UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Universitat Rovira i Virgili
Facultat de Lletres
Departament de Filologies Romàniques

Membrane Computing: Traces, Neural Inspired Models, Controls

PhD Dissertation

Presented by
Armand-Mihai IONESCU

Supervised by
Victor MITRANA and Takashi YOKOMORI

Tarragona, 2007

ISBN:978-84-691-9877-3/DL:T-127-2009

SUPERVISORS

Professor Victor MITRANA

Grup de Recerca de Linguïstica Matemàtica
Universitat Rovira i Virgili
Pça. Imperial Tàrraco 1
43005 Tarragona
Spain

and

Faculty of Mathematics and Computer Science
Bucharest University
Str. Academiei 14
Bucureşti
Romania

and

Professor Takashi YOKOMORI

Department of Mathematics
School of Education
Waseda University
1-6-1 Nishi-waseda, Shinjuku-ku
169-8050 Tokyo
Japan

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

To my wife, Raluca

Forward

I would like to start this (long) list of persons whom I want to thank with the person that introduced me into the world of Membrane Computing, guided, helped and inspired me all over this years, Dr. Gheorghe Păun. For his constant support I will be always indebted.

I also want to thank my supervisors Dr. Victor Mitrana and Dr. Takashi Yokomori for their precious time spent with my work, for the ideas and encouragements during our meetings, both in Tarragona and Tokyo.

Further thanks go to (all) the Research Group on Mathematical Linguistics starting with the head of the group, Dr. Carlos Martín-Vide, my colleagues from the 3rd Phd School in Formal Languages and Applications: Cristina, Mădălina, Tsena, Carlos, Guangwu, Szilárd, and the other colleagues and friends from the school: Areti, Lilica, Adrian, Cătălin, Peter, Remco, Robert.

This thesis would not be possible without the four year financial support given by the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario (FPU). This scholarship also financed my research visits to Waseda University (Tokyo, Japan), Microsoft Research - University of Trento Centre for Computational and Systems Biology (Trento, Italy), and Cambridge University (Cambridge, UK).

A special thank I address to all the persons I have worked with (not listed above) and from whom I have learned a lot: Matteo Cavaliere, Rudolf Freund, Oscar Ibarra, Tseren-Onolt Ishdorj, Andrei Păun, Mario J. Pérez-Jiménez, Bianca Popa, Dragoş Sburlan, and to Mădălina Barbaiani for the help at the software presented later on in the thesis.

Last, but, of course, not least I want to thank my family. To my parents, Dorina and Jean, for their love, encouragements and the time spent missing me, to my grandmothers for surviving with the too-few-words exchanged in the last four years, and to all other members of my family (including my parents-in-law), for all their support and affection.

I will end with a special message. This thesis is done in memory of my grandfather, Nicolae, who told me, when I left for Spain, that I would not see him again on my returning, and, sadly enough, I did not.

ISBN:978-84-691-9877-3/DL:T-127-2009

**Those who took other inspiration than from nature,
master of masters, were labouring in vain.**

Leonardo da Vinci

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

ISBN:978-84-691-9877-3/DL:T-127-2009

About the Thesis

The present work is dedicated to a very active branch of *natural computing* (which tries to discover the way nature computes, especially at a biological level), namely *membrane computing*, more precisely, to those models of membrane systems mainly inspired from the functioning of the neural cell.

Membrane computing area was initiated by Gh. Păun at the end of 1998 and it deals with seeking of computing models inspired by the structure and functioning of the living cell. The models obtained – in literature they are called *P systems* – process multisets of symbols (which are usually called *objects*) in a distributed parallel manner, inside a membrane structure of a cellular type or of a tissue-like type. There are various classes of P systems many of them being universal from a computational point of view (i.e., equivalent in power with Turing machines).

The present dissertation contributes to membrane computing in three main directions. First, we introduce a new way of defining the result of a computation by means of following the *traces* of a specified object within a cell structure or a neural structure. Then, we get closer to the biology of the brain, considering various ways to control the computation by means of *inhibiting/de-inhibiting* processes. Third, we introduce and investigate in a great – though preliminary, as many issues remain to be clarified – detail a class of P systems inspired from the way neurons cooperate by means of *spikes*, electrical pulses of identical shapes.

A frightening thought for a computer scientist is that there might be completely different ways of designing computing machinery, that we may miss by focusing on incremental improvements of current designs. In fact, we *know* that there exist much better design strategies, since the human brain has information processing capabilities that are in many aspects superior to our current computers. Furthermore the brain does not require expensive programming, debugging or replacement of failed parts, and it consumes just 10-20 Watts of energy.

Unfortunately, most information processing strategies of the brain are still a mystery, and even the most basic questions concerning the organization of the computational units of the brain and the way in which the brain implements learning and memory have not yet been answered. They are waiting to be unraveled by concerted efforts of scientists from many disciplines. Computer science is one of the disciplines from which substantial contributions are expected.

The interest for the various models of P systems does not come only from their direct biological motivation but also from the fact that different variants can compute at the level of Turing machine, where several types of P systems can be

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

used as a framework for devices which solve computationally hard problems in a polynomial time.

Recently, membrane computing also proved to be a fruitful framework for applications in several areas, especially in biology and bio-medicine. The website of the domain, at http://psystems.disco,unimib.it, provides a comprehensive information in this respect.

The present thesis addresses especially problems related to the computing power and he computational efficiency of the models we introduce. For instance, starting with *Chapter 3* - where itineraries (traces) of objects in a membrane structure are considered -, and continuing in *Chapter 4*, several results concerning the power of models involving the neural concept of inhibition/de-inhibition are given altogether with other results concerning the simulation of Boolean gates and circuits.

In *Chapter 5*, we address a new concept in P systems area based on spiking neurons. Here, we formalize the process where a neuron *fires* at certain points in time sending through its axon (and its dendrites) a stereotyped electric pulse which is actually *the action potential* or *the spike*. The size and shape of a spike is independent from the input to the neuron, but the *time* when a neuron fires depends on its input. Hence, we use the time as the support of information.

The study of spiking neural P systems was very active in the last couple of years. We present here only part of the results of research in which we were involved. A simulator for the spiking neural P systems is described in *Chapter 6*.

Some biological information regarding the neural cell as well as some prerequisites from theoretical computer science needed along this work can be found in *Chapter 2*. A short history of the Theory of Computation and Molecular Computing is given in *Chapter 1*.

Conclusions and possible directions for future research are given at the end of each section or chapter.

Large portions of *Chapters 3, 4,* and *5* have appeared in the following papers:

- ***Chapter 3:***
  - [41] M. Ionescu, C. Martín-Vide, Gh. Păun: P systems with symport/antiport rules: The traces of objects, *Grammars*, 5, 2002, 65–79.
  - [39] M. Ionescu, C. Martín-Vide, A. Păun, Gh. Păun: Membrane systems with symport/antiport: (Unexpected) Universality results, *Proceedings of The 8th International Meeting on DNA Based Computers* (M. Hagiya, A. Obuchi, eds.), Sapporo, Japan, 2002, 151–160, and *LNCS* 2568, Springer, Berlin, 2003, 281–290.
  - [40] M. Ionescu, C. Martín-Vide, A. Păun, Gh. Păun: Unexpected universality results for three classes of P systems with symport/antiport, *Natural Computing*, 2(4), 2003, 337–348.

  - [52] G. Liu, M. Ionescu: Further remarks on trace languages in P systems with symport/antiport, *Proceedings of Seventh Workshop on Membrane Computing* (WMC7), Leiden, The Nederlands, 2006, 417–428.

- ***Chapter 4:***
  - [13] M. Cavaliere, M. Ionescu, T.-O. Ishdorj: Inhibiting/de-inhibiting rules in P systems, *Proceedings of Fifth Workshop on Membrane Computing* (WMC5), and *LNCS* 3365, Springer, Berlin, 2005, 224–238.
  - [14] M. Cavaliere, M. Ionescu, T.-O. Ishdorj: Inhibiting/de-inhibiting P systems with active membranes, *Cellular Computing (Complexity Aspects)*, ESP PESC Exploratory Workshop, Fénix Editora, Sevilla, 2005, 117–130.
  - [37] M. Ionescu, T.-O. Ishdorj: Boolean circuits and a DNA algorithm in membrane computing, *Pre-proceedings of the* 6th *Workshop on Membrane Computing*, Vienna, 2005, 410–438, and *LNCS* 3850, Springer, Berlin, 2006, 272–291.

- ***Chapter 5:***
  - [43] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems, *Fundamenta Informaticae*, 71(2-3), 2006, 279–308.
  - [16] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems, *Fundamenta Informaticae*, 75(1-4), 2007, 141–162.
  - [19] H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by (small) spiking neural P systems, *Brainstorming Week on Membrane Computing 2006*, and *Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, 94–105.
  - [18] H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: Universality and languages, submitted.
  - [42] M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems, *Proceedings of The 12th International Meeting on DNA Computing* (DNA12) (C. Mao, T. Yokomori, B.-T. Zhang, eds.), Seul, June 2006, 32–42.
  - [44] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with an exhaustive use of rules, *International Journal of Unconventional Computing*, 3, 2007, 135–153.
  - [17] H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems, *Proceedings of the Eighth International Conference*

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

on Electronics, Information, and Communication, Ulaanbaatar, Mongolia, June 2006, 49–52.

- [46] M. Ionescu, D. Sburlan: Some applications of spiking neural P systems, *Proceedings of the Eighth Workshop of Membrane Computing*, WMC8, Thessaloniki, Greece, June 2007, 383–394.

There are also briefly mentioned some results, or ideas, or constructions from the following papers:

- [45] M. Ionescu, D. Sburlan: On P systems with promoters/inhibitors, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, 264–280, and *Journal of Universal Computer Science*, 10(5), 2004, 581–599.

- [38] M. Ionescu, T.-O. Ishdorj: Replicative-distribution rules in P systems with active membranes, *Proceedings of First International Colloquium on Theoretical Aspects of Computing*, Guiyang, China, 20-24 September 2004, UNU/IIST Report No. 310, 263–278, and *LNCS* 4705, Springer, Berlin, 2005, 69–84.

- [12] M. Cavaliere, O. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems: Decidability and undecidability, *Proceedings of The 13th International Meeting on DNA Computing* (DNA13), Memphis, Tennessee, USA, June 2007.

- [47] M. Ionescu, D. Sburlan: P systems with adjoining controlled communication rules, *Proceedings of 16th International Symposium on Fundamentals of Computation Theory*, August 27-30, 2007, Budapest, Hungary.

- [25] R. Freund, M. Ionescu, M. Oswald: Extended spiking neural P systems with decaying spikes and/or total spiking, *Proceedings of Automata for Cellular and Molecular Computing*, International Workshop, August 31, 2007, Budapest, Hungary.

ISBN:978-84-691-9877-3/DL:T-127-2009

Contents

About the Thesis 7

1 Introduction 15
1.1 Theoretical Computer Science: Roots and a Short History 15
1.2 Natural Computing: Molecular and Cellular Computing 16

2 Prerequisites 19
2.1 The Cell 19
2.1.1 Singleton Cells, Cells in Multicellular Organisms, Membranes 19
2.1.2 The Neural Cell 20
2.2 Formal Language Prerequisites 22
2.2.1 Languages, Grammars, and Turing Machines 22
2.2.2 Matrix Grammars 23
2.2.3 Register Machines 24
2.2.4 Lindenmayer Systems 26
2.3 Boolean Functions and Circuits 26
2.4 Membrane Computing Prerequisites 27
2.4.1 P Systems 27
2.4.2 P Systems with Symport/Antiport 28
2.4.3 P Systems with Active Membranes 30
2.4.4 Neural-Like P Systems 32

3 Following the Trace of Objects 35
3.1 Trace Languages Considering Multisets of Objects 35
3.1.1 The Generative Power 37
3.1.2 Characterizations of Recursively Enumerable Languages 40
3.1.3 Decreasing the Number of Membranes 44
3.2 Trace Languages Considering Sets of Objects 47
3.3 Remarks and Further Research 50

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

4 Inhibiting/De-Inhibiting P Systems 51
4.1 The Biological Source of the Concept 51
4.2 Inhibiting/De-Inhibiting Rules in P Systems 52
4.2.1 Using One Catalyst: Two Universality Results 54
4.2.2 Using Non-Cooperative Rules and One Switch 57
4.3 Inhibiting/De-Inhibiting P Systems with Active Membranes 58
4.3.1 Simulating Logical Gates 60
4.3.2 Simulating Boolean Circuits 63
4.3.3 Accepting and Generative Universality Results 66
4.3.4 An Efficiency Result for AID P Systems 68
4.4 Remarks and Further Research 70

5 Spiking Neural P Systems 71
5.1 The Biological Source of the Concept 71
5.2 The Initial Model 72
5.2.1 Examples 75
5.2.2 The Computational Power of SN P Systems 80
5.2.3 Remarks and Further Research 95
5.3 On String Languages Generated by SN P Systems 97
5.3.1 An Illustrative Example 98
5.3.2 The Language Generative Power of SN P Systems 100
5.3.3 Remarks and Future Research 113
5.4 Trace Languages Associated to SN P Systems 114
5.4.1 Two Examples 116
5.4.2 The Power of SN P Systems as Trace Generators 118
5.4.3 Remarks and Further Research 123
5.5 Spiking Neural P Systems with Extended Rules 123
5.5.1 Extended SN P Systems as Number Generators 124
5.5.2 Languages in the Restricted Case 126
5.5.3 Languages in the Non-Restricted Case 131
5.5.4 Remarks and Further Research 138
5.6 Two Small Universal SN P Systems 139
5.7 Using the Rules in an Exhaustive Way 141
5.7.1 Examples 143
5.7.2 Computational Completeness 144
5.7.3 Remarks and Further Research 151
5.8 Spiking Neural P Systems with Self-Activation 152
5.8.1 An Example 155
5.8.2 A Formal Presentation 157
5.8.3 Remarks and Further Research 159
5.9 Some Applications of SN P Systems 160
5.9.1 Simulating Logical Gates and Circuits 160
5.9.2 A Sorting Algorithm 164
5.9.3 Remarks and Further Research 166

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

6 An SN P Systems Simulator **167**
6.1 Construction Area 167
6.2 Simulation Area 168

Bibliography **170**

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Chapter 1

Introduction

1.1 Theoretical Computer Science: Roots and a Short History

Trying to put together and to summarize more than 7 thousands years of computing is always a difficult task both because of the variety of the bibliographical sources and the multitude of important steps to achieve computing as it is nowadays. What we do in this section is just pointing some major steps humanity made starting 5000 BC, (probably) in Asia, where a first mechanical device to aid calculation - the abacus - was used, until the parallel computations we are performing on our parallel machines.

We go beyond to one of the first algorithms - the Sieve of Erathostehenes - used to determine the prime numbers, or the work on logarithms started by Jaina mathematicians (between 200 BC - 400 BC), and continued by Muslim mathematicians (13th century) and John Napier (16th century), and we mark the year 1641 as being the moment when a first mechanical machine (used for adding) was made by the French mathematician and philosopher Blaise Pascal.

Charles Babbage is the name of the English mathematician, philosopher, and mechanical engineer who designed and built Difference Engine - a mechanical calculator -, and who originated the idea of a programmable computer. Later on, in 1936, Alain Turing's concept of Turing machine led to the construction of the first computers.

In 1944, Mauchly, Eckert, and John von Neumann were working on a stored-program electronic computer, the EDVAC (Electronic Dicrete Variable Automatic Computer). Von Neumann's report, "First Draft of a Report on the EDVAC", was very influential and contains many of the ideas still used in most modern digital computers, including a mergesort routine.

In the history of computing, electronic devices are, as seen before, one of the attempts of mankind of doing computation using the best technology available. It is true that their appearance has revolutionized computing but this is no reason to consider them as the ending point in the long evolution chain of computing.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Electronic computers have their limitations due to the fact that for the moment some physical barriers cannot be reached (for example, since 2002 the clock speed of the processor has improved less than 20%/year, after a long period characterized by around 50%/year).

A promising direction in theoretical computer science for the next generation of computing devices is *natural computing*, a new and fast growing field of interdisciplinary research driven by the idea that natural processes (or small "nature toolboxes") can be used for implementing computations.

1.2 Natural Computing: Molecular and Cellular Computing

Natural Computing is a widespread notion referring to computing performed in nature or by nature and computing inspired by nature. Our current understanding of this research area includes fields like *Quantum Computing*, *Evolutionary Computing*, *Neural Networks*, *Molecular* and *Cellular Computing*. Quantum Computing uses quantum parallelism to perform computations, while Evolutionary Computing deals with concepts as mutation, recombination, and natural selection from biology to design new ways to perform computations. Neural Networks construct computational paradigms inspired by the highly interconnected neural structures in the brain and in the nervous system.

One of the first cellular computing models, which has as the base of inspiration the brain (the way neurons are connected in the nervous system) was proposed in 1943 by Warren S. McCulloch and Walter Pitts (MIT) in the paper: "A logical calculus of the ideas immanent in nervous activity" [21].

In 1947 John von Neumann introduced the notion of *Cellular Automata* in his attempt to develop an abstract model of self-reproduction in biology. The question whether or not it is possible to construct robots that can construct identical robots, hence with the same complexity, was the main issue cellular automata focused on those times. Interesting enough, the model proposed by von Neumann gave a positive answer to this question.

In its book *Automata Studies* ([49]) Kleene addressed in 1956 the topic of the "Representation of the events in nerve nets and finite automata", while Feynman, in his famous talk "There is plenty of room at the bottom" given in 1959 (during the meeting of American Physical Society) proposed to manipulate directly atoms to construct nano-machines (including nano-computing machines).

In 1968 the biologist Aristid Lindenmayer introduced a mathematical model of development of multicellular organisms. What is extremely interesting here is that the development happens in *parallel* everywhere in the organism, and *distributed* to all its parts ([51]). Few years later, in 1973, C. Bennett discussed how to perform computations on a molecular scale (see [9]) proposing to use RNA molecules as a physical medium for implementing computations.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

It was in 1982 when Richard Feynman proposed the exploration of the idea of a *quantum computer*. The Nobel prize-winning physicist based its theory on the laws of quantum mechanics which assume unintuitive principles that may help in overpassing the restrictions of the current computer.

Starting 1987, when T. Head presented the *splicing systems*, a theoretical computational model having as core concept the recombination of DNA molecules, and continuing in 1994 with Adleman's revolutionary experiment (where an instance of the *Hamiltonian path problem* is solved in laboratory using DNA molecules and biomolecular operations), molecular computing attracted more and more attention and supported hopes in giving the solutions to break the current constraints of electronic computers.

