]_2 x_1]_3$
2. $[_3[_1[_1 1tp]_1 0sqx_1]_1[_2 0tq]_2]_3$ (used rule 2)
3. $[_3[_1[_1 1tpx_1]_1 1sq]_1[_2 0tq]_2]_3$ (used rule 8)
4. $[_3[_1 1stqz_1]_1[_2 0tq]_2]_3$ (used rule 9)

5. $[_3[_1 1tqy_2]_1[_2 0tq]_2]_3$ (used rule 11)
6. $[_3[_1 1tq]_1[_2 0tq]_2y_2]_3$ (used rule 3)
7. $[_3[_1 1tq]_1[_2 0tq]_2x_2]_3$ (used rule 1)
8. $[_3[_1 1tq]_1[_2 0tqx_2]_2]_3$ (used rule 2)
9. $[_3[_1 1tq]_1[_2 1tq]_2y_1]_3$ (used rule 5)
10. $[_3[_1 1tq]_1[_2 1tq]_2x_1]_3$ (used rule 1)
11. $[_3[_1 1tqx_1]_1[_2 1tq]_2]_3$ (used rule 2)
12. $[_3[_1 0tq]_1[_2 1tq]_2y_2]_3$ (used rule 6)
13. $[_3[_1 0tq]_1[_2 1tq]_2x_2]_3$ (used rule 1)
14. $[_3[_1 0tq]_1[_2 1tqx_2]_2]_3$ (used rule 2)
15. $[_3[_1 0tq]_1[_2[_2 1tp]_2 0sq]_2y_1]_3$ (used rule 6)
16. $[_3[_1 0tq]_1[_2[_2 1tp]_2 0sq]_2x_1]_3$ (used rule 1)
17. $[_3[_1 0tqx_1]_1[_2[_2 1tp]_2 0sq]_2]_3$ (used rule 2)
18. $[_3[_1 0tq]_1[_2[_2 1tp]_2 0sq]_2y_3]_3$ (used rule 12)
19. $[_3[_1 0tq]_1[_2[_2 1tp]_2 0sq]_2x_3]_3$ (used rule 1)

We have shown an example computation of P system which simulates the register machine which moves contents of register 1 to the register 2.

One instruction of the register machine is performed by number of computational steps which is logarithmic on the value of the register the instruction is operated on. The number of membranes is logarithmic as well. The number of objects is $3 *$ number of instructions $+ 6$.

Further optimalizations

Could the simulation be optimized even more? Encoding the register value to a chain of membranes is not making full use of membrane structure. There are many options for a represention of an integer by a tree. For efficient implementation of the increment and decrement instructions, we need an encoding with a property that a local change in the value of the encoding

of the entire tree corresponds to a local change in the value of the encodings of its child subtrees. Stein in 1999 [100] proposed a boustrophedonic ("ox-plowing") variant of Cantor pairing function. The implementation of a sequential active set P system simulating a register machine using this pairing function to encode child subtrees would be quite easy, but we would stick to the logarithmic time in the worst case (diagonal of the pairing function). Catalan pairing function [99] orders full binary trees by the number of nodes. The time would be logarithmic with a base 4, which is a slight improvement, but asymptotically still the same. Searching for a suitable encoding of a tree is a good proposal for the further study.

4.4.3 Modified membrane creation semantics

In this subsection we will investigate the effect of other semantics of membrane creation. The previous semantics assumed an explicit membrane creation rule. If the current membrane already contains child membrane with the same label as the membrane about to be created, then the rule is not applicable, and the membrane creation is aborted. Similar behavior is in the definition of sending objects to the child membrane. If such membrane does not exist, objects cannot be sent and the rule is not applicable.

These two behaviors are in fact complementary. It seems natural to join these two artificial rule abortions and provide a rule that will always be applicable if the precondition of left side inclusion is fulfilled.

Semantics inject-or-create

We will first examine the case where we have no explicit membrane creation rule. Any rule which is sending some objects to child membrane labeled j will create child membrane j if it does not exist.

Formally, rules can be of form:

- $u \to w$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$,
- a dissolving rule $u \to w\delta$, where $u \subseteq \Sigma$, $|u| \geq 1$, $w \subseteq (\Sigma \times \{\cdot, \uparrow, \downarrow_j\})$ and $1 \leq j \leq m$.

For a sequential active set P system $(\Sigma, C_0, R_1, R_2, \ldots, R_m)$, configuration $C = (T, l, c)$, membrane $d \in V(T)$ the rule $r \in R_{l(d)}$ is **applicable** iff:

- $r = u \to w$ and $u \subseteq c(d)$,
- $r = u \to w\delta$ and $u \subseteq c(d)$ and $d \neq r_T$.

Example 4.4.5. Assume the membrane configuration $[_1[_2]_2a]_1$. If we apply the rule $a \to a \downarrow_2$ in the skin membrane, the object a is sent to the membrane 2 because such child membrane already exists. The resulting membrane configuration is $[_1[_2a]_2]_1$.

If we apply the rule $a \to a \downarrow_3$ in the skin membrane, a new membrane labeled 3 is created, because such child membrane does not exist yet. The resulting membrane configuration is $[_1[_2]_2[_3a]_3]_1$.

Theorem 4.4.2. *Sequential active set P systems with inject-or-create semantics are computationally complete.*

Proof 4.4.2. The simulation is essentially the same as in section 4.4.2. All the rules which are sending objects into a child membrane are already assuming that the child membrane already exists. The only difference is in the rule for membrane creation: $x_j 1t \to [_i 1tp]_i y_k \uparrow 0s$. This rule is applied always in the innermost membrane with no child membranes. Modified simulation will therefore use rule $x_j 1t \to 1 \downarrow_i t \downarrow_i p \downarrow_i y_k \uparrow 0s$, which, when applied, creates a child membrane i, because no such child membrane exists. It results in the same configuration as in the simulation with original semantics.

The efficiency of this simulation is essentially the same as in section 4.4.2, that means logarithmic number of steps for simulating one instruction of the register machine as well as logarithmic number of membranes. The number of objects used is one less than in the simulation with original semantics: $3 *$ number of instructions $+ 5$.

Semantics wrap-or-create

In this variant we stay with explicit membrane creation rule, but when the membrane with the same label is already contained in the current membrane, the rule remains applicable and the child membrane will be wrapped by a new membrane with the given contents. For example, applying the rule $a \to [_2b]_2c$ in the membrane 1 of membrane structure $[_1a[_2d]_2]_1$ would result in $[_1c[_2b[_2d]_2]_2]_1$.

Theorem 4.4.3. *Sequential active set P systems with wrap-or-create semantics are computationally complete.*

Proof 4.4.3. Again, we will show how to simulate the register machine. The simulation will be similar to the one defined in subsection 4.4.2, but with additional control objects similar to the second simulation 4.4.2.

For a register machine (m, P, i, h, Lab) we will construct a sequential active set P system $(\Sigma, C_0, R_1, \ldots R_{m+1})$, where

$$\Sigma = \{x_j \text{ for } j \in Lab\} \cup \{t_i, s_i \text{ for each register } \ i\}$$

. Skin membrane will be labeled with $m+1$ and labels 1 to m are reserved for membranes representing registers 1 to m. C_0 will be the input word for the register machine encoded into a membrane structure by the following encoding:

For a configuration of register machine $(r_1, r_2, \ldots r_m, ip)$ the membrane structure will consist of a skin membrane, which will contain m chains consisting of r_i membranes embedded one into another like in a Matryoshka doll with label i. Membranes with a child labeled i will contain a single object s_i. If the membrane has no child labeled i, it contains an object t_i. If $r_i = 0$ then t_i is in the skin membrane and there is no membrane with label i. Object representing the label of the current instruction (x_{ip}) is in the skin membrane.

We will have following rules in the skin membrane:

1. $x_j s_i \rightarrow [_i s_i]_i s_i x_k$ for instruction $j : (add(i), k, \_)$,
2. $x_j t_i \rightarrow [_i t_i]_i s_i x_k$ for instruction $j : (add(i), k, \_)$,
3. $x_j t_i \rightarrow x_l t_i$ for instruction $j : (sub(i), k, l)$,
4. $x_j s_i \rightarrow x_j \downarrow_i$ for instruction $j : (sub(i), k, l)$,

For the membrane i:

5. $x_j \rightarrow x_k \delta$

For every *add* instruction there is just one rule applied in the simulation and for each *sub* instruction there is one or two instructions, depending on the register value. If $r_i > 0$ then the instruction enters the membrane labeled i and dissolves it, decreasing the number of stacked membranes with label i.

Example 4.4.6. Assume a register machine with two registers with values $r_1 = 2$ and $r_2 = 0$ and instructions:

- $1 : sub(1), 2, 3$
- $2 : add(2), 1, 1$
- $3 : halt$

.

The initial membrane configuration is $[_3[_1[_1 t_1]_1 s_1]_1 s_1 t_2 x_1]_3$. The computation of the P system is deterministic, at each step there is only one applicable rule. Starting with the rule 4, x_1 enters the inner membrane, where the rule 5 dissolves the membrane. The full example computation:

1. $[_3[_1[_1t_1]_1s_1]_1s_1t_2x_1]_3$
2. $[_3[_1[_1t_1]_1s_1x_1]_1t_2]_3$ (used rule 4)
3. $[_3[_1t_1]_1s_1t_2x_2]_3$ (used rule 5)
4. $[_3[_1t_1]_1[_2t_2]_2s_1s_2x_1]_3$ (used rule 2)
5. $[_3[_1t_1x_1]_1[_2t_2]_2s_2]_3$ (used rule 4)
6. $[_3[_2t_2]_2t_1s_2x_2]_3$ (used rule 5)
7. $[_3[_2[_2t_2]_2s_2]_2t_1s_2x_1]_3$ (used rule 1)
8. $[_3[_2[_2t_2]_2s_2]_2t_1s_2x_3]_3$ (used rule 3)

We have shown an example computation of P system which simulates the register machine which moves contents of register 1 to the register 2.

This simulation is the most suitable for simulating the register machine because for incrementing the number of stacked membranes we just need to wrap the topmost membrane into a new membrane. This gained us constant time for executing one instruction, however the number of membranes remains linear on the sum of register values. The number of objects is number of instructions $+ 2 *$ number of registers.