A possible further support for those hopes can be given within the relatively new born area of *Membrane Computing* (the models investigated in this area are also known as *P systems*). The area was initiated in 1998, when Gheorghe Păun proposed in the paper "Computing with membranes" ([68]) an abstract cellular computing model based on biological single living cell compartmental structure and the chemical evolution, communication, and interaction within its compartments. The core structural feature is that of a membrane by which a system is divided into compartments where chemical reactions take place. These reactions transform the inner multisets of objects into new objects sometimes changing the location of the latter ones into neighboring compartments or the environment.

We conclude this introductory chapter with the remark that in 2003 *Thomson Institute for Scientific Information*, *ISI*, has nominated membrane computing as *fast emerging research front in computer science* with the initial paper considered *fast breaking paper*.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

ISBN:978-84-691-9877-3/DL:T-127-2009

Chapter 2

Prerequisites

The present chapter begins with a brief survey to the biological notions used in our work and continues with basics of theory of computing and Membrane Computing sufficient to understand the following chapters.

After presenting some details on various types of cells we give the elementary information from Formal Language Theory and we continue presenting Chomsky and matrix grammars, register machines, Lindenmayer systems and some notions of Computational Complexity.

The last section of the chapter is dedicated to the presentation of the - already classical - models of P systems used later in introducing the new models and concepts this thesis is presenting.

2.1 The Cell

In this section we give some biological details about the anatomy of the cells with a focus on prokaryotic, eukaryotic, and neural cells emphasizing the features we are using as source of inspiration in our work.

2.1.1 Singleton Cells, Cells in Multicellular Organisms, Membranes

First discovered and named by Robert Hooke in 1663 the cell is also known as the "building block of life". The cells are amazing self-contained and self-maintained units which store different types of sets of instructions for various types of activities as feeding, reproduction, metabolic actions, response to external and internal stimuli and so on.

All the containment of a cell is protected by a cell *surface membrane* (Fig. 2.1) that contains proteins and a lipid bilayer.

There are two types of cells: *eukaryotic* and *prokaryotic*. Prokaryotic cells are smaller, usually singletons, with a not so compartmentalized structure, while eukaryotic (Figure 2.2) cells are usually found in multicellular organisms. Except

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

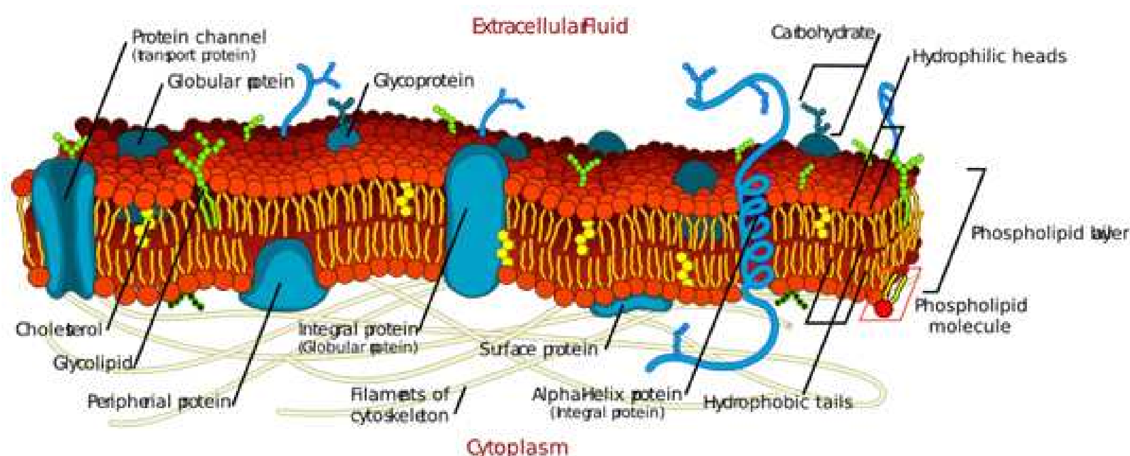


Figure 2.1: A cell membrane.

the membrane the other features that they have in common is the DNA, the cytoplasm, and the ribosomes.

A significant difference between the two types of cells is that the eukaryotic ones contain membrane-bound compartments in which specific metabolic activities take place. Most important among these is the presence of a cell nucleus, a membrane-delineated compartment that houses the eukaryotic cell's DNA. In Figure 2.2 such a cell is given where subcellular components can be observed: (1) nucleolus, (2) nucleus, (3) ribosome, (4) vesicle, (5) rough endoplasmic reticulum (ER), (6) Golgi apparatus, (7) cytoskeleton, (8) smooth ER, (9) mitochondria, (10) vacuole, (11) cytoplasm, (12) lysosome, (13) centrioles.

The shapes of cells are quite varied with some, such as neurons, being longer than they are wide and others, such as parenchyma (a common type of plant cell) and erythrocytes (red blood cells) having regular dimensions. Some cells are encased in a rigid wall, which constrains their shape, while others have a flexible cell membrane (and no rigid cell wall).

2.1.2 The Neural Cell

In [69] there are also considered, besides cell-like structures, systems of membranes with the membranes arranged in a network as the neurons in the brain. We briefly present here the structure and the functioning of a neuron, other more particular biological features being presented at the beginning of each chapter to better emphasize the biological source of each concept considered.

Over the past hundred years, biological research has accumulated an enor-

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

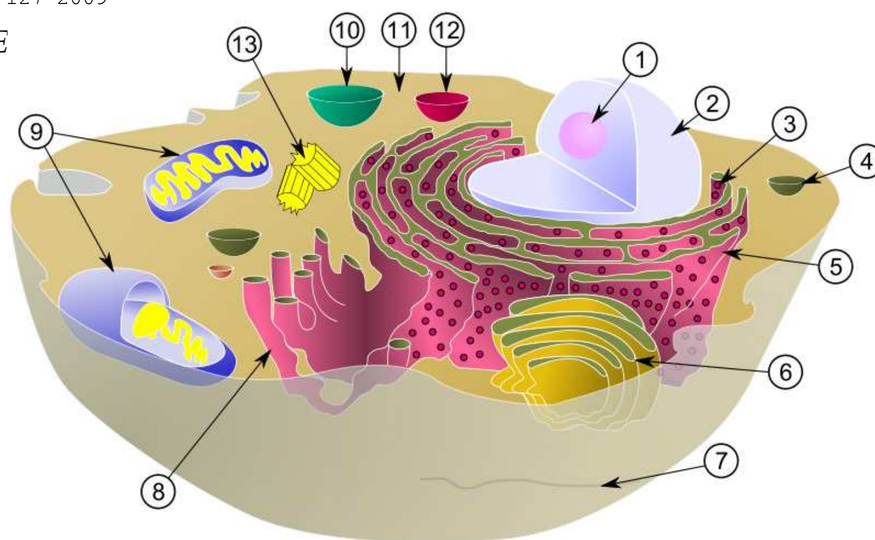


Figure 2.2: An eukaryotic cell.

mous amount of detailed knowledge about the structure and function of the brain. The elementary processing units in the central nervous system are neurons which are connected to each other in an intricate pattern. In reality, cortical neurons and their connections are packed into a dense network with more than 10^14 cell bodies and several kilometers of "wires" per cubic millimeter.

A typical neuron can be divided into three functionally distinct parts, called *dendrites*, *soma*, and *axon*; see Figure 2.3. Roughly speaking, the dendrites play the role of the 'input device' that collects signals from other neurons and transmits them to the soma. The soma is the 'central processing unit' that performs an important processing step: If the total input exceeds a certain threshold, then an output signal is generated. The output signal is taken over by the 'output device', the axon, which delivers the signal to other neurons.

The junction between two neurons is called a *synapse*. Let us suppose that a neuron sends a signal across a synapse. It is common to refer to the sending neuron as the presynaptic cell and to the receiving neuron as the postsynaptic cell. A single neuron in vertebrate cortex often connects to more than 10^4 postsynaptic neurons. Many of its axonal branches end in the direct neighborhood of the neuron, but the axon can also stretch over several centimeters so as to reach to neurons in other areas of the brain.

The transmission of impulses from one neuron to another one is done, roughly speaking, in the following way. A neuron will "fire" only if it gets sufficient excitation through its dendrites. These excitations should come more or less together, in a short period of time, called the *period of latent summation*. The input impulses can be of two types, *excitatory* and *inhibitory*, and in order to get the neuron excited it is necessary that the impulses exceeds a given *threshold* specific to the neuron.

After firing a neuron, there is a small interval of time necessary to synthesize the impulse to be transmitted to the neighboring neurons through the axon; also, there is a small interval of time necessary for the impulse to reach the endbulbs of

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

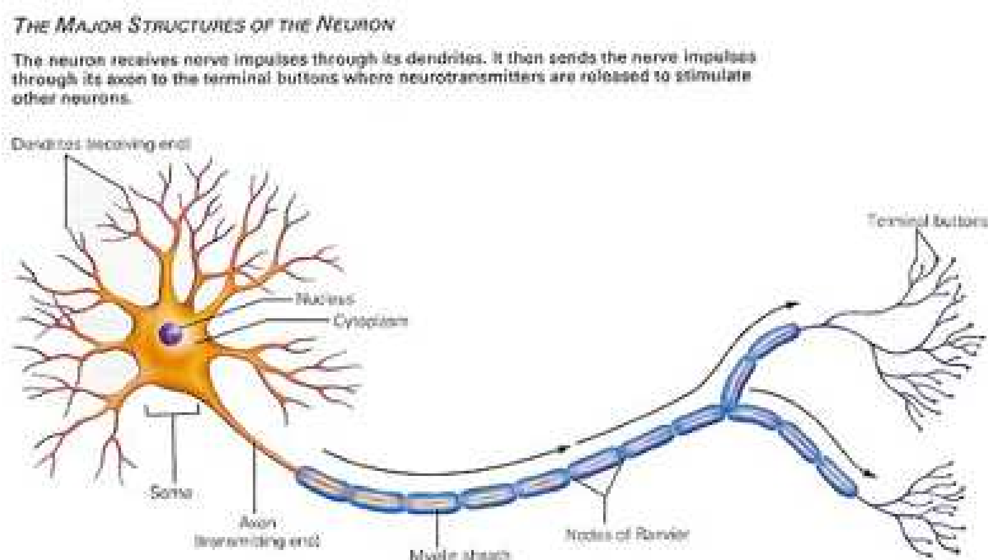


Figure 2.3: The major structures of the neuron.

the axon. After an impulse is transmitted through the axon, there is a time called the *refractory period* during which the axon cannot transmit another impulse.

For more details about neural biology we refer to [81] and [86].

We also remind the reader that more neural biological information can be found in Sections 4.1 and 5.1.

2.2 Formal Language Prerequisites

2.2.1 Languages, Grammars, and Turing Machines

The reader is ass

An *alphabet* is a finite set of symbols (letters) and a word (string) over an alphabet Σ is a finite sequence of letters from Σ. We denote the empty word by λ, the length of a word w by $|w|$, and the number of occurrences of a symbol a in w by $|w|_a$. The (con)catenation of two words x and y is denoted by $x \cdot y$ or simply xy.

A *language* over Σ is a (possibly infinite) set of words over Σ. The language consisting of all words over Σ is denoted by Σ^*, and Σ^+ denotes the language $\Sigma^* - \{\lambda\}$. A set of languages containing at least one language not equal to $\emptyset$ or $\{\lambda\}$ is also called a family of languages. We denote by REG, LIN, CF, CS, RE the families of languages generated by regular, linear, context-free, context-sensitive, and of arbitrary grammars, respectively (RE stands for recursively enumerable languages). By FIN we denote the family of finite languages. The following

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

strict inclusions hold:

$$FIN \subset REG \subset LIN \subset CF \subset CS \subset RE.$$

This is the Chomsky hierarchy. For a family FL of languages, NFL denotes the family of length sets of languages in FL. Therefore, NRE is the family of Turing computable sets of natural numbers. For $\Sigma = \{a_1, \cdots, a_n\}$, the *Parikh mapping* associated with Σ is the mapping $\Psi_\Sigma : \Sigma^* \rightarrow N$ defined by $\Psi_\Sigma(x) = (|x|_{a_1}, \cdots, |x|_{a_n})$ for each $x \in \Sigma^*$. The Parikh images of languages in RE is denoted by $PsRE$ (this is the family of all recursively enumerable sets of vectors of natural numbers). The multisets over a given finite support (alphabet) are represented by strings of symbols. The order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string. Clearly, using strings is only one of many ways to specify multisets (for more details we refer the reader to [11]).

We also mention here the notion of morphism and inverse morphism. For a morphism $h : V^* \rightarrow U^*$, the inverse morphism $h^{-1} : U^* \rightarrow 2^{V^*}$ is defined by $h^{-1}(y) = \{x \in V^* \mid h(x) = y\}, y \in U^*$.

The *left derivative* of a language $L \subseteq V^*$ with respect to a string $x \in V^*$ is $\partial_1^l(L) = \{w \in V^* \mid xw \in L\}$. The *right derivative* is defined in the same manner: $\partial_1^r(L) = \{w \in V^* \mid wx \in L\}$.

2.2.2 Matrix Grammars

We recall here the notion of matrix grammar because we will use in our work the characterization of recursively enumerable languages by means of *matrix grammars with appearance checking*.

Such a grammar is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i'', w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \rightarrow x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied – therefore we say that these rules are applied in the *appearance checking* mode.) If the set F is empty, then the grammar is said to be without appearance checking.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where "$\Longrightarrow^*$" is the reflexive and transitive closure of a relation "$\Longrightarrow$". The family of languages of this form is denoted by MAT_{ac}. It is known that $CF \subset MAT \subset$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$MAT_{ac} = RE, NREG = NCF = NMAT \subset NCS$ (for instance, the one-letter languages in MAT are known to be regular.).

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are in one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1 (that is why one uses to write it in the form $(S \to X_{init}A_{init})$, in order to fix the symbols X, A present in it), and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; $\#$ is a trap-symbol, because once introduced, it is never removed. A matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar there is an equivalent matrix grammar in the binary normal form. Details can be found in [22].

There exists even a more restricted normal form for matrix grammars with appearance checking. We say that a matrix grammar $G = (N, T, S, M, F)$ is in the *Z-binary normal form* if $N = N_1 \cup N_2 \cup \{S, Z, \#\}$ is the union of mutually disjoint sets, and the matrices in M are of one of the following forms:

1. $(S \to XA)$ with $X \in N_1, A \in N_2$;
2. $(X \to Y, A \to x)$ with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$;
3. $(X \to Y, A \to \#)$ with $X \in N_1, Y \in N_1 \cup \{Z\}, A \in N_2$;
4. $(Z \to \lambda)$.

Moreover, there is only one matrix of type 1, F consists exactly of all rules $A \to \#$ appearing in matrices of type 3 ($\#$ is a trap-symbol, if it is introduced, then it cannot be removed) and, if a sentential form generated by G contains the symbol Z, then it is of the form Zw, for some $w \in (T \cup \{\#\})^*$. The matrix of type 4 is used only once, in the last step of a derivation.

For each $L \in RE$ there is a matrix grammar with appearance checking in the Z-binary normal form such that $L = L(G)$. More details are available in [22].

2.2.3 Register Machines

We will also use in our work register machines, that is why we shortly recall here this notion (the reader can find more details in [60]). A register machine runs a program consisting of labeled instructions of several simple types. Several variants of register machines were shown to be computationally universal.

An *m-register machine* is a construct $M = (m, I, l_0, l_h)$, where:

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

- m is the number of registers,
- I is a set of labeled instructions of the form $l_i : (\texttt{OP}(r), l_j, l_k)$, where $\texttt{OP}(r)$ is an operation on register r of M, and l_i, l_j, l_k are labels from the set $lab(M)$ (that is the set of labels associated to the instructions, in a one-to-one manner),
- l_0 is the label of the initial instruction, and
- l_h is the label of the halting instruction.

The machine is capable of the following instructions:

$(\texttt{ADD}(r), l_j, l_k)$: Add one to the content of register r and proceed, in a non-deterministic way, to instruction with label l_j or to instruction with label l_k; in the deterministic variants usually considered in the literature we demand $l_j = l_k$.

$(\texttt{SUB}(r), l_j, l_k)$: If register r is not empty, then subtract one from its contents and go to instruction with label l_j, otherwise proceed to instruction with label l_k.

HALT : This instruction stops the machine. This additional instruction can only be assigned to the final label l_h.

A register machine M computes a number n in the following way: we start with all registers empty (i.e., storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers); if we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. The set of all numbers computed by M is denoted by $N(M)$.

It is known (see, e.g., [60]) that register machines (even with a small number of registers), compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its contents.

A register machine can also work in the *accepting* mode: a number n is introduced in the first register (all other registers are empty) and we start computing with the instruction with label l_0; if the computation eventually halts, then the number n is accepted.

Register machines are universal also in the accepting mode; moreover, this is true even for deterministic machines, having ADD rules of the form $l_i : (\texttt{ADD}(r), l_j, l_k)$ with $l_j = l_k$: after adding 1 to register r we pass precisely to one instruction, without any choice (in such a case, the instruction is written in the form $l_i : (\texttt{ADD}(r), l_j)$).

Again, without loss of generality, we may assume that in the halting configuration all registers are empty.

A deterministic *m*-register machine can also analyze an input $(m_1, ..., m_\alpha) \in \mathbf{N}^\alpha$, introduced in registers 1 to α, which is accepted if and only if the register machine finally stops by the halt instruction with all its registers being empty

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

(this last requirement is not necessary). If the machine does not halt, then the analysis was not successful.

2.2.4 Lindenmayer Systems

An $ET0L$ system is a construct $G = (\Sigma, T, H, w')$, where the components fulfill the following requirements: Σ is the alphabet; $T \subseteq \Sigma$ is the terminal alphabet; H is a finite set of of finite substitutions (tables) $H = \{h_1, h_2, \cdots, h_t\}$ (t is the number of tables); each $h_i \in H$ can be represented by a list of context-free rules $A \to x$, such that $A \in \Sigma$ and $x \in \Sigma^*$ (this list for h_i should satisfy that each symbol of Σ appears as the left side of some rule in h_i); $w' \in \Sigma^*$ is the axiom.

G defines a derivation relation $\Rightarrow$ by $x \Rightarrow y$ iff $y \in h_i(x)$, for some $1 \leq i \leq t$, where h_i is interpreted as a substitution mapping.

The language generated by G is $L(G) = \{w \in \Sigma^* \mid w' \Rightarrow^* w\} \cap T^*$, where $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

In what follows, for simplicity, we also say that a language L is in $ET0L$ (i.e., $L \in ET0L$) if it can be generated by some ET0L systems.

Of interest for our purpose is the following result:

$$CF \subset ET0L \subset CS \subset RE.$$

Moreover it is known that for each $L \in ET0L$, there exists an ET0L system G', with only 2 tables, such that $L = L(G')$ (see [79]).

We also need the following normal form for ET0L systems (for the proof we refer to [3]).

Lemma 2.2.1 (Normal form)
For each $L \in ET0L$ there is an extended tabled Lindenmayer system $G = (\Sigma, T, H, w')$ with 2 tables ($H = \{h_1, h_2\}$) generating L, such that the terminals are only trivially rewritten: for each $a \in T$ if $(a \to \alpha) \in h_1 \cup h_2$, then $\alpha = a$.

2.3 Boolean Functions and Circuits

An n-ary **Boolean function** is a function $f : \{true, false\}^n \mapsto \{true, false\}$.
$\neg$ (negation) is a unary Boolean function (the other unary functions are the constant functions and identity function). We say that Boolean expression φ with variables $x_1, \ldots, x_n$ *expresses* the n-ary Boolean function f if, for any n-tuple of truth values $t = (t_1, \cdots, t_n)$, $f(t)$ is $true$ if $T \vDash \varphi$, and $f(t)$ is false if $T \nvDash \varphi$, where $T(x) = t_i$ for $i = 1, \ldots, n$.

There are three primary Boolean functions that are widely used: NOT, AND, and OR. The NOT function - this is a just a negation; the output is the opposite of the input - takes only one input, so it is called a unary function or operator. The output is true when the input is false, and vice-versa. The output of an AND function is true only if its first input and its second input and its third input (etc.)

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

are all true. The output of an OR function is true if the first input is true or the second input is true or the third input is true (again, etc.). Both AND and OR can have any number of inputs, with a minimum of two.

Any n-ary Boolean function f can be expressed as a Boolean expression φ_f involving variables $x_1, \ldots, x_n$.

There is a potentially more economical way than expressions for representing Boolean functions, namely *Boolean circuits*. A Boolean circuit is a graph $C = (V, E)$, where the nodes in $V = \{1, \ldots, n\}$ are called the *gates* of C. Graph C has a rather special structure. First, there are no cycles in the graph, so we can assume that all edges are of the form (i, j), where $i < j$. All nodes in the graph have the "in-degree" (number of incoming edges) equal to 0, 1, or 2. Also, each gate $i \in V$ has a *sort* $s(i)$ associated with it, where $s(i) \in \{true, false, \vee, \wedge, \neg\} \cup \{x_1, x_2, \ldots\}$. If $s(i) \in \{true, false\} \cup \{x_1, x_2, \ldots\}$, then the in-degree of i is 0, that is, i must have no incoming edges. Gates with no incoming edges are called the *inputs* of C. If $s(i) = \neg$, then i has in-degree one. If $s(i) \in \{\vee, \wedge\}$, then the in-degree of i must be two. Finally, node n (the largest numbered gate in the circuit, which necessarily has no outgoing edges) is called the *output gate* of the circuit.

This concludes our definition of the *syntax* of circuits. The *semantics* of circuits specifies a truth value for each appropriate truth assignment. We let $X(C)$ be the set of all Boolean variables that appear in the circuit C (that is, $X(C) = \{x \in X \mid s(i) = x$ for some gate i of $C\}$). We say that a truth assignment T is *appropriate* for C if it is defined for all variables in $X(C)$. Given such a T, *the truth value of gate* $i \in V$, $T(i)$, is defined, by induction on i, as follows: If $s(i) = true$, then $T(i) = true$, and similarly if $s(i) = false$. If $s(i) \in X$, then $T(i) = T(s(i))$. If now $s(i) = \neg$, there is a unique gate $j < i$ such that $(j, i) \in E$. By induction, we know $T(j)$, and then $T(i)$ is $true$ if $T(j) = false$, and vice-versa. If $s(i) = \vee$, then there are two edges (j, i) and (j', i) entering i. $T(i)$ is then $true$ if only if at least one of $T(j)$, $T(j')$ is $true$. If $s(i) = \wedge$, then $T(i)$ is $true$ if only if both $T(j)$ and $T(j')$ are $true$, where (j, i) and (j', i) are the incoming edges. Finally, *the value of the circuit*, $T(C)$, is $T(n)$, where n is the output gate.

2.4 Membrane Computing Prerequisites

2.4.1 P Systems

For the reader convenience, we recall the fact that P systems are distributed parallel computing models which abstract from the structure and the functioning of the living cells. In short, we have a *membrane structure*, consisting of several membranes embedded in a main membrane (called the *skin*) and delimiting *regions* where multisets of certain *objects* are placed (Figure 2.4 illustrates these notions).

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

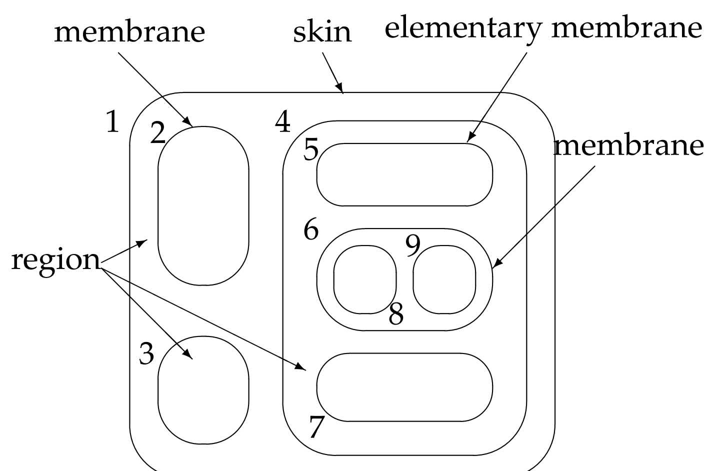


Figure 2.4: A membrane structure

A *membrane structure* is represented by a Venn diagram and is identified by a string of correctly matching parentheses, with a unique external pair of parentheses; this external pair of parentheses corresponds to the external membrane, called the *skin*. A membrane without any other membrane inside is said to be *elementary*. We say that the number of membranes is the *degree* of the membrane structure, while the height of the tree associated in the usual way with the structure is its *depth*. The membranes delimit *regions* (each region is bounded by a membrane and the immediately lower membranes, if there are any). In these regions we place *objects*, which are represented by symbols of an alphabet. Several copies of the same object can be present in a region, so we work with *multisets* of objects.

The objects evolve according to given *evolution rules*, which are applied nondeterministically (the rules to be used and the objects to evolve are randomly chosen) in a maximally parallel manner (in each step, all objects which can evolve must do it). The objects can also be communicated from a region to another one. In this way, we get *transitions* from a *configuration* of the system to the next one. A sequence of transitions constitutes a *computation*; with each *halting computation* we associate a *result*, the number of objects from a specified *output membrane*.

In literature there are known very many different variants of P systems. Details can be found at [85]

2.4.2 P Systems with Symport/Antiport

We start from the biological observation [1], [6] that there are many cases where two chemicals pass at the same time through a membrane, with the help of each other, either in the same direction, or in opposite directions; in the former case we say that we have a *symport*, in the latter case we have an *antiport*.

Mathematically, we can capture the idea of symport by considering rules of

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

the form (ab, in) and (ab, out) associated with a membrane, and stating that the objects a, b can enter, respectively, exit together the membrane. For antiport we consider rules of the form $(a, out; b, in)$, stating that a exits and at the same time b enters the membrane. Generalizing such kinds of rules, we can consider rules of the unrestricted forms (x, in), (x, out) (generalized symport) and $(x, out; y, in)$ (generalized antiport), where x, y are strings representing multisets of objects, without any restriction on the length of these strings.

Based on rules of these types, in [65] there are considered *membrane systems* (currently called *P systems*) *with symport/antiport* in the form of constructs

$$\Pi = (V, \mu, w_1, \ldots, w_m, E, R_1, \ldots, R_m, i_o),$$

where:

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

2. μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) injectively labeled with $1, 2, \ldots, m$; m is called the *degree* of Π;

3. $w_i, 1 \leq i \leq m,$ are strings over V which represent multisets of objects associated with the regions $1, 2, \ldots, m$ of μ, present in the system at the beginning of a computation;

4. $E \subseteq V$ is the set of objects which are supposed to continuously appear in the environment in arbitrarily many copies;

5. $R_1, \ldots, R_m$ are finite sets of symport and antiport rules over the alphabet V associated with membranes $1, 2, \ldots, m$ of μ;

6. i_o is the label of an elementary membrane of μ (the *output membrane*).

For a symport rule (x, in) or (x, out), we say that $|x|$ is the *weight* of the rule. The *weight* of an antiport rule $(x, out; y, in)$ is $\max(|x|, |y|)$.

The rules from a set R_i are used with respect to membrane i as explained above. In the case of (x, in), the multiset of objects x enters the region defined by the membrane, from the immediately upper region; this is the environment when the rule is associated with the skin membrane. In the case of (x, out), the objects specified by x are sent out of membrane i, into the region immediately outside; this is the environment in the case of the skin membrane. The use of a rule $(x, out; y, in)$ means expelling from membrane i the objects specified by x at the same time with bringing in membrane i the objects specified by y. The objects from E are supposed to appear in arbitrarily many copies in the environment (because we only move objects from a membrane to another membrane, hence we do not create new objects in the system, we need a supply of objects in order to compute with arbitrarily large multisets.) The rules are used in the nondeterministic maximally parallel manner specific to P systems with symbol-objects

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

(the objects to evolve and the rules by which these objects evolve are nondeterministically chosen, but the set of rules used at any step is maximal, no further object can evolve at the same time). In this way, we obtain transitions between the configurations of the system. A configuration is described by the m-tuple of the multisets of objects present in the m regions of the system, as well as the multiset of objects which were sent out of the system during the computation, others than the objects appearing in the set E; it is important to keep track of such objects because they appear in a finite number of copies in the initial configuration and can enter again the system. We do not need to take care of the objects from E which leave the system because they appear in arbitrarily many copies in the environment (the environment is supposed inexhaustible, irrespective how many copies of an object from E are introduced into the system, still arbitrarily many remain in the environment). The initial configuration is $(w_1, \ldots, w_m, \lambda)$. A sequence of transitions is called a computation, and with any halting computation we associate an output, in the form of the number of objects present in membrane i_o in the halting configuration. The set of these numbers computed by a system Π is denoted by $N(\Pi)$. The family of all sets $N(\Pi)$, computed by systems Π of degree at most $m \geq 1$, using symport rules of weight at most p and antiport rules of weight at most q, is denoted by $NP_m(sym_p, anti_q)$; when any of the parameters m, p, q is not bounded, we replace it with $*$.

Next the current best results in P systems with symport/antiport are given. We start with a result independently given in [28], [29], and [31] - one membrane is enough:

Theorem 2.4.1 $NRE = NP_1(sym_1, anti_2)$.

If only the symport operation is used, paying in the number of membranes, the following result is given in [30]:

Theorem 2.4.2 $NRE = NP_2(sym_3)$.

The symport/antiport rules can have associated *promoters* or *inhibitors* which control their application. For instance $(x, in)_a \in R_i$ can be applied only if the promoting object a is present in the region i, while $(x, in)_{\neg a} \in R_i$ can be applied only if the inhibiting object a is not present in the region i.

2.4.3 P Systems with Active Membranes

In this subsection, we describe P systems with active membranes following the concept defined in [69], where more details can also be found.

Informally speaking, in P systems with active membranes one uses the following types of rules: (a_0) multiset rewriting rules, (b_0) rules for introducing objects into membranes, (c_0) rules for sending objects out of membranes, (d_0) rules for dissolving membranes, (e_0) rules for dividing elementary membranes, and (f_0) rules for dividing non-elementary membranes, see [5]. In these rules,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

a single object is involved. Furthermore, (g_0) membrane merging rules, (h_0) membrane separation rules, and (i_0) membrane release rules were introduced in [4], (k_0) replicative-distribution rule (for sibling membranes), and (l_0) replicative-distribution rule (for nested membranes) were introduced in [38]. Their common feature is that they involve multisets of objects. The rules of type (a_0) are applied in a parallel way (all objects which can evolve by such rules have to evolve), while the rules of types (b_0), (c_0), (d_0), (e_0), (f_0), (g_0), (h_0), (i_0), (k_0), and (l_0) are used sequentially, in the sense that one membrane can be used by at most one rule of these types at a time. In total, the rules are used in the non-deterministic maximally parallel manner: all objects and all membranes which can evolve, should evolve.

A P system *with active membranes* (without electrical charges) is a construct

$$\Pi = (O, H, \mu, w_1, \ldots, w_m, R),$$

where:

1. $m \geq 1$ is the initial degree of the system;
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure*, consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H;
5. $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
6. R is a finite set of *developmental rules*, of the types $(a_0) - (l_0)$ mentioned above. We define here only a few of these types:

   (b_0) $r : a[\ ]_h \to [\ b]_h$, for $h \in H, a, b \in O$
   (communication rules; an object is introduced in the membrane during this process);

   (c_0) $r : [\ a\ ]_h \to [\ \ ]_h b$, for $h \in H, a, b \in O$
   (communication rules; an object is sent out of the membrane during this process);

   (f_0) $[\ a]_h \to [\ b]_h [\ c]_h$, for $h \in H, a, b, c \in O$
   (division rules for non-elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label; the object specified in the rule is replaced in the two new membranes by possibly new objects; the remaining objects and membranes contained in this membrane are duplicated, and then are part of the contents of both new copies of the membrane);

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

(l_0) $[\, a[\,]_{h_1}]_{h_2} \rightarrow [\,[\, u]_{h_1}]_{h_2} v$, for $h_1, h_2 \in H, a \in O, u, v \in O^*$
(replicative-distribution rule (for nested membranes); an object is replaced by two multisets, one introduced into a directly inner membrane and the other outside the directly surrounding membrane).

The subscript 0 indicates the fact that we do not use polarization for membranes; as shown in [70], [78] the membranes can have one of the negative, positive, neutral "electrical charges", represented by $-$, $+$, and 0, respectively.)

The result is the multiplicity of objects expelled into the environment during the computation; We can also distinguish these objects and consider as result the vector of natural numbers describing the multiplicity of the objects sent to the environment. The set of vectors computed in this way by all possible halting computations of Π is denoted by $Ps(\Pi)$. By $PsOP_m(r)$ we denote the family of sets $Ps(\Pi)$ computed as described above by P systems with at most m membranes using rules of types listed in r. When the rules of a given type (α_0) are able to change the label(s) of the involved membranes, we denote that type of rules by (α'_0). For example,

(l'_0) $[\, a[\,]_{h_1}]_{h_2} \rightarrow [\,[\, u]_{h_3}]_{h_4} v$, for $h_1, h_2, h_3, h_4 \in H, a \in O, u, v \in O^*$
(replicative-distribution rule (for nested membranes); an object is replicated and distributed into a directly inner membrane and outside the directly surrounding membrane while the labels of membranes change).

P systems with certain combinations of these rules are universal and efficient. Further details can be found in [4, 5, 38, 62].

To understand what solving a problem in a semi-uniform/uniform way means, we briefly recall here some related notions. Consider a decisional problem X. A family $\Pi_X = (\Pi_X(1), \Pi_X(2), \cdots)$ of P systems (with active membranes in our case) is called *semi-uniform* (*uniform*) if its elements are constructible in polynomial time starting from $X(n)$ (from n, respectively), where $X(n)$ denotes the instance of size n of X. We say that X can be solved in polynomial (linear) time by the family Π_X if the system $\Pi_X(n)$ will always stop in a polynomial (linear, respectively) number of steps, sending out the object yes if and only if the instance $X(n)$ has a positive answer. For more details about complexity classes for P systems see [69, 78].

2.4.4 Neural-Like P Systems

We also recall here the initial definition of a neural-like P system as considered in [59], [69] so that we can use it (with some modifications) in the next sections. The basic idea is to consider *cells* related by *synapses* and behaving according to their *states*; the states can model the firing of neurons, depending on the inputs, on the time of the previous firing, etc.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Formally, a *neural-like P system*, of degree $m \geq 1$, is a construct

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_o),$$

where:

1. O is a finite non-empty alphabet (of *objects*, usually called *impulses*);
2. $\sigma_1, \ldots, \sigma_m$ are *cells* (also called *neurons*), of the form

$$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, R_i), 1 \leq i \leq m,$$

   where:

   a) Q_i is a finite set (of *states*);
   b) $s_{i,0} \in Q_i$ is the *initial state*;
   c) $w_{i,0} \in O^*$ is the *initial multiset* of impulses of the cell;
   d) R_i is a finite set of *rules* of the form $sw \rightarrow s'xy_{go}z_{out}$, where $s, s' \in Q_i, w, x \in O^*$, $y_{go} \in (O \times \{go\})^*$, and $z_{out} \in (O \times \{out\})^*$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, 2, \ldots, m\}$ different from i_o;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ (synapses among cells);
4. $i_o \in \{1, 2, \ldots, m\}$ indicates the *output cell*.

The standard rules used in this model are of the form $sw \rightarrow s'w'$, where s, s' are states and w, w' are multisets of impulses. The mark "go" assigned to some elements of w' means that these impulses have to leave immediately the cell and pass to the cells to which we have direct links through synapses. The communication among the cells of the system can be done in a replicative manner (the same object is sent to all adjacent cells), or in a non-replicative manner (the impulses are sent to only one neighboring cell, or can be distributed non-deterministically to the cells to which we have synapses). The objects marked with "out" (they can appear only in the cell i_o) leave the system. The computation is successful only if it halts, reaches a configuration where no rule can be applied.

The sequence of objects (note that they are symbols from an alphabet) sent to the environment from the output cell is the string computed by a halting computation, hence the set of all strings of this type is the language computed/generated by the system.

We will modify below several ingredients of a neural-like P system as above, bringing the model closer to the way the neurons communicate by means of spikes.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

ISBN:978-84-691-9877-3/DL:T-127-2009

Chapter 3

Following the Trace of Objects

This chapter introduces a new way of interpreting the result in a P system with symport/antiport (defined in Section 2.4.2). Instead of the (number of) objects collected in a specified membrane, we consider as the result of a computation the *itineraries* of a certain object through membranes, during a halting computation, written as a coding of the string of labels of the visited membranes.