Comparison of semantics

We have investigated the bridge between membrane systems and reaction systems for the sequential variant with active membranes. We have shown computational completeness by a simulation of a register machine. When using sets instead of multisets, the original definition of creating a membrane may seem obsolete. Therefore, we have proposed alternative definitions for membrane creation: inject-or-create and wrap-or-create. In either case the resulting system has been shown to be universal.

As some simulations are not very effective, we have also proposed ways to improve the efficiency. For the simulation with original membrane creation we managed to reduce time needed for executing one instruction of the register machine from linear to logarithmic time. The wrap-or-create semantics is the most suitable for the simulation as for every instruction of the register machine only constant number of steps of the P system is needed.

The register value in the simulations is encoded in either unary or binary form. We propose other options to encode the register value as a tree structure of membranes and leave the construction of the simulation as a topic for further study.

semantics	membranes	time	alphabet
original	$O(n)$	$O(n)$	$2i + r$
original	$O(log(n))$	$O(log(n))$	$3i + 6$
inject-or-create	$O(log(n))$	$O(log(n))$	$3i + 6$
wrap-or-create	$O(n)$	$O(1)$	$i + 2r$

Figure 4.6: A comparison of different membrane creation semantics. Alphabet size in the last column depends on number of instructions i and number of registers r.

For the future study we suggest an investigation of decidability properties of these models as well as other inspirations from reaction systems, e. g. non-permanency of objects.

Conclusions

We have studied several variants of sequential P systems in order to obtain universality without using maximal parallelism. A variant with rewriting rules that can use inhibitors was shown to be universal in both generating and accepting case. The generating model is able to simulate maximal parallel P system and the accepting model can simulate a register machine. The constructive proof for the generating case is valuable not only for the universality, but also can be seen as a method of conversion between P systems in sequential manner and maximally parallel manner, which may be essential for future works on P systems and other multiset rewriting systems. The simulation also shows a method how to synchronize application of multiple rules in a membrane and how to synchronize this parallel rule application across whole membrane structure. Sequential variants are promising alternative to traditional maximal parallel variants and will be good subject for the further research. Future plans include research of other more restricted variants such as omitting cooperation in the rules or restricting the power of inhibitors.

In addition, we have defined a new variants of zero-testing, aiming to fit in layers between mere reformulations of the basic sequential P system and universal sequential P systems with inhibitors and possibly to reveal some unexpected connection with other models of computation. We studies variants with various forms of detection of empty membranes - a notion specific for membrane systems. The results obtained have been just the computational completeness. However, one variant with objects avoiding empty regions is more promising for our goal because the standard contruction of register machine do not work. We conjecture this variant is not universal, possibly equivalent with Petri nets or other model of computation weaker than Turing machine.

There are many features not yet combined, so we suggest them for the further research (non-cooperative rules, rules with priorities, decaying objects, deterministic steps, ...).

Aside from the research of the computational power, there are many open

problems in the area of decision problems of certain properties. Interesting ideas for future work can be taken from [14]. They define an abstract notion of negative application conditions for general rewriting systems, which is for multiset rewriting rendered as the usage of inhibitors. Although they considered only nondeleting rules (after application of each rule the resulting multiset is a superset of the current multiset), interesting results were shown that the termination of rewriting was shown to be decidable.

We have investigated the decidability problems of existence of (in)finite computation for a universal class of P systems with active membranes. We have shown and published our results that are on both sides of the decidability barrier. Regarding the open problem stated in [50] about sequential active P systems with hard membranes (without communication between membranes), it could be interesting to find a connection between the universality and decidability of these termination problems.

We research sequential P systems with active membranes also in combination with notions inspired by reaction systems. Variants using sets instead of multisets are shown to be computational complete. We have provided a proof by a simulation of a register machine. We have proposed alternative definitions for membrane creation: inject-or-create and wrap-or-create. In either case the resulting system has been shown to be universal. We suggest investigating of decidability properties of these models as well as other inspirations from reaction systems, e. g. non-permanency of objects. There are no results yet in this area and our proposals could be set as a single topic for the future study.

Bibliography

[1] (2008): *LifeWiki*. Available at http://web.archive.org/web/20161229101754/http://www.conwaylife.com/wiki/Main_Page.

[2] (2010): *NP-Completeness*. In Claude Sammut & GeoffreyI. Webb, editors: *Encyclopedia of Machine Learning*, Springer US, pp. 731–732, doi:10.1007/978-0-387-30164-8_603. Available at http://dx.doi.org/10.1007/978-0-387-30164-8_603.

[3] LM Adleman (1994): *Molecular computation of solutions to combinatorial problems*. *Science* 266(5187), pp. 1021–1024, doi:10.1126/science.7973651. Available at http://www.sciencemag.org/content/266/5187/1021.abstract.

[4] Oana Agrigoroaiei & Gabriel Ciobanu (2010): *Flattening the transition P systems with dissolution*. In: *Proceedings of the 11th international conference on Membrane computing*, CMC'10, Springer-Verlag, Berlin, Heidelberg, pp. 53–64. Available at http://dl.acm.org/citation.cfm?id=1946067.1946077.

[5] Artiom Alhazov (2005): *P systems without multiplicities of symbol-objects*. In: *Information Processing Letters, accepted*.