The family of languages generated in this way by several variants of P systems with traces is investigated with respect to its place in the Chomsky hierarchy.

The study of traces started as a pure theoretical approach, but later on we found out that there exist current technologies which allow the observation of chemicals (from small ions like Ca^{2+} or H^{+} to large macromolecules such as specific proteins, RNAs, or DNA sequences) using the optical microscope. Green fluorescent protein or the light - to name only two - are tools to tag such chemicals in living cells and organisms and to observe their paths.

3.1 Trace Languages Considering Multisets of Objects

Initially introduced in [41] the new class of P systems (where traces of objects were considered) is defined below. Specifically, we consider P systems of the form

$$\Pi = (V, t, T, h, \mu, w_1, \dots, w_m, E, R_1, \dots, R_m),$$

where

1. V is an alphabet;
2. $t \in V$ (a distinguished object, "the traveler");
3. T is an alphabet;
4. $h : \{1, 2, \dots, m\} \longrightarrow T \cup \{\lambda\}$ is a weak coding;

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

5. μ is a membrane structure with m membranes labeled $1, 2, \ldots, m$;

6. $w_1, \ldots, w_m$ are strings representing the multisets of objects present in the m regions of μ;

7. E is the set of objects present in arbitrarily many copies in the environment;

8. $R_1, \ldots, R_m$ are the sets of symport and antiport rules (with promoters or inhibitors) associated with the m membranes.

The traveler is present in exactly one copy in the system, that is, $|w_1 \ldots w_m|_t = 1$ and $t \notin E$.

Let $\sigma = C_1 C_2 \ldots C_k, k \geq 1$, be a halting computation with respect to Π, with $C_1 = (w_1, \ldots, w_m, \lambda)$ the initial configuration, and $C_i = (z_1^{(i)}, \ldots, z_m^{(i)}, z_e^{(i)})$ the configuration at step $i, 1 \leq i \leq k$. If $|z_j^{(i)}|_t = 1$ for some $1 \leq j \leq m$, then we write $C_i(t) = j$ (therefore, $C_i(t)$ is the label of the membrane where t is placed). If $|z_j^{(i)}|_t = 0$ for all $j = 1, 2, \ldots, m$, then we put $C_i(t) = \lambda$. Then, *the trace* of t in the computation σ is

$$trace(t, \sigma) = C_1(t) C_2(t) \ldots C_k(t).$$

The computation σ is said to generate the string $h(trace(t, \sigma))$, hence the language generated by Π is $L(\Pi) = \{h(trace(t, \sigma)) \mid \sigma \text{ is a halting computation in } \Pi\}$.

We denote by $TPP_m(psym_p, panti_q)$ the family of languages generated by P systems with at most m membranes, with symport rules of weight at most p and antiport rules of weight at most q, using promoters; when the rules have associated forbidding contexts (inhibitors) we write *fsym, fanti* instead of *psym, panti*; when the rules are used in the free mode (they have no promoter/inhibitor symbols associated), we remove the initial "p" and "f" from *psym, panti* and *fsym, fanti*. As usual, the subscript m is replaced by $*$ when no bound on the number of used membranes is considered; similarly, if we use symport or antiport rules of an arbitrary weight, then the subscripts p, q are replaced with $*$.

Before starting to investigate the power of our machineries, let us examine an **example**. Consider the system

$$\begin{aligned}\Pi &= (\{d, t\}, t, \{a, b, c\}, h, [\,[\,[\,[\,[\,]_5]_4]_3]_2]_1, t, \\ &\quad \emptyset, \emptyset, \emptyset, \emptyset, \{d\}, R_1, R_2, R_3, R_4, R_5),\end{aligned}$$

with $h(1) = a, h(3) = b, h(5) = c, h(2) = h(4) = \lambda$, and the following sets of rules:

$$\begin{aligned}R_1 = R_3 = R_5 &= \{(t, out),\ (td, in)\}, \\ R_2 = R_4 &= \{(t, in),\ (d, in)\}.\end{aligned}$$

First, the traveler brings $n \geq 1$ copies of d from the environment (each of them enters immediately membrane 2), then the traveler goes to membrane 2. From here, the traveler brings $m \leq n$ copies of d into membrane 3 (each of them enters

ISBN:978-84-691-9877-3/DL:T-127-2009

immediately membrane 4), then the traveler goes to membrane 4 (at that moment, it is possible that some copies of d remain in membrane 2). From membrane 4, all copies of d are carried into membrane 5; the computation stops with the traveler in membrane 4. Thus, for any computation σ of this type, we have

$$trace(t, \sigma) = 1^{n+1}(23)^m(45)^m4, \text{ for some } n \geq 1, m \leq n.$$

The traveler can also end-up in membrane 2, after introducing all copies of d in membrane 3, and returning to membrane 2. In the case of such a computation σ we have

$$trace(t, \sigma) = 1^{n+1}(23)^n2, \text{ for some } n \geq 1.$$

Finally, we can also have the trivial computation where t just enters membrane 2, without any copy of d present in the system, and this leads to $trace(t, \sigma) = 12$.

Consequently,

$$L(\Pi) = \{a^{n+1}b^mc^m \mid n \geq 1, m \leq n\} \cup \{a^{n+1}b^n \mid n \geq 1\} \cup \{a\}.$$

Clearly, this language is not a context-free one. Note that the system Π has only symport rules, and that the rules are freely applied (we use no promoter or inhibitor).

3.1.1 The Generative Power

The example from above proves the relation $TPP_5(sym_2, anti_0) - CF \neq \emptyset$. Clearly, by removing membranes 2, 3 (as well as all their associated elements), we get a system of degree 3 which generates a non-regular language, hence we also have $TPP_3(sym_2, anti_0) - REG \neq \emptyset$. The result for the context-free case can be slightly improved.

Theorem 3.1.1 $TPP_4(sym_2, anti_0) - CF \neq \emptyset$.

Proof Consider the system

$$\Pi = (\{t, d\}, t, \{a, b, c\}, h, [\,[\,[\,[\,\,]_4]_3]_2]_1, t, \emptyset, \emptyset, \emptyset, \{d\}, R_1, R_2, R_3, R_4),$$

with the morphism $h(1) = a, h(2) = b, h(3) = c, h(4) = \lambda$, and with the rules

$$R_1 = R_2 = R_3 = \{(t, out),\ (td, in)\},$$
$$R_4 = \{(t, in)\}.$$

It is easy to see that

$$L(\Pi) \cap a^*(ba)^*(bc)^* = \{a^{n+1}(ba)^m(bc)^r \mid n \geq 1, n \geq m \geq r\},$$

and this is not a context-free language (the traveler ends-up in membrane 4, after carrying copies of d through membranes; some of these copies can be left behind, hence the relations among n, m, r follow). □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

One can see that the power of following the traces of objects in a P system is very large (even for systems without promoters or inhibitors): all one-letter recursively enumerable languages can be obtained in this way. This is an easy consequence of the fact that P systems with symport/antiport rules can generate all recursively enumerable sets of natural numbers. We denote by $1RE$ the family of one-letter recursively enumerable languages, and, based on the results in [57] and [65], we get:

Theorem 3.1.2 $1RE = TPP_5(sym_2, anti_0) = TPP_2(sym_2, anti_2) = TPP_4(psym_2, anti_0) = TPP_3(fsym_2, anti_0)$.

Note that these results do not improve the assertion from Theorem 3.1.1, but imply that all families mentioned in Theorem 3.1.2 contain languages outside the families CS and REC.

The passing from languages over the one-letter alphabet to languages over an arbitrary alphabet does not seem trivial. Actually, the number of symbols appearing in the strings of a language obviously induces an infinite hierarchy with respect to the number of membranes used by a P system able to generate that language: each symbol should correspond to a membrane, hence $L \in TPP(psym_*, panti_*) \cup TPP(fsym_*, fanti_*)$ implies $card(alph(L)) \leq m$ (for $x \in V^*$ we denote $alph(x) = \{a \in V \mid |x|_a \geq 1\}$ and for a language $L \subseteq V^*$ we write $alph(L) = \bigcup_{x \in L} alph(x)$). Otherwise stated, the hierarchies on the number of membranes are infinite for all types of systems (and irrespective of the weights of rules).

The question we now address is weather or not any of the families $TPP_*(sym_p, anti_q)$ equals RE. In the next section we will prove such a result for the case of symport and antiport with promoters or inhibitors, but mentioning that the current best result is given in [31].

A partial result in this respect is the following one: the regular languages can be generated even in the case when no antiport rule is used (and no promoter or inhibitor).

Theorem 3.1.3 $REG \subset TPP_*(sym_2, anti_0)$.

Proof We only have to prove the inclusion, its properness is known from Theorems 3.1.1 and 3.1.2.

Consider a (nondeterministic) finite automaton $A = (K, T, s_0, F, P)$ (the set of states, the alphabet, the initial state, the set of final states, the set of transition rules), with the rules from P given in the style used in [80], that is, in the form $r : sa \rightarrow s'$, for $s, s' \in K, a \in T$. Assume that we have k rules in P, labeled with $r_1, \dots, r_k$.

We construct the P system (of degree $k + 4$)

$$\Pi = (V, t, T, h, \mu, w_1, \dots, w_{k+4}, E, R_1, \dots, R_{k+4}),$$

ISBN:978-84-691-9877-3/DL:T-127-2009

with the following components:

$$
\begin{aligned}
V &= K \cup \{c, d, t\},\\
h &: \{1, 2, \ldots, k+4\} \longrightarrow T,\\
&\quad \text{defined by } h(4+i) = a \text{ for } r_i : sa \to s',\\
&\quad 1 \le i \le k, \text{ and } h(1) = h(2) = h(3) = h(4) = \lambda,\\
\mu &= [\,[\;]_2[\;]_3[\;]_4[\;]_5 \cdots [\;]_{k+4}]_1,\\
w_1 &= c,\\
w_2 &= t,\\
w_3 &= w_4 = \lambda,\\
w_{4+i} &= d, \text{ for all } i = 1, 2, \ldots, k,\\
E &= K,\\
R_1 &= \{(c, out)\} \cup \{(cs, in) \mid s \in K\},\\
R_2 &= \{(cs_0, in),\ (ts_0, out)\},\\
R_3 &= \{(ts, in) \mid s \in F\},\\
R_4 &= \{(s, in) \mid s \in K\} \cup \{(t, in),\ (t, out),\ (d, in),\ (d, out)\},\\
R_{4+i} &= \{(cs', in),\ (ts, in),\ (ts', out) \mid r_i : sa \to s'\}\\
&\quad \cup \{(c, out),\ (td, out)\}, \text{ for all } i = 1, 2, \ldots, k.
\end{aligned}
$$

We start with the traveler symbol in membrane 2 and the "carrier" object c in membrane 1. The carrier will bring states of A from the environment and place them in membranes $5, 6, \ldots, k+4$. Arbitrarily many states are brought in these membranes, but only copies of state s' are introduced in membrane $4+i$ for $r_i : sa \to s'$. If a state s remains in the skin membrane and the carrier c exits again the system, then the state s is immediately moved into the "garbage collector" membrane 4.

At any moment when the state s_0 is brought into the system, it can be introduced into membrane 2, by the rule $(cs_0, in) \in R_2$; in this way, c ends its work (it remains forever in membrane 2), and the traveler can leave this membrane together with s_0. Note that only s_0 is present in the skin membrane – and from now on, only one state will be available in the skin membrane.

Assume that in the skin membrane we have, together with t, a state s such that $r_i : sa \to s'$ appears in P. If s enters membrane 4, then t will pass forever back and forth through membrane 4 by using the rules $(t, in), (t, out) \in R_4$, hence the computation will never end. If we use a rule $(ts, in) \in R_{4+i}$, then at the next step, the rule $(ts', out) \in R_{4+i}$ can be used. This simulates the use of the rule r_i in the automaton A: the traveler has passed through membrane $4+i$, which, by the weak coding h, provides the symbol a parsed by rule r_i, while the state s has been replaced by s'. The work of A is simulated in this manner until reaching a final state, $s \in F$. In that moment, the rule $(ts, in) \in R_3$ can be used, and the computation stops.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

If we do not have sufficient copies of states s' in membranes $4+i$ associated with rules $r_i : sa \to s'$, then, instead of the rule $(ts', out) \in R_{4+i}$ we have to use the rule $(td, out) \in R_{4+i}$ and the computation will never stop: the symbol d will pass forever across membrane 4, by using the rules $(d, in), (d, out) \in R_4$. Similarly, if in the skin membrane we have a state s such that there is no rule $sa \to s'$ in P, then the symbol t has to use forever the rules $(t, in), (t, out) \in R_4$, and the computation will never stop.

Thus, we can simulate in Π all correct parsings of A and, conversely, each halting computation in Π corresponds to a correct parsing in A. That is, $L(A) = L(\Pi)$, which concludes the proof. □

In the previous proof we have essentially used the fact that a finite automaton recognizes a string by scanning it from left to right, symbol by symbol. The proof can probably be extended to other language generating/recognizing devices which characterize strings by a left to right parsing, possibly with a control mechanism stronger than by means of a finite state memory, as in the case of finite automata. It remains a topic for a further research to see how far we can go in this way.

3.1.2 Characterizations of Recursively Enumerable Languages

At the price of using antiport rules of an arbitrary weight (depending on the number of symbols appearing in the strings of our language), as well as promoters or inhibitors, we can generate all recursively enumerable languages.

In the proofs of these results, we essentially use the characterization of recursively enumerable languages by means of *matrix grammars with appearance checking*.

Theorem 3.1.4 $RE = TPP_*(fsym_2, panti_*)$.

Proof We prove only the inclusion $\subseteq$, as the opposite one can be obtained in a standard manner (or we can invoke the Turing-Church thesis).

Consider a language $L \in RE, L \subseteq T^*$, with $T = \{a_1, \dots, a_k\}$. Interpret the strings in T^* as numbers in base $k+1$, in the following sense: with the string $x = a_{i_1} a_{i_2} \dots a_{i_r}, 1 \le i_j \le k, 1 \le j \le r$, we associate the numerical value

$$val(x) = i_1 + i_2(k+1) + i_3(k+1)^2 + \cdots + i_r(k+1)^{r-1}.$$

(Note that we do not use the digit 0 and that the "numbers" from T^* are read in the reverse direction, the most significant digit is the righmost one.) Denote by $val(L)$ the one-letter language $\{a^n \mid n = val(x), x \in L\}$. Because $L \in RE$, we also have $val(L) \in RE$. Take a matrix grammar with appearance checking, $G = (N, \{a\}, S, M, F)$, in the Z-normal form, such that $L(G) = val(L)$ (as mentioned above, $N = N_1 \cup N_2 \cup \{S, Z, \#\}$).

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

We construct the P system (of degree $k+4$)

$$\Pi = (V, t, T, h, \mu, w_1, w_2, \ldots, w_{k+4}, E, R_1, \ldots, R_{k+4}),$$

with

$$\begin{aligned}
V &= N_1 \cup \{X' \mid X \in N_1\} \cup N_2 \cup \{A' \mid A \in N_2\} \\
&\quad \cup \{e_i \mid 1 \le i \le k\} \cup \{a, b, c, d, e, f, g, t, Z, Z'\}, \\
h &: \{1, 2, \ldots, k+4\} \longrightarrow T \text{ defined by } h(4+i) = a_i, 1 \le i \le k, \\
&\quad \text{and } h(1) = h(2) = h(3) = h(4) = \lambda, \\
\mu &= [\,[\,[\,[\,[\;]_4]_3[\,]_5[\;]_6 \cdots [\;]_{k+4}]_2]_1, \\
w_1 &= XA, \text{ for } (S \to XA) \text{ the initial matrix of } G, \\
w_2 &= gt, \\
w_{2+i} &= \lambda,\ 1 \le i \le k+2, \\
E &= V - \{t\}, \\
R_1 &= \{(XA, out; Yx, in) \mid (X \to Y, A \to x) \in M\} \\
&\quad \cup \{(X, out; f, in) \mid X \in N_1\} \\
&\quad \cup \{(X, out; A'X', in), \\
&\qquad (X', out; Ye, in), \\
&\qquad (eA', out), \\
&\qquad (A'A, out; f, in) \mid (X \to Y, A \to \#) \in M\} \\
&\quad \cup \{(f, out; f, in), \\
&\qquad (Z, out; Zb, in), \\
&\qquad (Z, out; cd, in)\} \\
&\quad \cup \{(Z, out; Ze_i, in) \mid 1 \le i \le k\}, \\
R_2 &= \{(a, in)_g, \\
&\qquad (c, in), \\
&\qquad (g, out; d, in), \\
&\qquad (b, out; a, in), \\
&\qquad (a^{k+1}, out; b, in)_c\} \\
&\quad \cup \{(ca^i, out; e_i, in) \mid 1 \le i \le k\}, \\
R_3 &= \{(da, in)\}, \\
R_4 &= \{(da, in), \\
&\qquad (da, out)\}, \\
R_{4+i} &= \{(te_i, in), \\
&\qquad (t, out)\},\ i = 1, 2, \ldots, k.
\end{aligned}$$

Note that there are only two conditionally used rules, the symport rule $(a, in)_g$ and the antiport rule $(a^{k+1}, out; b, in)_c$, both in R_2; thus, the former rule cannot be

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

used after removing g from membrane 2, and the latter rule cannot be used before introducing the symbol c into membrane 2.

Roughly speaking, the system Π works as follows. Membranes 1 and 2 are used in order to generate the language $val(L)$. When the generation of a string $a^n = val(x)$ is finished, hence the symbol Z is produced, we start "translating" the number n in such a way to recover the string x ("codified" by the labels of the membranes $5, 6, \ldots, k+4$ associated with its symbols). This is mainly done by the rules from R_2, which "read" the string x from left to right and make possible the entrance of t into the right membrane.

Here are the details of this process. We start with XA in the skin membrane (and gt in membrane 2), corresponding to the start matrix of G. Assume that in any moment we have here a multiset (described by a string) of the form Xw, for some $X \in N_1, w \in N_2^*$ (at any moment, each copy of a is immediately sent to membrane 2, by the rule $(a, in)_g \in R_2$, hence we ignore in this phase the occurrences of a).

A matrix $(X \rightarrow Y, A \rightarrow x) \in M$ is simulated by the antiport rule $(XA, out; Yx, in) \in R_1$.