[6] Artiom Alhazov & Rudolf Freund (2014): *P Systems with Toxic Objects*. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa, Petr Sosík & Claudio Zandron, editors: *Membrane Computing*, *Lecture Notes in Computer Science* 8961, Springer International Publishing, pp. 99–125, doi:10.1007/978-3-319-14370-5_7. Available at http://dx.doi.org/10.1007/978-3-319-14370-5_7.

[7] IoanI. Ardelean (2006): *Biological Roots and Applications of P Systems: Further Suggestions*. In HendrikJan Hoogeboom, Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa, editors: *Membrane*

Computing, *Lecture Notes in Computer Science* 4361, Springer Berlin Heidelberg, pp. 1–17, doi:10.1007/11963516_1. Available at `http://dx.doi.org/10.1007/11963516_1`.

[8] FAC Azevedo, LRB Carvalho, LT Grinberg, JM Farfel, REL Ferreti, REP Leite, WJ Filho, R Lent & S Herculano-Houzel (2009): *Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain*. *Journal of Comparative Neurology* 513(5), pp. 532–541, doi:10.1002/cne.21974.

[9] Henry Givens Baker (1973): *Rabin's proof of the undecidability of the reachability set inclusion problem of vector addition systems*. Massachusetts Institute of Technology, Project MAC.

[10] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo & Simone Tini (2010): *Membrane systems working in generating and accepting modes: expressiveness and encodings*. In: *Proceedings of the 11th international conference on Membrane computing*, CMC'10, Springer-Verlag, Berlin, Heidelberg, pp. 103–118. Available at `http://dl.acm.org/citation.cfm?id=1946067.1946081`.

[11] Roberto Barbuti, Andrea Maggiolo–Schettini, Paolo Milazzo & Angelo Troina (2007): *The Calculus of Looping Sequences for Modeling Biological Membranes*. In George Eleftherakis, Petros Kefalas, Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa, editors: *Membrane Computing*, *Lecture Notes in Computer Science* 4860, Springer Berlin Heidelberg, pp. 54–76, doi:10.1007/978-3-540-77312-2_4. Available at `http://dx.doi.org/10.1007/978-3-540-77312-2_4`.

[12] Daniela Besozzi (2004): *Computational and modelling power of P systems*. Ph.D. thesis, Universita' degli Studi di Milano, Milano, Italy.

[13] Henning Bordihn, Henning Fernau & Markus Holzer (1999): *Accepting Pure Grammars and Systems*. In: *Otto-von-Guericke-Universitat Magdeburg, Fakultat fur Informatik, Preprint Nr.*

[14] Paolo Bottoni, Kathrin Hoffmann & Francesco Parisi-Presicce (2006): *Termination of Algebraic Rewriting with Inhibitors*. *ECEASST* 4. Available at `http://dblp.uni-trier.de/db/journals/eceasst/eceasst4.html#BottoniHP06`.

[15] Catalin Buiu, Cristian Vasile & Octavian Arsene (2012): *Development of membrane controllers for mobile robots. Information Sciences* 187(0), pp. 33 – 51, doi:10.1016/j.ins.2011.10.007. Available at `http://www.sciencedirect.com/science/article/pii/S0020025511005421`.

[16] Mónica Cardona, M. Angels Colomer, Antoni Margalida, Antoni Palau, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez & Delfí Sanuy (2011): *A Computational Modeling for Real Ecosystems Based on P Systems* 10(1), pp. 39–53. doi:10.1007/s11047-010-9191-3. Available at `http://dx.doi.org/10.1007/s11047-010-9191-3`.

[17] Mónica Cardona, M. Angels Colomer, Mario J. Pérez-Jiménez, Delfí Sanuy & Antoni Margalida (2009): *Membrane Computing.* chapter Modeling Ecosystems Using P Systems: The Bearded Vulture, a Case Study, Springer-Verlag, Berlin, Heidelberg, pp. 137–156, doi:10.1007/978-3-540-95885-7_11. Available at `http://dx.doi.org/10.1007/978-3-540-95885-7_11`.

[18] Professor Cayley (1881): *On the Analytical Forms Called Trees. American Journal of Mathematics* 4(1), pp. pp. 266–268. Available at `http://www.jstor.org/stable/2369158`.

[19] José M. Cecilia, José M. García, Ginés D. Guerrero, Miguel A. Martínez del Amor, Ignacio Pérez-Hurtado & Mario J. Pérez-Jiménez (2010): *Simulating a P system based efficient solution to SAT by using GPUs. Journal of Logic and Algebraic Programming* 79, pp. 317–325, doi:dx.doi.org/10.1016/j.jlap.2010.03.008. Available at `http://dx.doi.org/10.1016/j.jlap.2010.03.008`. Membrane computing and programming.

[20] N. Chomsky (1956): *Three models for the description of language. Information Theory, IRE Transactions on* 2(3), pp. 113–124, doi:10.1109/TIT.1956.1056813.

[21] George M. Church, Yuan Gao & Sriram Kosuri (2012): *Next-Generation Digital Information Storage in DNA. Science* 337(6102), p. 1628, doi:10.1126/science.1226355. Available at `http://www.sciencemag.org/content/337/6102/1628.abstract`.

[22] G. Ciobanu & Gheorghe Păun (2005): *The minimal parallelism is still universal.*

[23] Gabriel Ciobanu, Linqiang Pan, Gheorghe Pun & Mario J. Pérez-Jiménez (2007): *P systems with minimal parallelism. Theor. Comput. Sci.* 378(1), pp. 117–130, doi:10.1016/j.tcs.2007.03.044. Available at `http://dx.doi.org/10.1016/j.tcs.2007.03.044`.

[24] Matthew Cook (2004): *Universality in Elementary Cellular Automata. Complex Systems* 15(1), pp. 1–40.

[25] A. Cremers & O. Mayer (1973): *On matrix languages. Information and Control* 23(1), pp. 86 – 96, doi:http://dx.doi.org/10.1016/S0019-9958(73)90917-0. Available at `http://www.sciencedirect.com/science/article/pii/S0019995873909170`.

[26] B. Crowther (2003): *Flocking of autonomous unmanned air vehicles. The Aeronautical Journal (1968)* 107(1068), p. 99–109, doi:10.1017/S0001924000018388.

[27] Zhe Dang & Oscar H. Ibarra (2004): *On P systems operating in sequential mode.* In: *International Journal of Foundations of Computer Science*, pp. 164–177.

[28] Martin Davis (1973): *Hilbert's tenth problem is unsolvable. American Mathematical Monthly*, pp. 233–269.

[29] Rocco De Nicola & Frits Vaandrager (1995): *Three Logics for Branching Bisimulation. J. ACM* 42(2), pp. 458–487, doi:10.1145/201019.201032. Available at `http://doi.acm.org/10.1145/201019.201032`.

[30] Reinhard Diestel (1997): *Graph Theory. Graduate Texts in Mathematics* 173, Springer.

[31] M. Dorigo, V. Maniezzo & A. Colorni (1996): *Ant System: Optimization by a Colony of Cooperating Agents. Trans. Sys. Man Cyber. Part B* 26(1), pp. 29–41, doi:10.1109/3477.484436. Available at `http://dx.doi.org/10.1109/3477.484436`.

[32] Marco Dorigo & Thomas Stützle (2004): *Ant Colony Optimization.* Bradford Company, Scituate, MA, USA.

[33] Catherine Dufourd, Alain Finkel & Ph. Schnoebelen (1998): *Reset Nets Between Decidability and Undecidability.* In: *Proceedings of the 25th International Colloquium on Automata, Languages and*

Programming, ICALP '98, Springer-Verlag, London, UK, UK, pp. 103–115. Available at `http://dl.acm.org/citation.cfm?id=646252.686157`.

[34] A. Ehrenfeucht & G. Rozenberg (2009): *Introducing Time in Reaction Systems. Theor. Comput. Sci.* 410(4-5), pp. 310–322, doi:10.1016/j.tcs.2008.09.043. Available at `http://dx.doi.org/10.1016/j.tcs.2008.09.043`.

[35] Diego Figueira, Santiago Figueira, Sylvain Schmitz & Philippe Schnoebelen (2011): *Ackermannian and Primitive-Recursive Bounds with Dickson's Lemma*. In: *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*, LICS '11, IEEE Computer Society, Washington, DC, USA, pp. 269–278, doi:10.1109/LICS.2011.39. Available at `http://dx.doi.org/10.1109/LICS.2011.39`.

[36] Alain Finkel (1993): *The minimal coverability graph for Petri nets*. In Grzegorz Rozenberg, editor: *Advances in Petri Nets 1993*, *Lecture Notes in Computer Science* 674, Springer Berlin Heidelberg, pp. 210–243, doi:10.1007/3-540-56689-9_45. Available at `http://dx.doi.org/10.1007/3-540-56689-9_45`.

[37] Rudolf Freund (2005): *Asynchronous P systems and P systems working in the sequential mode*. In: *Proceedings of the 5th international conference on Membrane Computing*, WMC'04, Springer-Verlag, Berlin, Heidelberg, pp. 36–62, doi:10.1007/978-3-540-31837-8_3. Available at `http://dx.doi.org/10.1007/978-3-540-31837-8_3`.

[38] Rudolf Freund, Lila Kari, Marion Oswald & Petr Sosík (2005): *Computationally universal P systems without priorities: two catalysts are sufficient. Theoretical Computer Science* 330(2), pp. 251 – 266, doi:10.1016/j.tcs.2004.06.029. Available at `http://www.sciencedirect.com/science/article/pii/S0304397504006553`. Descriptional Complexity of Formal Systems.

[39] Rudolf Freund, Alberto Leporati, Marion Oswald & Claudio Zandron (2005): *Sequential P systems with unit rules and energy assigned to membranes*. In: *Proceedings of the 4th international conference on Machines, Computations, and Universality*, MCU'04, Springer-Verlag, Berlin, Heidelberg, pp. 200–210, doi:10.1007/978-3-540-31834-7_16. Available at `http://dx.doi.org/10.1007/978-3-540-31834-7_16`.

[40] M. Gardner (1970): *Mathematical Games: the Fantastic Combinations of John Conway's New Solitaire Game 'Life'. Scientific American* 223(4), pp. 120–123.

[41] Renana Gershoni, Ehud Keinan, Gheorghe Păun, Ron Piran, Tamar Ratner & Sivan Shoshani (2008): *Research topics arising from the (planned) P systems implementation experiment in Technion.* In Daniel Díaz-Pernil, Carmen Graciani, Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Ignacio Pérez-Hurtado & Agustín Riscos-Nú nez, editors: *Sixth Brainstorming Week on Membrane Computing*, pp. 183–192.