A matrix $(X \rightarrow Y, A \rightarrow \#) \in M$, with the second rule used in the appearance checking manner, is simulated in a slightly more complex manner: first, X gets out and brings into the system the symbols A' and X' (by the rule $(X, out; A'X', in) \in R_1$); at the next step, if A is present in the skin membrane, then the rule $(A'A, out; f, in) \in R_1$ brings into the system the trap-object f, which will pass forever through the skin membrane by the rule $(f, out; f, in) \in R_1$, hence the computation will never end. If A is not present, then A' waits one step; at the same time, the rule $(X', out; Ye, in) \in R_1$ brings the symbol e into the system (and completes the simulation of the rule $X \rightarrow Y$). At the next step, the symbols A' and e leave the system. In this way, the matrix is correctly simulated.

If, at any time, we cannot simulate a matrix as above, although a symbol $X \in N_1$ is present in the skin membrane, then the rule $(X, out; f, in) \in R_1$ will bring the trap-symbol f into the system and the computation will never finish.

In this way, any derivation in G can be correctly simulated, and, conversely, all correct (that is, not introducing the trap symbol f) computations in Π correspond to correct derivations in G. When the derivation is terminal, that is, the symbol Z is introduced, we pass to the second phase of the work of Π.

First, the symbol Z will bring into the system arbitrarily many copies of the symbols b and $e_i, 1 \leq i \leq k$. The work of Z ends by introducing the symbols c and d into the system (by the rule $(Z, out; cd, in) \in R_1$; note that Z is left in the environment). The symbols c and d enter immediately membrane 2, d in exchange with g, hence from now on the rule $(a, in)_g$ cannot be used any more.

In the presence of c, the rule $(a^{k+1}, out; b, in)_c$ can be used. Because of the maximal parallel manner of using the rules, the application of this rule means dividing the number of occurrences of a present in membrane 2 by $k+1$. The result is obtained as the number of occurrencs of the symbol b. At the same time, the remainder is obtained in the form of a symbol e_i. Specifically, if we had n

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

copies of a, such that $n = m(k+1)+i$, for some $m \geq 0$ and $1 \leq i \leq k$ (the remainder is never zero), then in membrane 2 we get m copies of b and one copy of e_i. The correctness of this operation is ensured by the fact that if any copy of a remains unused after the application of the rules $(a^{k+1}, out; b, in)_c$ (maximal parallelism) and $(ca^i, out; e_i, in)$ (the choice of the maximal i), then the rule $(da, in) \in R_3$ can be used, and afterthat the pair da will pass forever through membrane 4, by the rules $(da, in), (da, out) \in R_4$, hence the computation will never end. The same result is obtained if we do not have sufficient copies of b or e_i in the skin membrane: any remaining copy of a will be used by the rule $(da, in) \in R_3$ and the computation will never halt.

At the next step, in the presence of e_i, the traveler enters the corresponding membrane $4 + i$. Because the remainder i corresponds to the symbol a_i from the string of L which is "parsed" at the present step, we get the same symbol via the weak coding h. Simultaneously, by means of the rule $(b, out; a, in) \in R_2$, we change all symbols b from membrane 2 with symbols a from the skin membrane (note that we have enough copies of a in the skin membrane, hence the exchange of symbols b is complete, and that we can bring in exactly as many copies of a as many copies of b we have, because the rule $(a, in)_g$ is no longer applicable), and in this way the process can be iterated, the division of the current number (m in the above notation) by $k + 1$ can be repeated. In the moment of division, t exits membrane $4 + i$, hence it is available for the next step.

We continue in this way until exhausting the number n we have started with, in the sense that we reach a stage when we have at most k copies of a in membrane 2; this means that only a rule $(ca^i, out; e_i, in) \in R_2$ is used. The symbol c will return to membrane 2, but no further rule can involve it, the symbol e_i will enter membrane $4 + i$ together with t, the traveler exits again, and the computation stops. The trace of t, with all occurrences of label 2 ignored (removed by h) is exactly the string $x \in L$ whose value was initially codified by the number of a's produced by the derivation in G. In conclusion, $L(\Pi) = L$, and this concludes the proof. □

The previous proof can easily be modified in order to use forbidding contexts only. Specifically, we use the symbol g, placed initially in membrane 2, in a forbidding mode: we replace the permitting context rule $(a^{k+1}, out; b, in)_c \in R_2$ with the forbidding context rule $(a^{k+1}, out; b, in)_{\neg g}$, and we also replace the rule $(a, in)_g$ with $(a, in)_{\neg d}$. Thus, as long as g is present in membrane 2, the rule $(a^{k+1}, out; b, in)_{\neg g}$ cannot be used. After finishing the simulation of a derivation in G, the symbol Z introduces the symbols c and d into the system, and d will entail the removing of g from membrane 2 (and will also prevent the entrance of any further copy of a). From now on we work (in the forbidding mode) in the same way as we have proceeded in the previous proof (in the permitting mode). If Π' is the system obtained by modifying as above the system Π from the proof of Theorem 3.1.4, then we get $L(\Pi) = L(\Pi') = L$. Consequently, we have obtained the following result.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Theorem 3.1.5 $RE = TPP_*(fsym_2, fanti_*)$.

The weight of symport rules used in the previous proofs is minimal, but we use antiport rules of an arbitrary weight (actually, depending on the number of symbols which appear in the considered language); also, there are one symport and one antiport rule which are used conditionally.

As we have pointed out before, the number of symbols appearing in the strings of a language induces an infinite hierarchy with respect to the number of membranes used by a P system able to generate that language. That is why, for any family FL of languages, it is natural to consider the subfamily nFL, of all languages in FL over the alphabets with n symbols.

The previous results has been essentially improved in [31] (by simulating register machines):

Theorem 3.1.6 $nRE = nTPP_{n+1}(sym_0, anti_2) = nTPP_{n+1}(sym_3,\ anti_0) = nTPP_{n+2}(sym_2, anti_0)$, *for all* $n \geq 1$.

Thus, at the same time both the permitting/forbidding conditions have been removed and the weight of the antiport rules has been bounded by small values. Note that the last equality is obtained in terms of symport rules only, of a weight as encountered in biology: two symbols at a time pass through a membrane. At the first sight, this is a rather surprising result, but the explanation lies in the natural connection with register machines and the fact that P systems with symport/antiport rules have an in-built context-sensitivity.

3.1.3 Decreasing the Number of Membranes

By the definition, in order to obtain a language over an alphabet with m symbols, we have to use a system with at least m membranes, hence the hierarchy on the number of membranes is trivially infinite in the case of trace languages. This direct dependence of the number of membranes on the number of symbols raises the question whether it is possible to keep bounded the number of membranes even when generating languages over arbitrary alphabets, or at least to use a number of membranes smaller than the number of symbols.

We present here three possible ways to address this question.

The *first* proposal is to consider several travelers. For instance, let us suppose that we have $T = \{t_1, t_2, \dots, t_k\}$ a set of k travelers and that the membrane structure of our P system contains m membranes. In this case, the alphabet of trace symbols, contains $k \cdot m$ symbols $a_{i,j}$, where $1 \leq i \leq k$ and $1 \leq j \leq m$.

Let $\sigma = C_1C_2 \dots C_k,\ k \geq 1$, be a halting computation and let $C_i(T) = \{a_{i,j} \mid t_i$ is in membrane $j\}$. We consider $trace(T, \sigma) = \{w_1w_2 \dots w_k \mid w_i \in V^*, \Psi_V(w_i) = \Psi_V(C_i(T))\}$, i.e., we concatenate permutations of strings representing each $C_i(T)$. Then, the trace language defined by a P system with several travelers is $LT(\Pi) =$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$\{h(trace(T,\sigma)) \mid \sigma$ is a halting computation in $\Pi\}$. Thus, $LT(\Pi)$ is a language over an alphabet with $k \cdot m$ symbols, although we only use m membranes.

Let us consider an **example**. We take the P system

$$\Pi = (V, T, \mu, w_1, w_2, w_3, \{d\}, R_1, R_2, R_3),$$

with:

1. $V = \{d, t_1, t_2\}$ the set of all objects in the system; object d is present in arbitrarily many copies in the environment;

2. $T = \{t_1, t_2\}$ is the set of travelers;

3. $\mu = [\ [\ ]_2[\ ]_3\ ]_1$ is a membrane structure with 3 membranes;

4. $w_1 = t_1t_2$, $w_2 = w_3 = \emptyset$ are the sets of objects in the initial configuration;

5. R_i is the set of symport rules associated with the membrane i, as follows:

   - $R_1 = \{(t_1, out), (t_1d, in), (t_2, out), (t_2d, in)\}$,
   - $R_2 = \{(t_1d, in), (t_1, out), (t_2, in)\}$,
   - $R_3 = \{(t_2d, in), (t_2, out), (t_1, in)\}$.

The computation begins with travelers t_1 and t_2 in membrane labeled 1, and with the inner membranes (labeled 2, and 3) without any object. The initial configuration of the system is: $[\ t_1t_2[\ ]_2[\ ]_3]_1$. According to the rules mentioned above, we can distinguish many cases in the evolution of the system. We discuss here only four:

- t_1 enters membrane labeled 3 and in the same time t_2 enters membrane labeled 2. In this case the computation halts, because membranes 3 and 2 act as trap membranes for travelers t_1 and t_2, respectively. The result is:

$$trace_1(\{t_1, t_2\}, \sigma) = \{a_{1,3}a_{2,2},\ a_{2,2}a_{1,3}\}.$$

- t_2 enters membrane 2 and is trapped, while traveler t_1 makes several (let us say n) journeys to the environment, and then to membrane 2 (bringing in objects d) until it enters membrane labeled 3, and the computation halts. We remind that we work with multisets of objects. For that, it is obvious that the number of trips traveler t_1 is paying to the environment and back to membrane 1 is as least as high as the number of trips it is paying to membrane 2. Now, we obtain:

$$trace_2(\{t_1, t_2\}, \sigma) = a_{2,2}a_{1,1}^n a_{1,2}^m a_{1,3},\ \text{for some } m \geq n.$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

- t_1 enters membrane 3 and is trapped while traveler t_2 behaves as traveler t_1 in the case above. The result of the computation is:

$$trace_3(\{t_1, t_2\}, \sigma) = a_{1,3}a_{2,3}^s a_{2,1}^t a_{2,2}, \text{ for some } t \leq s.$$

- Both t_1 and t_2 make different trips to membranes 2 and 3, respectively, provided that they have copies of d to do so. An easy-to-follow trace of the travelers, when both of them are moving, is the case when they get out membrane 1 and enter back (with a d), in an arbitrary number of steps, and then they go directly to their trap membranes. Thus,

$$trace_4(\{t_1, t_2\}, \sigma) = a_{1,1}^p a_{2,1}^p a_{1,3}a_{2,2}.$$

  The other cases (when, for example both t_1 and t_2 go out and back membrane 1 for an arbitrary number of steps and then, at a point traveler t_1 enters and exits membrane 2 for an arbitrary number of steps, and then decides to go again out of membrane 1, and so on) are already very difficult to follow, and no precise relationship between the trips of the two travelers can be given.

The *second* idea we propose in order to diminish the number of membranes is to consider an inverse morphism (for a morphism $h : V^* \to U^*$, the inverse morphism $h^{-1} : U^* \to 2^{V^*}$ is defined by $h^{-1}(y) = \{x \in V^* \mid h(x) = y\}$, $y \in U^*$). The idea is explained on the following example: take two alphabets, $V = \{a_1, a_2, \ldots, a_n\}$ and $U = \{0, 1\}$ and the morphism h defined by $h(a_i) = 0^i1$, $1 \leq i \leq m$. It is obvious that this mapping is injective, hence $card(h^{-1}(y)) = 1$ for each $y \in h(V^*)$. Thus, for any language $L \subseteq V^*$ we have $L = h^{-1}(h(L))$. It it obvious (from the way we defined h) that choosing L from a family nFL we have $h(L) \in 2FL$.

Therefore, from the relation mentioned in the previous section we obtain the following result:

Proposition 3.1.1 *Every* $L \in nREG$ *can be written in the form* $L = h^{-1}(L')$, *for* $L' \in 2TPP_3(sym_0, anti_2) = 2TPP_3(sym_3, anti_0) = 2TPP_4(sym_2, anti_0)$.

The *third* proposal for obtaining more symbols in the trace languages than the number of membranes is to consider the possibility of changing the labels of membranes, as used in the area of P systems with active membranes (details on active membranes can be found, for example, in [4]). Changing the labels is not considered in standard symport/antiport rules, but we can introduce this feature in a simple way: symport rules of the form (x, in), (x, out) can be written as $x[\ ]_i \to [\,x]_i$, and $[\,x]_i \to [\ ]_i x$, respectively, while an antiport rule $(x, out; y, in)$ can be written as $y[\,x]_i \to [\,y]_i x$. Generalizing, we can consider that whenever an object enters or gets out of a membrane it can change its label. Thus, our rules will become: $x[\ ]_i \to [\,x]_j$, $[\,x]_i \to [\ ]_j x$, and $y[\,x]_i \to [\,y]_j x$, respectively.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

In this way, we can have as many different labels as we want – with an important aspect to take care: not to have conflicts among the used rules, i.e., not to apply at the same time two rules which intend to change the label of the membrane in different ways. There are several possibilities for avoiding such conflicts: using the rules sequentially (only one in each membrane), using in parallel only a set of rules which change the label in the same way, allowing to only certain rules to change the labels and to use rules from this set in a restrictive way (e.g., sequentially), etc.

3.2 Trace Languages Considering Sets of Objects

In this section, we consider P systems whose regions contain finite *sets* of objects, not *multisets* as in the classical variant of P systems (such systems were investigated in [2] as number generating devices); moreover, we assume that the environment is empty.

Such a system is a construction

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

where:

1. V is the alphabet of chemicals (objects);
2. $t \in V$ is the *traveler*. There is exactly one traveler t in the system, that is, $|w_1 \cdots w_m|_t = 1$;
3. μ is a membrane structure with m membranes (injectively labeled by positive integers $1, 2, \ldots, m$);
4. w_i are strings representing the sets of objects present in the regions of $\mu,\ 1 \leq i \leq m$;
5. R_i is the set of symport and antiport rules associated with the membrane i; they have the forms $(x, in), (x, out)$ and $(x, out; y, in)$, for $x, y \in V^*,\ 1 \leq i \leq m$.

The trace of the traveler across membranes is encoded as a string over the alphabet $\{a_1, a_2, \ldots, a_m\}$, by recording, in order, every membrane visited by t.

We consider two different cases of marking the trace. In the first one, we count only the event of the traveler entering a membrane, and in this case we denote by $LTP_m^{set}(sym_p, anti_q, in)$ the family of languages generated by P systems with traces and symport/antiport rules over finite sets of objects, using at most m membranes, symport rules of weight at most p and antiport rules of weight at most q. In the second case we take into account both the fact that the traveler enters a membrane and that it exits it (so we collect twice the label of the membrane t visits). We denote by $LTP_m^{set}(sym_p, anti_q, in/out)$ the family of languages generated in this case, with the usual meaning of the parameters m, p, q.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Note that we do not consider initially objects in the environment.

In what follows we investigate the place of these families with respect to Chomsky hierarchy.

Theorem 3.2.1 $LTP_*^{set}(sym_*, anti_*, in) = REG$.

Proof Given a P system $\Pi = (V, t, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$ we can construct a regular grammar $G = (C, T, C_0, R)$, where C is the set of all configurations which can be reached by Π starting from the initial configuration (this set is finite, because the system only handles a finite number of objects), C_0 is the initial configuration, and $T = \{a_1, a_2, \ldots, a_m\}$. The rules of the grammar are constructed as follows.

1. $C_i \rightarrow C_j$ if $C_i \rightarrow C_j$ is a transition and the traveler does not enter any membrane;

2. $C_i \rightarrow a_k C_j$ if $C_i \rightarrow C_j$ is a transition and the traveler enters membrane k $(1 \leq k \leq m)$;

3. $C_i \rightarrow \lambda$ if C_i is a halting configuration.

It is obvious that $trace(\Pi) = L(G)$ which implies $LTP_*^{set}(sym_*, anti_*, in) \subseteq REG$.

Conversely, given a regular grammar $G = (N, T, S, R)$ (N is the set of nonterminals, $T = \{a_1, a_2, \ldots, a_m\}$, $S \in N$, and R is the set of productions of the form $A \rightarrow a_i B$, and $A \rightarrow a_i$, with $A, B \in N, a_i \in T$), we can construct a P system as follows. The initial configuration (in the bracketed form) is $[\, cSt[\, dNN']_0[\, f_1]_1[\, f_2]_2 \ldots [\, f_m]_m]_s$, $\{s, 0, 1, 2, \ldots, m\}$ is the set of labels for membranes, s is the label of the skin membrane, f_i $(1 \leq i \leq m)$ is the symbol within membrane i, dNN' is the (strings which describes the) set of objects in membrane 0 (N is the nonterminal alphabet of grammar G, N' is the set of primed versions of elements of N, and $d \in V$; dNN' is a short-hand writing for $N \cup N' \cup \{d\}$), cSt is the set of objects in the skin membrane, where t is the traveler, S is the initial symbol of grammar G, and $c \in V$.

In order to simulate a rule $A \rightarrow a_i B \in R$ we will use the following rules:

step	R_0	R_i
1		$(f_i, out; cAt, in)$
2	$(d, out; f_i, in)$	(At, out)
3	$(f_i B', out; A, in)$	$(c, out; d, in)$
4		$(d, out; f_i, in)$
5	$(B, out; dB', in)$	

where R_0 and R_i are the sets of rules associated with membranes labeled 0 and i, respectively. For the other membranes no rules are specified.

The process of simulating a rule $A \rightarrow a_i B$ is detailed below.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

In the first step, the antiport rule $(f_i, out; cAt, in)$ makes the traveler t (altogether with objects c and A) enter membrane labeled i, thus introducing the symbol a_i in the trace. In the same time, symbol f_i, initially present in membrane i, is expelled within the skin membrane.

What is left to simulate is the process of transforming A to B and to bring the system back to its starting configuration, in order to be ready for a new simulation of a rule.

In the second step of computation, rules $(d, out; f_i, in)$, and (At, out) indicate that our traveler goes out membrane i (altogether with object A), while f_i enters membrane labeled s, with the help of object d (its counterpart in the antiport rule).