[42] Alex Graves, Greg Wayne & Ivo Danihelka (2014): *Neural Turing Machines. CoRR* abs/1410.5401. Available at `http://arxiv.org/abs/1410.5401`.

[43] M. Hack (1976): *Decidability Questions for Petri Nets.* Technical Report, Cambridge, MA, USA.

[44] Michel Hack (1974): *The Recursive Equivalence of the Reachability Problem and the Liveness Problem for Petri Nets and Vector Addition Systems.* In: *Proceedings of the 15th Annual Symposium on Switching and Automata Theory (Swat 1974)*, SWAT '74, IEEE Computer Society, Washington, DC, USA, pp. 156–164, doi:10.1109/SWAT.1974.28. Available at `http://dx.doi.org/10.1109/SWAT.1974.28`.

[45] Michel Hack (1974): *The Recursive Equivalence of the Reachability Problem and the Liveness Problem for Petri Nets and Vector Addition Systems.* In: *Proceedings of the 15th Annual Symposium on Switching and Automata Theory (Swat 1974)*, SWAT '74, IEEE Computer Society, Washington, DC, USA, pp. 156–164, doi:10.1109/SWAT.1974.28. Available at `http://dx.doi.org/10.1109/SWAT.1974.28`.

[46] Michel Hack (1976): *The equality problem for vector addition systems is undecidable. Theoretical Computer Science* 2(1), pp. 77 – 95, doi:http://dx.doi.org/10.1016/0304-3975(76)90008-6. Available at `http://www.sciencedirect.com/science/article/pii/0304397576900086`.

[47] Oscar H. Ibarra, Zhe Dang & Omer Egecioglu (2004): *Catalytic P Systems, Semilinear Sets, and Vector Addition Systems. Theor.*

Comput. Sci. 312(2-3), pp. 379–399, doi:10.1016/j.tcs.2003.10.028. Available at `http://dx.doi.org/10.1016/j.tcs.2003.10.028`.

[48] Oscar H. Ibarra, Hsu chun Yen & Zhe Dang (2004): *Dang: The Power of Maximal Parallelism in P Systems*. In: *Proceedings of the Eight Conference on Developments in Language Theory*, Springer, pp. 212–224.

[49] OscarH. Ibarra, Zhe Dang, Omer Egecioglu & Gaurav Saxena (2003): *Characterizations of Catalytic Membrane Computing Systems*. In Branislav Rovan & Peter Vojtáš, editors: *Mathematical Foundations of Computer Science 2003*, *Lecture Notes in Computer Science* 2747, Springer Berlin Heidelberg, pp. 480–489, doi:10.1007/978-3-540-45138-9_42. Available at `http://dx.doi.org/10.1007/978-3-540-45138-9_42`.

[50] OscarH. Ibarra, Sara Woodworth, Hsu-Chun Yen & Zhe Dang (2005): *On Sequential and 1-Deterministic P Systems*. In Lusheng Wang, editor: *Proceedings of the 11th annual international conference on Computing and Combinatorics*, *COCOON'05* 3595, Springer-Verlag, Berlin, Heidelberg, pp. 905–914, doi:10.1007/11533719_91. Available at `http://dx.doi.org/10.1007/11533719_91`.

[51] Mihai Ionescu & Dragos Sburlan (2004): *On P Systems with Promoters/Inhibitors*. *Journal of Universal Computer Science* 10(5), pp. 581–599. Available at `http://www.jucs.org/jucs_10_5/on_p_systems_with`.

[52] Ryuichi Ito (1969): *Every semilinear set is a finite union of disjoint linear sets*. *Journal of Computer and System Sciences* 3(2), pp. 221 – 231, doi:http://dx.doi.org/10.1016/S0022-0000(69)80014-0. Available at `http://www.sciencedirect.com/science/article/pii/S0022000069800140`.

[53] Dervis Karaboga & Bahriye Basturk (2007): *A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm*. *Journal of Global Optimization* 39(3), pp. 459–471, doi:10.1007/s10898-007-9149-x. Available at `http://dx.doi.org/10.1007/s10898-007-9149-x`.

[54] Lila Kari, Greg Gloor & Sheng Yu (2000): *Using DNA to solve the Bounded Post Correspondence Problem*. *Theoretical Computer Science* 231(2), pp. 193 – 203, doi:10.1016/S0304-3975(99)00100-0.

Available at http://www.sciencedirect.com/science/article/pii/S0304397599001000.

[55] Richard M. Karp & Raymond E. Miller (1969): *Parallel Program Schemata. J. Comput. Syst. Sci.* 3(2), pp. 147–195, doi:10.1016/S0022-0000(69)80011-5. Available at http://dx.doi.org/10.1016/S0022-0000(69)80011-5.

[56] S. Khrisna & A. Păun (2003): *Three universality results on P systems. TR 28/03, URV Tarragona.*

[57] Jetty Kleijn & Maciej Koutny (2011): *Membrane Systems with Qualitative Evolution Rules. Fundam. Inf.* 110(1-4), pp. 217–230. Available at http://dl.acm.org/citation.cfm?id=2362097.2362112.

[58] Peter Korosec & Jurij Silc (2009): *Applications of the Differential Ant-Stigmergy Algorithm on Real-World Continuous Optimization Problems.* In Wellington Pinheiro dos Santos, editor: *Evolutionary Computation*, chapter 11, InTech, Rijeka, doi:10.5772/9604. Available at http://dx.doi.org/10.5772/9604.

[59] Michal Kováč (2014): *Inhibiting the Parallelism in P Systems.* In: *Informal electronic proceedings of CiE 2014.*

[60] Michal Kováč (2015): *Decidability of Termination Problems for Sequential P Systems with Active Membranes*, pp. 236–245. 9136, Springer International Publishing, Cham, doi:10.1007/978-3-319-20028-6_24. Available at http://dx.doi.org/10.1007/978-3-319-20028-6_24.

[61] Michal Kováč & Damas P. Gruska (2015): *Sequential P Systems with Active Membranes Working on Sets.* In: *CS&P*, 1492.

[62] Dexter C. Kozen (1997): *Automata and Computability*, 1st edition. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[63] Yanzhang Li, Changjun Zhou & Xuedong Zheng (2014): *The Application of Artificial Bee Colony Algorithm in Protein Structure Prediction*, pp. 255–258. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-662-45049-9_42. Available at http://dx.doi.org/10.1007/978-3-662-45049-9_42.

[64] Aristid Lindenmayer (1968): *Mathematical Models for Cellular Interactions in Development, II. Simple and Branching Filaments with Two-sided Inputs*. *J. Theoretical Biology*, pp. 280–315.

[65] Richard J. Lipton (1976): *The reachability problem requires exponential space*. Research report (Yale University. Department of Computer Science), Department of Computer Science, Yale University. Available at `http://books.google.sk/books?id=7iSbGwAACAAJ`.

[66] Stelios Manousakis (2006): *Musical L-Systems*. Master's thesis, The Royal Conservatory, The Hague.

[67] Ernst W. Mayr (1981): *An Algorithm for the General Petri Net Reachability Problem*. In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, ACM, New York, NY, USA, pp. 238–246, doi:10.1145/800076.802477. Available at `http://doi.acm.org/10.1145/800076.802477`.

[68] Paolo Milazzo (2007): *Qualitative and Quantitative Formal Modeling of Biological Systems*. Ph.D. thesis, Universita di Pisa.

[69] Marvin L. Minsky (1967): *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[70] A. V. Moere (2004): *Time-Varying Data Visualization Using Information Flocking Boids*. In: *IEEE Symposium on Information Visualization*, pp. 97–104, doi:10.1109/INFVIS.2004.65.

[71] David J. Montana & Lawrence Davis (1989): *Training feedforward neural networks using genetic algorithms*. In: *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1*, IJCAI'89, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 762–767. Available at `http://dl.acm.org/citation.cfm?id=1623755.1623876`.

[72] Research Group on Natural Computing (2000): *MeCoSim Membrane Computing Simulator*. Available at `http://www.p-lingua.org/mecosim/`.

[73] John Von Neumann (1966): *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA.

[74] Gabriela Ochoa (1998): *On genetic algorithms and lindenmayer systems*, pp. 335–344. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/BFb0056876. Available at `http://dx.doi.org/10.1007/BFb0056876`.

[75] Rohit J. Parikh (1966): *On Context-Free Languages. J. ACM* 13(4), pp. 570–581, doi:10.1145/321356.321364. Available at `http://doi.acm.org/10.1145/321356.321364`.

[76] Andrei Paun & Gheorghe Paun (2002): *The power of communication: P systems with symport/antiport. New Gen. Comput.* 20(3), pp. 295–305, doi:10.1007/BF03037362. Available at `http://dx.doi.org/10.1007/BF03037362`.

[77] Gheorghe Păun (1998): *Computing with Membranes*. Technical Report 208, Turku Center for Computer Science-TUCS. (www.tucs.fi).

[78] Gheorghe Paun & Mario J. Pérez-Jiménez (2012): *Towards bridging two cell-inspired models: P systems and R systems. Theoretical Computer Science* 429, pp. 258–264, doi:http://dx.doi.org/10.1016/j.tcs.2011.12.046. Available at `http://www.sciencedirect.com/science/article/pii/S0304397511010127`.

[79] Gheorghe Paun, Grzegorz Rozenberg & Arto Salomaa (2010): *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA.

[80] Mario J. Pérez-Jiménez (2006): *P systems-based modelling of cellular signalling pathways. Workshop on Membrane Computing, WMC7*, pp. 54–73.

[81] James Lyle Peterson (1981): *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[82] Carl Adam Petri (1962): *Kommunikation mit Automaten*. Ph.D. thesis, Universität Hamburg.

[83] Ignacio Pérez-Hurtado, Luis Valencia-Cabrera, Mario J. Pérez-Jiménez, Maria Angels Colomer & Agustin Riscos-Núñez (2010): *MeCoSim: A general purpose software tool for simulating biological phenomena by means of P systems*. In: *BIC-TA'10*, pp. 637–643.

[84] Gheorghe Păun (1999): *P Systems with Active Membranes: Attacking NP Complete Problems*. *Journal of Automata, Languages and Combinatorics* 6, pp. 75–90.