In the next step, the position of A and B' in the system is interchanged, in the same time with expelling from membrane labeled 0 the symbol f_i. The position of objects c, and d is also modified by the antiport rule that applies for the membrane labeled i. In this moment, our system has the following configuration:

$$[\, f_i cB't[\, N\{N' - B'\}]_0[\, f_1]_1[\, f_2]_2 \ldots [\, d]_i \ldots [\, f_m]_m]_s.$$

We have now succeeded in rewriting A to B' (which is a copy of B). The following thing to accomplish is to make the system gain its initial configuration, this time with B in the place of A so the derivation could continue.

In step 4, we use rule $(d, out; f_i, in)$ to move back f_i in its initial place. In the last step of the computation, rule $(B, out; dB', in)$ sends d and B' in membrane labeled 0 while B is expelled from membrane labeled 0 to the skin membrane, and the system can now continue to simulate the derivation process.

We can continue the above process to simulate nonterminal rules of G.

A rule $D \rightarrow a_i$ is simulated using the rule

$$(f_i, out; cDt, in) \in R_i.$$

The work of this rule is obvious.

The derivation grammar G ends by using such a rule, hence also our system will halt.

Clearly, $trace(\Pi) = L(G)$, hence we also have the inclusion $REG \subseteq LTP_*^{set}(sym_*, anti_*, in)$. □

If we consider the degree of the P system and the weight of the rules, we can write the previous result in the form

$$mREG = LTP_{m+2}^{set}(sym_2, anti_3, in).$$

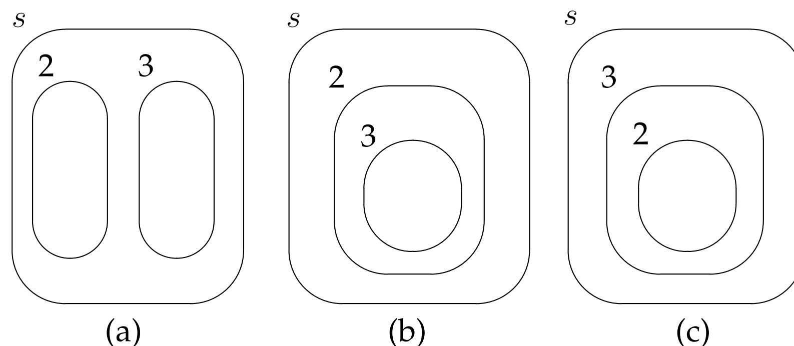
We consider now the in-out case.

Theorem 3.2.2 $LTP_*^{set}(sym_*, anti_*, in/out) \subsetneq REG.$

Proof We can use the same idea as in the previous theorem to prove the inclusion $LTP_*^{set}(sym_*, anti_*, in/out) \subseteq REG$. The properness of the inclusion is proved by the following counterexample.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Consider the language $L = \{a_2a_3, a_2a_2a_3\}$ and assume that there is a P system Π such that $trace(\Pi) = L$. Besides the skin membrane, this system must contain at least two membranes, with labels 2 and 3. These membranes can have one of the three relationships indicated in the next figure.



(a) (b) (c)

In case (a), in order to generate the string $a_2a_2a_3$ we must have the traveler outside membrane 2, and then a_2a_3 cannot be generated.

In case (b), in order to generate the string $a_2a_2a_3$ we must have the traveler in the region between membrane 2 and membrane 3, and then again a_2a_3 cannot be generated.

Similarly, in case (c), in order to generate the string $a_2a_2a_3$ we must have the traveler in the region between membrane 3 and membrane 2, and then a_2a_3 cannot be generated.

Thus, the equality $trace(\Pi) = L$ is not possible, this language is not in $LTP_*^{set}(sym_*, anti_*, in/out)$. □

3.3 Remarks and Further Research

Computing by communication (in the framework of membrane systems) proves to be surprisingly powerful – universality is reached by systems with a very small number of membranes, and this happens for the "standard" P systems with symport/antiport, when following the traces of certain objects through membranes, and when considering the analysing mode of using a system, as well. The biologically well motivated symport/antiport P systems deserve a special attention, both from a mathematical and a computational point of view.

In the case were P systems with sets of objects were considered (instead of multisets) we showed that the computational power of these systems with respect to Chomsky hierarchy does not go beyond the family of regular languages.

We also proposed, as a possible further research direction, three ideas to brake the infinite hierarchy provoked by the direct relationship between the number of membranes and the cardinality of the alphabet of languages.

ISBN:978-84-691-9877-3/DL:T-127-2009

Chapter 4

Inhibiting/De-Inhibiting P Systems

4.1 The Biological Source of the Concept

Before introducing the formal definition of the proposed mechanism let us briefly recall the reader the biological background of neural-cells. The core concept of this chapter is the process of inhibition/de-inhibition observed at the level of the axon, more precisely in the Ranvier nodes.

The neuron is not one homogeneous integrative unit but is (potentially) divided in many sub-integrative units, each one with the ability of mediating a local synaptic output to another cell or local electro-tonic output to another part of the same cell.

We have already mentioned in Section 2.1.2 that neurons are considered to have 3 main parts: a soma, the main part of the cell where the genetic material is present and life functions take place; a dendrite tree, the branches of the cell where impulses come in; an axon, the branch of the neuron over which the impulse (or signal) is propagated.

An axon can be provided with a structure composed by special sheaths. These sheaths are involved in molecular and structural modifications of axons needed to propagate impulse signals rapidly over long distance. There is a gap between neighboring myelinated regions that is know as the node of Ranvier, which contains a high density of voltage-gated Na^+ channels for impulse generation. When the transmitting impulses reach the node of Ranvier or junction nodes of dendrite and terminal trees, or the end bulbs of the trees, it causes the change in polarization of the membrane. The change in potential can be excitatory (moving the potential toward the threshold) or inhibitory (moving the potential away from the threshold).

The impulse transmission through a neuron follows this path: from dendrite to soma to axon to terminal tree to synapse. If different impulses reach at the same time a certain node, then, it might happen, that the combined effects of the excitation and inhibition may cancel each other out. Once the threshold of the membrane potential is reached, an impulse is propagated along the neuron or to the next neuron. More details about neural biology can be found in [81].

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

It is possible to formalize the mechanism described above by using rewriting rules equipped with ability to send excitation/inhibition signals.

An inhibited rule is formally written as $r : \neg(u \rightarrow v)$, and the meaning is that the rule cannot be applied, more precisely u cannot evolve to v. An evolution rule can de-inhibit an inhibited rule allowing it to be applied. To this aim, we also consider rules of the form $r_i : (u \rightarrow v)\{r_1, \cdots, r_k\}$, which say that u evolves into v and the rules $r_1, \cdots, r_k$ are inhibited or de-inhibited according to the previous state of each rule. Here rules $r_j, 1 \leq j \leq k$, are any kind of developmental rules. The evolution rule can be a deletion rule, and then we denote it by $r_i : (u \rightarrow \lambda)\{r_1, \cdots, r_k\}$.

In the following section, we introduce a class of P systems using inhibiting/de-inhibiting rules and explore the computational power of the class considering catalytic and non-cooperative inhibiting/de-inhibiting rules. In particular we prove that universality can be obtained (in generative and accepting cases) by using one catalyst. If we use only non-cooperative rules, then the systems can generate, at least, the Parikh sets of the languages generated by ET0L systems.

4.2 Inhibiting/De-Inhibiting Rules in P Systems

A P system *with inhibiting/de-inhibiting rules* (in short, an ID P system), of degree $m \geq 1$, is a construct

$$\Pi = (O, C, H, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

1. $m \geq 1$ is the degree of the system;
2. O is the alphabet of *objects*;
3. $C \subseteq O$ is the set of catalysts;
4. To each rule in $R = R_1 \cup R_2 \cup \cdots \cup R_m$, a unique label is associated. The set of all rule labels is $H = \{r_1, \cdots, r_k\}$. We denote $\neg H = \{\neg r_i \mid r_i \in H\}$. For a set Q of rules we indicate with $lab(Q)$ the set of labels of the rules that compose Q.
5. μ is a *membrane structure*, consisting of m membranes, labeled $1, 2, \ldots, m$;
6. $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
7. R_i is a finite set of *developmental rules*, associated with region i. The rules in R_i are of the forms:

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$$r_j : \neg(a \to w)S, \quad r_j : (a \to w)S,$$
$$r_j : \neg(ca \to cw)S, \quad r_j : (ca \to cw)S$$

where $r_j \in H, a \in O - C, w \in ((O - C) \times TAR)^*, c \in C, S \subseteq H$, for $TAR = \{here, out, in\}$; when $S = \emptyset$, we omit writing it, and we also omit the parentheses around the rule;

8. i_0 is the output region.

A configuration of an ID P system is described by using the m-tuple of multisets of objects present in the m regions of the system. To each region a finite number of objects is associated together with a finite number of rules. The m-tuple $(w_1, w_2, \cdots, w_m)$ is the initial configuration of the system. Some of the rules are initially inhibited (if the symbol $\neg$ is written immediately before the rule). A transition between two configurations is governed by the application in a non-deterministic and maximally parallel way of the rules that are not inhibited.

When a rule $r_j : (a \to w)S$ is applied, the object a is rewritten with the objects in w (as in standard P systems) and the rules from S are de-inhibited (if they were inhibited) or inhibited (if they were de-inhibited). In the same way is defined the application of catalytic rules.

For simplicity, each element in S is called a *switch*.

If simultaneously a rule r is subject of two or more switches, then the effect is that if a single switch, hence a change from inhibited to de-inhibited or conversely.

The system continues the application of the rules in maximally parallel way until there remain no applicable rules in any region of the system. Then the system halts (the computation is successful) and we consider the number of objects contained in the output region i_0 as the result of the computation.

We use the notation

$$Ps_{gen}IDP_m(\alpha), \alpha \in \{ncoo\} \cup \{cat_k \mid k \geq 0\},$$

to denote the family of sets of vectors of natural numbers generated by ID P systems with at most m membranes, evolution rules that can be non-cooperative ($ncoo$), or catalytic (cat_k), using at most k catalysts (as usual, $*$ indicates that the corresponding number is not bounded). When using systems in *the accepting* case, the subscript *gen* is replaced by *acc*. (A P system can also be used in the accepting mode: we introduce a number into the system in the form of a multiset and start the computation. If the system halts then the number introduced is accepted.)

Example We now illustrate the functioning of an ID P system with an example that shows how the simple mechanism of inhibiting/de-inhibiting rules can be very powerful.

We consider the ID P system *of degree 1*,

$$\Pi = (\{A, a\}, \emptyset, \{r_1, r_2, r_3\}, [\,]_1, A, R, 1),$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

where:

$$R = \{r_1 : A \to AA,\ r_2 : (A \to AA)\{r_1, r_2, r_3\},\ r_3 :\ \neg(A \to a)\}.$$

When the computation starts in the initial configuration with the object A in the skin region, the rules r_1 or r_2 can be applied but the rule r_3 cannot be applied since it is inhibited. We use the rule r_1 $m-1$ times and then we apply the rule r_2. In this way, we produce 2^m copies of object A. Simultaneously the rules r_1 and r_2 are inhibited (so they cannot be used any more), and the rule r_3 is de-inhibited. We have used context-free and inhibiting/de-inhibiting rules to obtain a^{2^m}, hence

$$\{(2^m) \mid m \geq 1\} \in Ps_{gen}IDP_1(ncoo)$$

and this set is not in $PsCF$.

4.2.1 Using One Catalyst: Two Universality Results

In this paragraph we prove the universality of P systems using catalytic rules (one catalyst) and one membrane.

We first consider the generative case.

Theorem 4.2.1 $Ps_{gen}IDP_1(cat_1) = PsRE.$

Proof Consider a matrix grammar $G = (N, T, S, M, F)$ with appearance checking, in the binary normal form, hence with $N = N_1 \cup N_2 \cup \{S, \#\}$ as introduced in Section 2.2.2. Assume that all matrices are labeled in an injective manner with $m_i, 1 \leq i \leq n$, and each terminal matrix $(X \to \lambda, A \to x)$ is replaced by $(X \to f, A \to x)$, where f is a new symbol. We define the set of rules $R_\# = \{X \to \# \mid X \in N_1 \cup N_2\}$.

We construct the P system of degree 1,

$$\begin{aligned}
\Pi &= (O, C, H, \mu, w_1, R_1, i_0), \text{where:} \\
O &= N_1 \cup T \cup N_2 \cup \{p, p', p'', \overline{p'}, \overline{p''}, c, d, d', d'', d''', f, \#\}, \\
C &= \{c\}, \\
H &= \{r_i \mid 1 \leq i \leq 13\} \cup \{r_{2,i}, r_{3,i}, r_{8,i}, r_{9,i}, r_{12,i} \mid 1 \leq i \leq n\} \\
&\cup\ \{r'_4, r'_5, r'_6\} \cup Lab_\Pi(R_\#), \\
\mu &= [\ ]_1, \\
w_1 &= cpX_{init}A_{init}, \\
i_0 &= 0,
\end{aligned}$$

and the set R_1 is constructed in the following way.

- The simulation of a matrix of type 2, $m_i : (X_i \to Y_i, A_i \to x_i)$, with $X_i \in N_1, Y_i \in N_1, A_i \in N_2, x_i \in (N_2 \cup T)^*, |x_i| \leq 2$, is done by using the following rules added to the set R_1:

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$$\begin{aligned}
r_1 &: (p \to p'p'')\{r_{2,i}, r_{3,i}\} \\
r_{2,i} &: \neg(X_i \to Y_i)\{r'_5, r_5, r_{2,i}\} \\
r_{3,i} &: \neg(cA_i \to cx_id')\{r'_6, r_6, r_{3,i}\} \\
r_2 &: d' \to d \\
r_3 &: (d \to p)\{r'_5, r'_6\} \\
r_4 &: p' \to \overline{p'} \\
r'_4 &: p'' \to \overline{p''} \\
r_5 &: \neg(\overline{p'} \to \lambda)\{r_5, r'_5\} \\
r'_5 &: \overline{p'} \to \# \\
r_6 &: \neg(\overline{p''} \to \lambda)\{r_6, r'_6\} \\
r'_6 &: \overline{p''} \to \#
\end{aligned}$$

The idea is the following one. The rule r_1 chooses the matrix i to apply and this is made by the simultaneous de-inhibition of rules $r_{2,i}$ and $r_{3,i}$ (all other rules of other matrices remains inhibited). The execution of both $r_{2,i}$ and $r_{3,i}$ inhibits the rules r'_5 and r'_6 that are used to trash the computation in the case the matrix chosen is not correctly applied. If the matrix is applied in the correct way (both rules are executed), then d is changed to p and the process can be iterated (the original configuration of inhibited/de-inhibited rules is re-established).

- The simulation of a matrix of type 3, $m_i : (X_i \to Y_i, A_i \to \#)$, with $X_i, Y_i \in N_1$ and $A_i \in N_2$, is done by using the following rules, added to the set of rules R_1:

$$\begin{aligned}
r_7 &: (p \to p')\{r_{8,i}, r_{9,i}\} \\
r_{8,i} &: \neg(X_i \to Y_id'')\{r'_5, r_5, r_{8,i}, r_{9,i}\} \\
r_8 &: p' \to \overline{p'} \\
r_{9,i} &: \neg(A_i \to \#) \\
r_9 &: d'' \to d''' \\
r_{10} &: d''' \to p
\end{aligned}$$

The idea of the simulation of this kind of matrix is the following one. The rule r_7 de-inhibits the rules corresponding to the matrix i to be simulated. If the rule $r_{8,i}$ is not applied, then the rule r'_5 is not inhibited and then the computation never halts.

- The simulation of a terminal matrix $m_i : (X_i \to f, A_i \to x_i)$, with $X_i \in N_1$, $A_i \in N_2$, and $x_i \in T^*, |x_i| \leq 2$, is done using the following rules (added to the set R_1):

$$\begin{aligned}
r_{11} &: (p \to \lambda)(Lab_\Pi(R_\#) \cup \{r_{12,i}\}) \\
r_{12,i} &: \neg(cA_i \to cx_i)\{r_{12,i}\}
\end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$$R_{11} = \{\neg(X \to \#) \mid X \in N_1 \cup N_2\}.$$

These rules are used to simulate a terminal matrix and then to halt the computation. In fact, p is deleted and the rule $r_{12,i}$ is executed; the rules in R_{11} are de-inhibited and they guarantee that, when the computation halts, only terminal objects are present.

R_1 also contains the following rules:

$$r_{12}: \ a \to (a, out)$$
$$r_{13}: \ \# \to \#.$$

The result of the computation is collected in the environment; from the above explanation follows that the set of vectors generated by Π is exactly the Parikh image of $L(G)$. □

We consider now the accepting case. We notice that, in the following proof, the switches are used only by non-cooperative rules.

Theorem 4.2.2 $Ps_{acc}IDP_1(cat_1) = PsRE.$

Proof In order to prove the inclusion "$\supseteq$" we will simulate an m–register machine $M = (m, I, l_0, l_h)$ (see Section 2.2.3). At each time during the computation, the current contents of register r is represented by the multiplicity of the object a_r.

Formally, we define the P system *of degree 1*,

$$\begin{aligned}
\Pi &= (O, C, H, [\,]_1, w_1, R, i_0), \text{where:}\\
O &= \{a_r, S_r, S'_r, S''_r, S'''_r \mid 1 \le r \le m\} \cup \{c, e, e', F, p, l_h\} \cup lab(M),\\
C &= \{c\},\\
H &= \{r_i \mid 1 \le i \le 15\},\\
w_1 &= cl_0,\\
i_0 &= 0,
\end{aligned}$$

and R is defined as follows:

- for each instruction $l_i : (\mathtt{ADD}(r), l_j, l_j) \in I$, we add to R the rule:

  $$r_1: \ l_i \to a_r l_j$$

- for each instruction $l_i : (\mathtt{SUB}(r), l_j, l_k) \in I$, we add to R the rules:

  $$\begin{aligned}
  &r_2: \ l_i \to eS_r\\
  &r_3: \ e \to e'\\
  &r_4: \ (S_r \to S'_r)\{r_5\}\\
  &r_5: \ \neg(ca_r \to cF)\\
  &r_6: \ S'_r \to S''_r
  \end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$$
\begin{aligned}
r_7 &: (F \to \lambda)\{r_5, r_{10}\} \\
r_8 &: S_r'' \to S_r''' \\
r_9 &: (S_r''' \to \lambda)\{r_{11}\} \\
r_{10} &: \neg(e' \to l_j p)\{r_{10}\} \\
r_{11} &: \neg(e' \to l_k)\{r_{11}, r_5\} \\
r_{12} &: (p \to \lambda)\{r_{11}\} \\
r_{13} &: l_h \to \lambda
\end{aligned}
$$

The system works as follows. We start by introducing in the system the objects $a_1^{k_1}, \dots, a_m^{k_m}$ (representing the input to be accepted). We also have inside the catalyst c and and the label l_0 of the first instruction of the register machine. The vector $(k_1, \cdots, k_m)$ represents the vector that has to be accepted by our P system.