[85] Gheorghe Păun, Yasuhiro Suzuki & Hiroshi Tanaka (2001): *P systems with energy accounting*. *International Journal of Computer Mathematics* 78(3), pp. 343–364, doi:10.1080/00207160108805116. Available at `http://www.tandfonline.com/doi/abs/10.1080/00207160108805116`.

[86] Charles Rackoff (1978): *The covering and boundedness problems for vector addition systems*. *Theoretical Computer Science* 6(2), pp. 223 – 231, doi:http://dx.doi.org/10.1016/0304-3975(78)90036-1. Available at `http://www.sciencedirect.com/science/article/pii/0304397578900361`.

[87] Paul Rendell (2015): *Turing Machine Universality of the Game of Life*, 1st edition. Springer Publishing Company, Incorporated.

[88] Craig W. Reynolds (1987): *Flocks, Herds and Schools: A Distributed Behavioral Model*. *SIGGRAPH Comput. Graph.* 21(4), pp. 25–34, doi:10.1145/37402.37406. Available at `http://doi.acm.org/10.1145/37402.37406`.

[89] H. G. Rice (1953): *Classes of Recursively Enumerable Sets and Their Decision Problems*. *Trans. Amer. Math. Soc.* 74, pp. 358–366.

[90] A. E. Rizzoli, R. Montemanni, E. Lucibello & L. M. Gambardella (2007): *Ant colony optimization for real-world vehicle routing problems*. *Swarm Intelligence* 1(2), pp. 135–151, doi:10.1007/s11721-007-0005-x. Available at `http://dx.doi.org/10.1007/s11721-007-0005-x`.

[91] G. Rozenberg & A. Salomaa (2012): *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*. Springer Berlin Heidelberg. Available at `https://books.google.sk/books?id=BbqAS5IUWAUC`.

[92] Grzegorz Rozenberg (2009): *Reaction Systems: A Formal Framework for Processes*. In Giuliana Franceschinis & Karsten Wolf, editors: *Applications and Theory of Petri Nets*, *Lecture Notes in Computer Science* 5606, Springer Berlin Heidelberg, pp. 22–22, doi:10.1007/978-3-642-02424-5_3. Available at `http://dx.doi.org/10.1007/978-3-642-02424-5_3`.

[93] Dragoş Sburlan (2006): *Non-cooperative P Systems with Priorities Characterize PsET0L*. In: *Proceedings of the 6th International Conference on Membrane Computing*, WMC'05, Springer-Verlag, Berlin, Heidelberg, pp. 363–370, doi:10.1007/11603047_25. Available at `http://dx.doi.org/10.1007/11603047_25`.

[94] Dragos Sburlan: *Promoting and Inhibiting Contexts in Membrane Computing*. Ph.D. thesis, University of Seville.

[95] Dragoş Sburlan (2006): *Further Results on P Systems with Promoters/Inhibitors*. *International Journal of Foundations of Computer Science* 17(01), pp. 205–221, doi:10.1142/S0129054106003772. Available at `http://www.worldscientific.com/doi/abs/10.1142/S0129054106003772`.

[96] Michael John Sebastian Smith (2008): *Application-Specific Integrated Circuits*, 1st edition. Addison-Wesley Professional.

[97] Petr Sosík & Rudolf Freund (2003): *P Systems without Priorities Are Computationally Universal*. In: *Revised Papers from the International Workshop on Membrane Computing*, WMC-CdeA '02, Springer-Verlag, London, UK, UK, pp. 400–409. Available at `http://dl.acm.org/citation.cfm?id=647270.721876`.

[98] Mark Stamp (2003): *Once Upon a Time-Memory Tradeoff*.

[99] R P Stanley (1986): *Enumerative Combinatorics*. Wadsworth Publ. Co., Belmont, CA, USA.

[100] S. K. Stein (1999): *Mathematics: the Man-made Universe.* New York: McGraw-Hill.

[101] Julian Togelius, Noor Shaker & Joris Dormans (2016): *Grammars and L-systems with applications to vegetation and levels*, pp. 73–98. Springer International Publishing, Cham, doi:10.1007/978-3-319-42716-4_5. Available at `http://dx.doi.org/10.1007/978-3-319-42716-4_5`.

[102] L. Wu Y. Zhang & S. Wang (2011): *Magnetic Resonance Brain Image Classification by an Improved Artificial Bee Colony Algorithm*. *Progress In Electromagnetics Research* 116, pp. 65–79.

[103] Hsu-Chun Yen (2006): *Introduction to Petri Net Theory*. In Zoltán Esik, Carlos Martín-Vide & Victor Mitrana, editors: *Recent Advances*

in Formal Languages and Applications, *Studies in Computational Intelligence* 25, Springer Berlin Heidelberg, pp. 343–373, doi:10.1007/978-3-540-33461-3_14. Available at `http://dx.doi.org/10.1007/978-3-540-33461-3_14`.

[104] Ge-Xiang Zhang, Chun-Xiu Liu & Hai-Na Rong (2010): *Analyzing radar emitter signals with membrane algorithms. Mathematical and Computer Modelling* 52(11–12), pp. 1997 – 2010, doi:10.1016/j.mcm.2010.06.002. Available at `http://www.sciencedirect.com/science/article/pii/S0895717710002736`. The BIC-TA 2009 Special Issue, International Conference on Bio-Inspired Computing: Theory and Applications.

Register