The addition instructions are directly simulated by the corresponding rules r_1.

If a subtraction instruction $(l_i : \mathtt{SUB}(r), l_j, l_k) \in I$ has to be simulated then the rule $l_i \to eS_r$ is executed. In the following step the object S_r is used to de-inhibit rule r_5, and to produce object S_r' in rule r_4, while the object e evolves into e'.

In the next step, if the number of objects a_r in register r is greater than 0, then the execution of the de-inhibited rule $ca_r \to cF$ decreases the number of objects a_r by 1, and produces object F for the next step.

Meanwhile, the object S_r' evolves into S_r'' as shown in rule r_6.

The object F used in rule r_7 to inhibit the de-inhibited rule r_5 guarantees that r_5 is applied only once. The rule r_{10} is also de-inhibited by the execution of the rule r_7.

The execution of the r_{10} generates the label l_j of the next register machine instruction (it is executed only once because it is inhibited by itself)

In the other case (i.e., if there is no object a_r in region) the rule $a_r \to F$ cannot be executed, therefore the object F is not produced and rule r_7 cannot be applied.

On the other hand, object S_r'' evolves into S_r''' by the execution of rule r_8 and at the next step S_r''' de-inhibits rule r_{11}, so e' evolves to label l_k (notice that the object e' still appears, because rule r_{10} has not been applied in the previous steps and then rule r_{11} can generate label l_k of the next register machine instruction); rule r_{11} must also inhibit rule r_5 that has been previously de-inhibited by rule r_4.

In the next step the rule r_{12} is executed and then the rule r_{11} in again inhibited (in the case that the rule has not been used because the rule r_{10} has been applied). When the label of the next instruction has been generated then the entire process can be iterated. The simulation stops (and then the input is accepted) when label l_h (which stands for halt instruction) is generated. □

4.2.2 Using Non-Cooperative Rules and One Switch

We present now a result without proof. IDP systems, using non-cooperative rules, are able to generate (at least) the family of Parikh images of languages in $ET0L$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

The proof is made by simulating an ET0L system by using a very restricted IDP system with at most one switch for each non-cooperative evolution rule. More details are given in [13].

Theorem 4.2.3 $Ps_{gen}IDP_1(ncoo) \supseteq PsET0L$.

4.3 Inhibiting/De-Inhibiting P Systems with Active Membranes

As above, the basic idea of the inhibiting/de-inhibiting P systems with active membranes is that, when a rule (acting on the membranes or on the objects) is inhibited, then it cannot be applied until another rule de-inhibits it. The application of a rule can inhibit other rules (and in particular might inhibit itself).

A P system *with active membranes and inhibiting/de-inhibiting mechanism*, in short, an AID P system, without electrical charges and without using catalysts, is a construct

$$\Pi = (O, H, I, \mu, w_1, \dots, w_m, R),$$

where:

1. $m \geq 1$ is the initial degree of the system;
2. O is the alphabet of *objects*;
3. H is a finite set of *labels* for membranes;
4. I is a finite set of *labels* for rules;
5. μ is a *membrane structure*, consisting of m membranes, labeled with elements of H;
6. $w_1, \dots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
7. R is a finite set of *developmental rules* of various usual forms. Here are some examples:

    (in) $r : (a[\ ]_h \rightarrow [\ b]_h)S$, for $r \in I, h \in H, a, b \in O, S \subseteq I$ (communication rules; an object is introduced in the membrane during this process);

    (out) $r : ([\ a\ ]_h \rightarrow [\ ]_h b)S$, for $r \in I, h \in H, a, b \in O, S \subseteq I$ (communication rules; an object is sent out of the membrane during this process);

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

(rdn) $r : ([a[\ ]_{h1}]_{h2} \rightarrow [[u]_{h1}]_{h2}v)S$, for $r \in I, h_1, h_2 \in H, a \in O, u, v \in O^*, S \subseteq I$
(replicative-distribution rule; an object is replicated and distributed into a directly inner membrane and outside the directly surrounding membrane);

(d) $r : ([\ a\ ]_h \rightarrow b)S$, for $r \in I, h \in H, a, b \in O, S \subseteq I$
(dissolving rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(e) $r : ([\ a\ ]_h \rightarrow [\ b\ ]_h[\ c\ ]_h)S$, for $r \in I, h \in H, a, b, c \in O, S \subseteq I$
(division rules for elementary membranes; in reaction with an object, the membrane is divided into two membranes with the same label).

The set H of labels has been specified because it is also possible to allow the change of membrane labels. For instance, a communication rule can be of the more general form:

(in′) $r : (a[\ \ ]_{h_1} \rightarrow [\ b]_{h_2})S$, for $r \in I, h_1, h_2 \in H, a, b \in O, S \subseteq I$;

(out′) $r : ([\ a\ ]_{h_1} \rightarrow [\ \ ]_{h_2}b)S$, for $r \in I, h_1, h_2 \in H, a, b \in O, S \subseteq I$.

The rules in R are written as $r_j : (\neg r)S$ or as $r_j : (r)S$, where $r_j \in I$ and r is a rule of P systems with active membranes, and S is a subset of I. The AID P systems work like general P systems with active membranes. The only difference consists in the fact, that, at each step, only the non-inhibited rules can be used. When a rule $r_j : (r)S$ is applied, the rules whose labels are specified in S are inhibited (if they were de-inhibited) or de-inhibited (if they were inhibited). Now, starting from an initial configuration, the system evolves according to the rules and objects present in the membranes, in a non-deterministic maximally parallel manner. The system will make a successful computation if and only if it halts, meaning there is no applicable rule to the objects present in the halting configuration. The result of a successful computation is the number of objects present in the environment in a halting configuration of Π. If the computation never halts, then we will have no output.

We use the notation $Ps_{gen}AIDP_m(\alpha)$ to denote the family of sets of vectors of natural numbers generated by AID P systems with at most m membranes ($*$ if the number is not bounded) and developmental rules of types described by α.

A system Π as above can be also used in the accepting mode in the following way. Given a vector v of natural numbers, let x be a string over the alphabet O such that $\Psi_O(x) = v$; the occurrences of objects corresponding to the multiset described by the string x are inserted in a specified region and the vector v is accepted by the system Π if and only if the computation halts.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

We use the notation $Ps_{acc}AIDP_m(\alpha)$ to denote the family of sets of vectors of natural numbers accepted by AID P systems with at most m membranes ($*$ if the number is not bounded) and developmental rules of types α.

We first investigate the possibility of simulating Boolean circuits by means of AID P systems and then we consider their computing power.

4.3.1 Simulating Logical Gates

In this section we present AID P systems which simulate logical gates. We will consider that the input for a gate is given in the inner membrane, while the output will be computed and sent out to the outer region.

Simulation of AND Gate

Lemma 4.3.1 *Boolean AND gate can be simulated by AID P systems with rules of types* in′ *and* out′*, using two membranes and two objects (only the input), in at most four steps.*

Proof We construct the AID P system

$$\begin{aligned}
\Pi_{AND} &= (O, H, I, \mu, w_0, w_s, R), \text{with} \\
O &= \{\lambda, 0, 1\}, \\
\mu &= [\,[\ ]_0]_s, \\
w_0 &= w_s = \lambda, \\
H &= \{0, 1, s\}, \\
I &= \{r_i \mid 0 \le i \le 9\},
\end{aligned}$$

and the set R consisting of the following rules:

$$\begin{aligned}
&r_1 : [\,0]_0 \to [\ ]_1 0 \\
&r_2 : [\,0]_1 \to [\ ]_0 \lambda\{r_2, r_8\} \\
&r_3 : [\,1]_0 \to [\ ]_1 1\{r_2, r_4, r_5, r_6\} \\
&r_4 : [\,1]_1 \to [\ ]_0 \lambda\{r_2, r_8\} \\
&r_5 : \neg[\,0]_1 \to [\ ]_1 0\{r_5, r_7\} \\
&r_6 : \neg[\,1]_1 \to [\ ]_1 \lambda\{r_4, r_6, r_9\} \\
&r_7 : \neg 1[\ ]_1 \to [\,\lambda]_0\{r_4, r_6, r_7, r_8\} \\
&r_8 : \neg[\,0]_s \to [\ ]_s 0\{r_2, r_8\} \\
&r_9 : \neg[\,1]_s \to [\ ]_s 1\{r_2, r_5, r_9\}
\end{aligned}$$

We start by placing the input values x_1 and x_2 in the membrane with label 0. Depending on the value of the initial variables x_1 and x_2, the rules we apply for each of the four cases are:

for $x_1x_2 = 00$ – r_1, r_2, r_8,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

for $x_1x_2 \in \{01, 10\}$ – r_1, r_4, r_8, or r_3, r_5, r_7, r_8, and
for $x_1x_2 = 11$ – r_3, r_6, r_9.

More precisely, if two 1s are in membrane 0, in the first step, rule r_3 is applied, a 1 is expelled and membrane's label is changed to 1. In the same time according to the inhibition/de-inhibition concept, rules r_2 and r_4 are inhibited, while rules r_5 and r_6 are de-inhibited and ready to be used. In the second step we notice that only rule r_6 can be applied, thus, object 1, placed inside membrane labeled 1 is transformed, in its way out, into λ. One may notice that rule r_6, after is applied, restores the original status of rule r_4 and itself, and also de-inhibits rule r_9. In the third step, rule r_9 performs and the right answer 1 is sent out the skin membrane, while rules r_2, r_5, and r_9 come back to their original status.

In other words, after these three steps, our system sends out the skin membrane the right answer (given the input 11) and comes back to its initial configuration, thus being ready for a new input.

In the case when the input is 01 or 10, we can start by using r_1 or r_3. Let us examine the second case. Rule r_3 sends 1 out of membrane 0 and changes its label to 1. In the same time, rules r_2 and r_4 are inhibited, while rules r_5 and r_6 are de-inhibited. The only rule we can use in the second step is r_5 which expels 0 out of membrane 1, inhibits itself and de-inhibits rule r_7. In this moment we have the following configuration of our system $[\,[\ ]_1 01]_s$. We now apply rule r_7 which transforms object 1 to λ on its way in the inner membrane and changes its label from 1 to 0. Rule r_7 de-inhibits the inhibited rule r_4, inhibits r_6 and itself, and de-inhibits rule r_8. The fourth step is the one in which the right answer 0 is sent out skin membrane, while the system gets back to its initial configuration.

Thus, our system gives the right answer, in four steps, when we have input 01. In the other two cases (when we have the input 01 and we start by using first the rule r_1, or the input is 00) our system performs the rules mentioned above; the details are left to the reader. □

Simulation of OR Gate

Lemma 4.3.2 *Boolean OR gate can be simulated by AID P systems with rules of types* (b'_0) *and* (c'_0)*, using two membranes and two objects (only the input), in at most four steps.*

Proof We construct the AID P system

$$\begin{aligned}
\Pi_{OR} &= (O, H, I, \mu, w_0, w_s, R), \text{with} \\
O &= \{\lambda, 0, 1\}, \\
\mu &= [\,[\ ]_0]_s, \\
w_0 &= w_s = \lambda, \\
H &= \{0, 1, s\}, \\
I &= \{r_i \mid 0 \le i \le 9\},
\end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

and the following set R of rules:

$$r_1 : [\ 1]_0 \to [\ ]_1 1$$
$$r_2 : [\ 1]_1 \to [\ ]_0 \lambda\{r_2, r_8\}$$
$$r_3 : [\ 0]_0 \to [\ ]_1 0\{r_2, r_4, r_5, r_6\}$$
$$r_4 : [\ 0]_1 \to [\ ]_0 \lambda\{r_2, r_8\}$$
$$r_5 : \neg[\ 1]_1 \to [\ ]_1 1\{r_5, r_7\}$$
$$r_6 : \neg[\ 0]_1 \to [\ ]_1 \lambda\{r_4, r_6, r_9\}$$
$$r_7 : \neg 0[\ ]_1 \to [\ \lambda]_0\{r_4, r_6, r_7, r_8\}$$
$$r_8 : \neg[\ 1]_s \to [\ ]_s 1\{r_2, r_8\}$$
$$r_9 : \neg[\ 0]_s \to [\ ]_s 0\{r_2, r_5, r_9\}$$

As in the case of AND gate, we place initial values x_1 and x_2 in the membrane labeled 0 from the membrane structure. The succession of rules we apply for each case is (as expected due to the duality of the system) the following:

for $x_1x_2 = 00$ – r_3, r_6, r_9,

for $x_1x_2 \in \{01, 10\}$ – r_3, r_5, r_7, r_8, or r_1, r_4, r_8, and

for $x_1x_2 = 11$ – r_1, r_2, r_8.

We only give here the details of the case when x_1 and x_2 are both 1. Our system has the following initial configuration: $[\ [\ 11]_0]_s$. As mentioned above, the only rule we can apply is r_1, and our system evolves to the following configuration: $[\ [\ 1]_1 1]_s$. The next rule we can apply is r_2 through which the object in membrane 1 is transformed into λ and the membrane label changes to 0, the system evolving to $[\ [\ ]_0 1]_s$. After applying rule r_2, rule r_8 is de-inhibited while rule r_2 is inhibited. We now can apply r_8, which sends out the skin membrane the answer 1 and restores the initial configuration of the system inhibiting rule r_8 and de-inhibiting rule r_2.

We showed how our systems expels, in three steps, the right answer, given the input 11.

The details of the behavior of the system in the other three cases are left to the reader. □

Simulation of NOT Gate

Lemma 4.3.3 *Boolean unary NOT gate can be simulated by AID P systems with rules of type (b_0), in one step.*

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

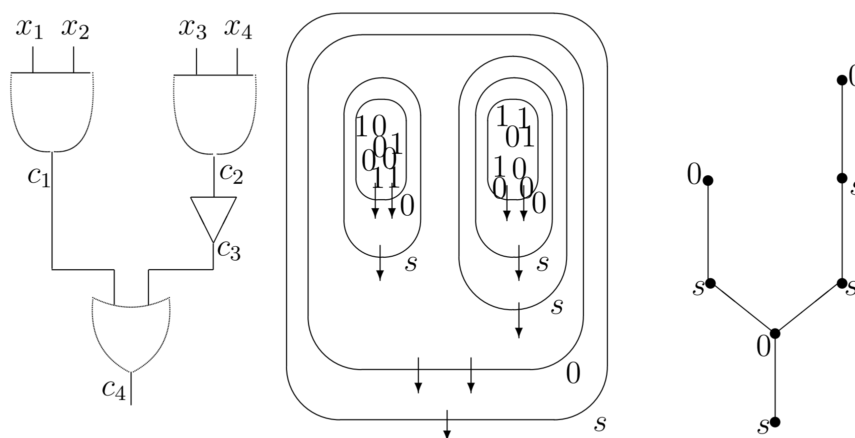


Figure 4.1: A Boolean circuit and its associated membrane structure of simulation by AID P systems.

Proof We construct the AID P system

$$
\begin{aligned}
\Pi_{NOT} &= (O, H, S, \mu, w_s, R), \text{with} \\
O &= \{0, 1\}, \\
\mu &= [\ ]_s, \\
w_s &= x_1 x_2, \\
H &= \{s\}, \\
S &= \{r_0, r_1\}, \\
R &= \{r_0 : [\,0]_s \to [\ ]_s 1, r_1 : [\,1]_s \to [\ ]_s 0\}.
\end{aligned}
$$

The correct computation of the NOT gate is obvious. □

4.3.2 Simulating Boolean Circuits

We give now an example of how to construct a global AID P system which simulates a Boolean circuit, designed for evaluating a Boolean function, using sub-AID P systems in it, namely including Π_{AND}, Π_{OR}, and Π_{NOT} constructed in the previous section.

An Example

We take into consideration the example used in [15], namely we consider the function $f : \{0,1\}^4 \to \{0,1\}$ given by the formula $f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee \neg(x_3 \wedge x_4)$. Our circuit and its assigned membrane structure is represented in Figure 4.1. As shown above, the circuit has a tree as its underlying graph, with the

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

leaves as input gates, and the root as output gate. We simulate this circuit with the P system $\Pi_C = (\Pi^{(1)}_{AND}, \Pi^{(2)}_{AND}, \Pi^{(3)}_{NOT}, \Pi^{(4)}_{OR})$ constructed from the distributed sub-AID P systems which work in parallel in the global P system, and we obtain an unique result in the following way:

1. for every gate of the circuit with inputs from input gates, we have an appropriate P system simulating it, with the innermost membrane containing the input values;

2. for every gate which has at least one input coming as an output of a previous gate, we construct an appropriate P system to simulate it by embedding in a membrane the "environments" of the P systems which compute the gates at the previous level.

For the particular formula $(x_1 \wedge x_2) \vee \neg(x_3 \wedge x_4)$ and the circuit depicted in Figure 4.1 we will have:

– $\Pi^{(1)}_{AND}$ computes the first AND$_1$ gate $(x_1 \wedge x_2)$ with inputs x_1 and x_2.

– $\Pi^{(2)}_{AND}$ computes the second AND$_2$ gate $(x_3 \wedge x_4)$ with inputs x_3 and x_4; these two P systems, $\Pi^{(1)}_{AND}$ and $\Pi^{(2)}_{AND}$, act in parallel.

– $\Pi^{(3)}_{NOT}$ computes NOT gate $\neg(x_3 \wedge x_4)$ with input $(x_3 \wedge x_4)$. While $\Pi^{(3)}_{NOT}$ is working, the output value of the first AND$_1$ gate performs the rules that can be applied (in $\Pi^{(4)}_{OR}$) and at a point waits for the second input (namely, the output of $\Pi^{(3)}_{NOT}$) to come.

– After the second input enters in the inner membrane of OR gate, P system $\Pi^{(4)}_{OR}$ will be able to complete its task. The result of the computation for OR gate (which is the result of the global P system), is sent into the environment of the whole system.

The idea we want to stress here is that, as noticed from the above explanations, our system has a self-embedded synchronization. By that, we mean that if either of the gates AND or OR receives only one (part of the) input from an upper level of the tree, the gate will wait for the other part of the input to come in order to expel the output. So, an extra synchronization system, as considered in [15], is not needed in AID P systems.

Based on the previous explanations the following result holds:

Theorem 4.3.1 *Every Boolean circuit α, whose underlying graph structure is a rooted tree, can be simulated by a P system, Π_α, in linear time. Π_α is constructed from AID P systems of type Π_{AND}, Π_{OR}, and Π_{NOT}, by reproducing in the architecture of the membrane structure, the structure of the tree associated to the circuit.*

Proof The statements follows from the previous considerations, with the observation that the simulation lasts at most four times the duration of evaluating the Boolean circuit. □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

CIRCUIT-SAT **Efficiency**

There is an interesting computational problem related to circuits, called CIRCUIT-SAT. Given a circuit C, is there a truth assignment T appropriate to C such that $T(C) = true$? It is easy to argue that CIRCUIT-SAT is computationally equivalent to SAT, and hence **NP**-complete.

We can now appeal to a well-known construction (see, e.g., [63]) to reduce a CIRCUIT-SAT instance to a CNF formula. Given a circuit C, we will construct a CNF formula φ_C such that there is an assignment to the inputs of C producing a 1 output iff the formula φ_C is satisfiable. The formula φ_C will have $n+|C|$ variables, where $|C|$ denotes the number of gates in C; if C acts on inputs $x_1, \cdots, x_n$ and contains gates $g_1, \cdots, g_{|C|}$, then φ_C will have variable set $\{x_1, \cdots, x_n, g_1, \cdots, g_{|C|}\}$. For each gate $g \in C$, we define a set of clauses as follows:

1. if $c = \text{AND}(a, b)$, then add $(\neg c \vee a), (\neg c \vee b), (c \vee \neg a \vee \neg b)$,
2. if $c = \text{OR}(a, b)$, then add $(c \vee \neg a), (c \vee \neg b), (\neg c \vee a \vee b)$,
3. if $c = \text{NOT}(a)$, then add $(c \vee a), (\neg c \vee \neg a)$.

The formula φ_C is simply the conjunction of all the clauses over all the gates of C.

We assume below that C consists of gates from a standard complete basis such as AND, OR, NOT. Our results can easily be generalized to allow other gates (e.g., with a larger fan-in); the final bounds are interesting as long as the number of clauses per gate (and the maximum fan-in in the circuit) is upper bounded by a constant. Recall that a circuit C is a directed acyclic graph (DAG). We define the underlying undirected graph as follows:

Definition 4.3.1 *Given a circuit C with inputs $X = \{x_1, \cdots, x_n\}$ and gates $S = \{g_1, \cdots, g_s\}$, let $G_C = (V, E)$ be the undirected and unweighted graph with $V = X \cup S$ and $E = \{\{x, y\} \mid x$ is an input to gate y or vice versa$\}$.*

Theorem 4.3.2 *For a circuit C with gates from* $\{\text{AND, OR, NOT}\}$*, the* CIRCUIT-SAT *instance for C can be solved by an AID P system.*

Proof Here is the sketch of the proof.

We know that propositional formula φ_C in CNF is simply the conjunction of all the clauses over all the gates of C. In our previous example, for the Boolean circuit considered in Section 4.3.2, φ_C is:

$$\begin{aligned}\varphi_C \quad = \quad & (\neg c_1 \vee x_1) \wedge (\neg c_1 \vee x_2) \wedge (c_1 \vee \neg x_1 \vee \neg x_2) \wedge (\neg c_2 \vee x_3) \wedge \\ & (\neg c_2 \vee x_4) \wedge (c_2 \vee \neg x_3 \vee \neg x_4) \wedge (c_2 \vee c_3) \wedge (\neg c_2 \vee \neg c_3) \wedge \\ & (\neg c_1 \vee c_4) \wedge (\neg c_3 \vee c_4) \wedge (\neg c_4 \vee c_1 \vee c_3).\end{aligned}$$

In P systems literature, there are already known algorithms which solve SAT (written as Boolean propositional formula in CNF) with P systems with active membranes. We also present a result of this type, in Theorem 4.3.5 below. Then our φ_C can be solved following the ideas from the proofs of these theorems. □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

4.3.3 Accepting and Generative Universality Results

P systems with active membranes and with particular combinations of several types of rules can reach universality without using polarizations, but only when additional features are used, for instance, the change of the labels of membranes.

We show here that P systems with active membranes using the inhibiting/de-inhibiting mechanism are universal even without polarizations or other additional features.

The first universality result follows from the one shown in Theorems 4.2.1 and 4.2.2. There has been shown that P systems with catalytic inhibiting/de-inhibiting evolution rules are universal in the accepting and generative sense. The same proofs works also for our model by using only one membrane and catalytic developmental rules (with one catalyst).

Therefore

Theorem 4.3.3 $Ps_{gen}AIDP_1(cat_1) = Ps_{acc}AIDP_1(cat_1) = PsRE.$

The next universality result is for the generative case and its proof is based on the simulation of matrix grammars with appearance checking.

Theorem 4.3.4 $Ps_{gen}AIDP_4(\mathsf{rdn}) = PsRE.$

Proof Let $G = (N, T, S, M, F)$ be a matrix grammar with appearance checking in the binary normal form as introduced in Section 2.2.2. Assume that all matrices of forms 2,3, and 4 are injectively labeled with elements of a set B (without loss of generality, suppose $0 \notin B$), $B = B_1 \cup B_2, B_1$ for the matrices of the form 2 and 4, and B_2 for the matrices of form 3. We construct the AID P system of degree 4

$$\begin{aligned}
\Pi &= (O, H, I, \mu, w_0, w_1, w_2, w_3, R),\\
O &= T \cup N_2 \cup \{\alpha_m \mid \alpha \in N_2 \cup T, m \in B\} \cup \{\lambda, \#\},\\
H &= \{0, 1, 2, 3\},\\
I &= \{r_{im} \mid 1 \leq i \leq 10, m \in B\} \cup \{r_{11}, r_{12} \cup \{r_a \mid a \in T\},\\
\mu &= [\,[\,[\,[\ ]_3]_2]_1]_0,\\
w_1 &= AX, w_0 = w_2 = w_3 = \lambda.
\end{aligned}$$

and the set R contains the following rules.

The simulation of a matrix $m : (X \to Y, A \to x)$, with $X \in N_1, Y \in N_1 \cup \{\lambda\}$, and $A \in N_2$, $|x| \leq 2$, $x = x'x'', x', x'' \in N_2 \cup T \cup \{\lambda\}$, is done using the following inhibiting/de-inhibiting rules added to the set R:

$$\begin{aligned}
r_{1m} &: [\, X[\ ]_2]_1 \to [\,[\, Y_m]_2]_1 \lambda\{r_{2m}\},\\
r_{2m} &: \neg[\, A[\ ]_2]_1 \to [\,[\, x'_m]_2]_1 x''_m\{r_{2m}, r_{3m}, r_{5m}\},\\
&\quad |x'_m| \leq 1, |x''_m| \leq 1,\\
r_{3m} &: \neg[\, Y_m[\ ]_3]_2 \to [\,[\,\lambda]_3]_2 Y\{r_{3m}, r_{4m}\},
\end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$$r_{4m} : \neg[\, x'_m[\ ]_3]_2 \rightarrow [\,[\,\lambda]_3]_2 x'\{r_{4m}\},$$
$$r_{5m} : \neg[\, x''_m[\ ]_1]_0 \rightarrow [\,[\,x'']_1\,]_0\{r_{5m}\}.$$

These rules simulate the matrices of the second and fourth type of M.

Initially, objects X and A are placed in the membrane with label 1. In the first step, by using rule r_{1m} object X is replicated to Y_m and λ, and object Y_m sent to the inner membrane.

In the same step rule r_{2m} is de-inhibited so it can be executed in the next step. By using rule r_{2m} the object A is replicated to object x'_m and object x''_m, $|x'_m| \leq 1$, $|x''_m| \leq 1$, and these objects are distributed into membrane 2 and membrane 0, respectively, while the rule inhibits itself; moreover, rules r_{3m} and r_{5m} are de-inhibited. In the next step, objects Y and x'' are introduced in the membrane with label 1 simultaneously, while rule r_{4m} is de-inhibited by rule r_{3m}. Then object x' enters into membrane 1 from membrane 2. In this way the first rule of the matrix has been simulated.

The simulation of a matrix $m : (X \rightarrow Y, A \rightarrow \#)$, with $X, Y \in N_1$, and $A \in N_2$, is done using the rules:

$$r_{6m} : [\,X[\ ]_2]_1 \rightarrow [\,[\,Y_m]_2]_1 \lambda\{r_{7m}, r_{8m}\},$$
$$r_{7m} : \neg[\,A[\ ]_2]_1 \rightarrow [\,[\,\lambda]_2]_1 \#,$$
$$r_{8m} : \neg[\,Y_m[\ ]_3]_2 \rightarrow [\,[\,Y_m]_3]_2 \lambda\{r_{8m}\},$$
$$r_{9m} : [\,Y_m[\ ]_4]_3 \rightarrow [\,[\,\lambda]_4]_3 Y_m\{r_{10m}\},$$
$$r_{10m} : \neg[\,Y_m[\ ]_3]_2 \rightarrow [\,[\,\lambda]_3]_2 Y\{r_{10m} r_{7m}\},$$
$$r_{11} : [\,\#[\ ]_1]_0 \rightarrow [\,[\,\#]_1]_0 \lambda,$$
$$r_{12} : [\,\#[\ ]_2]_1 \rightarrow [\,[\,\lambda]_2]_1 \#,$$
$$r_a : [\,a]_0 \rightarrow [\ ]_0 a, \text{ for all } a \in T.$$

When object X is rewritten to the object Y_m, it enters to the inner membrane with label 2 by using the rule r_{6m} and the rule r_{7m} is de-inhibited. If any object A appears in region 1, then the trap object $\#$ is produced and so the computation cannot halt. If the rule r_{7m} is not applied, after 4 steps, the object Y come in the region where object X is present and the rule r_{7m} will be inhibited again. Thus, also the matrix in appearance checking is simulated in the correct way and the process can be iterated.

The result of the computation is collected in the environment; from the above explanation it follows that the set of vectors generated by Π is exactly the Parikh image of $L(G)$. □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

4.3.4 An Efficiency Result for AID P Systems

The SAT problem (satisfiability of propositional formula in the conjunctive normal form) is probably the most known **NP**-complete problem. It asks whether or not for a given formula in the conjunctive normal form there is a truth-assignment of variables such that the formula assumes the value *true*. Let us consider a propositional formula in the conjunctive normal form:

$$\begin{aligned}
\beta &= C_1 \wedge \cdots \wedge C_m, \\
C_i &= y_{i,1} \vee \cdots \vee y_{i,l_i},\ 1 \leq i \leq m, \text{ where} \\
y_{i,k} &\in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m, 1 \leq k \leq l_i.
\end{aligned}$$

The instance β of SAT will be encoded in the rules of a P system by using the multisets v_j and v'_j of symbols, corresponding to the clauses satisfied by *true* and *false* assignment of x_j, respectively:

$$\begin{aligned}
v_j &= \{c_i \mid x_j \in \{y_{i,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq m\}, 1 \leq j \leq n, \\
v'_j &= \{c_i \mid \neg x_j \in \{y_{i,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq m\}, 1 \leq j \leq n.
\end{aligned}$$

We use here the most investigated way to obtain exponential work space–membrane division.

The next theorem shows how to solve the SAT problem by using the inhibiting/de-inhibiting mechanism without changing the labels of membranes, still in linear time.

Theorem 4.3.5 *AID P systems with rules of types* (e) *and* rdn*, constructed in a semi-uniform manner, can deterministically solve* SAT *in linear time with respect to the number of the variables and the number of clauses.*

Proof

We construct the AID P system

$$\begin{aligned}
\Pi &= (O, H, I, \mu, w_0, \cdots, w_7, R), \text{ with} \\
O &= \{a_i \mid 1 \leq i \leq n\} \cup \{c_i \mid 1 \leq i \leq m\} \\
&\cup\ \{d_i \mid 1 \leq i \leq m\} \cup \{e_i \mid 0 \leq i \leq 2n+m+3\} \\
&\cup\ \{\mathtt{yes}, \mathtt{no}\} \cup \{t_i, f_i \mid 1 \leq i \leq n\}, \\
H &= \{i \mid 0 \leq i \leq 6\}, \\
I &= \{g_i \mid 1 \leq i \leq 2n+m+2\} \cup \{h_i \mid 1 \leq i \leq 4\}, \\
\mu &= [\,[\,[\,[\,[\ ]_3]_2]_1[\,[\,[\ ]_6]_5]_4]_0, \\
w_1 &= a_1 \cdots a_n, w_4 = e_0, w_0 = w_2 = w_3 = w_5 = w_6 = \lambda,
\end{aligned}$$

with the following rules, divided according to their use.

The global control rules are the following ones:

$h_1 :\ [\, e_i[\ ]_5]_4 \rightarrow [\,[\, e_{i+1}]_5]_4\lambda,$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$h_2: \; [\, e_i[\,]_6]_5 \rightarrow [\,[\,\lambda]_6]_5 e_{i+1}, 0 \leq i \leq 2n+m.$

The control variables e_i counts the computing steps in the nested control membranes. As we shall see, after $2n+m+2$ derivation steps, the answer yes appears outside the skin membrane if the given satisfiability problem has a solution, whereas in the case that no solution exists, in one or two steps more, the answer no appears in the environment.

- Generation phase:

$g_1: \; [\, a_1]_1 \rightarrow [\, t_1]_1[\, f_1]_1\{g_2\},$

$g_i: \; \neg[\, a_i]_1 \rightarrow [\, t_i]_1[\, f_i]_1\{g_{i+1}\}, 2 \leq i \leq n.$

In the first step of the computation, using rule g_1 with object a_1, we produce the truth values $true$ and $false$ assigned to the variable x_1, placed in two new separate copies of membrane 1, and, at the same time, the rule g_2 is de-inhibited.

Therefore the previously de-inhibited rules $g_i, 2 \leq i \leq n$, will be executed in the next step.

In this way, in n steps we assign the truth values to all variables, hence we get all 2^n truth-assignments, placed in 2^n separate copies of membrane 1.

Objects t_i corresponds to the $true$ value of variable x_i, while object f_i corresponds to the $false$ value of variable x_i.

In the n-th step, after applying the rule g_n, the rule g_{n+1} can be executed.

$g_{n+i}: \; \neg[\, t_i[\;\,]_2]_1 \rightarrow [\,[\, v_i]_2]_1\lambda\{g_{n+i+1}\},$

$[\, f_i[\;\,]_2]_1 \rightarrow [\,[\, v_i']_2]_1\lambda\{g_{n+i+1}\}, 1 \leq i \leq n.$

By using the rules g_{n+i}, $1 \leq i \leq n$, every object t_i and f_i evolves into objects c_i (corresponding to clauses C_i, satisfied by the $true$ or $false$ values chosen for x_i) and object λ, respectively; and they are distributed into the inner and surrounding membranes.

- Checking phase:

$g_{2n+i}: \; [\, c_i[\;\,]_3]_2 \rightarrow [\,[\, c_i]_3]_2 d_i\{g_{2n+i+1}, g_{2n+i}\}, 1 \leq i \leq m.$

In the checking phase, by using the rules g_{2n+i}, object $c_i, 1 \leq i \leq n$, is placed in membranes labeled 2, and replicated into object c_i and counter object d_i. Object c_i is sent into the direct inner elementary membrane, and object d_i is sent out to the surrounding membrane. Meanwhile, the rule g_{2n+i} itself is inhibited and the rule g_{2n+i+1} is de-inhibited in order to check the next object.

If all objects $c_i, 1 \leq i \leq m$, are present in any membrane, then after m steps, object d_m is produced into the membranes with label 1.

- Output phase:

$g_{2n+m+1}: \; \neg[\, d_m[\;\,]_2]_1 \rightarrow [\,[\,\lambda]_2]_1 d_m,$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$g_{2n+m+2} : [\, d_m[\ ]_1]_0 \rightarrow [\,[\,\lambda]_1]_0\texttt{yes}\ \{g_{2n+m+2}h_4\}$.

If β has a solution, after $2n + m + 2$ steps, objects d_m appears in the skin membrane by using the rules g_{2n+m+1}, and again, one object d_m, non-deterministically chosen, will output object yes into environment, while the rule is inhibited to avoid further output. This is in the case that the formula is satisfiable and the computation stops (this step was the $(2n + m + 3)$th step of the entire computation).

$h_3 : [\, e_{2n+m+1(2)}[\ ]_5]_4 \rightarrow [\,[\,\lambda]_5]_4 e_{2n+m+2(3)}$,
$h_4 : [\, e_{2n+m+2(3)}[\ ]_4]_0 \rightarrow [\,[\,\lambda]_4]_0\texttt{no}$.

If β has no solution and if $2n + m + 1$ is an odd step, then after two more steps the counter object e_{2n+m+3} will output the answer no to the environment. Otherwise, after one more step object e_{2n+m+2} will execute this operation. □

4.4 Remarks and Further Research

In this chapter, we have considered a mechanism of chemical reactions in the cell biology, which is used to control the computation steps in P systems by inhibiting/de-inhibiting their general rules.

We investigated in Section 4.2 the computational power of this mechanism in both generative and accepting P systems. In particular we proved that universality can be obtained by using one catalyst and one membrane. If we use only non-cooperative rules and one membrane, then we can obtain at least the family of Parikh images of the languages generated by ET0L systems.

In Section 4.3 we have illustrated how the inhibit and de-inhibit mechanism can be introduced to P systems using active membranes. In particular, we have shown that **NP**-complete problems, in particular SAT, can be solved in linear time by using this model. Moreover, universality (in the generative and accepting case) can be obtained by using simple inhibiting/de-inhibiting active membrane's rules and without the change of labels.

Finally, in the previous section we have introduced a new way of simulating Boolean gates and circuits. This idea is very attractive because apart from using less biological resources (only two objects and two types of rules for the simulation of Boolean gates) than the previous simulations, we also proposed a system which has a self-embedded synchronization of the objects in the circuit without being necessary other tools to coordinate the computation.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Chapter 5

Spiking Neural P Systems

5.1 The Biological Source of the Concept

We recall here from [1], [54], [55] some notions about the neural cell, mainly focusing on the electric pulses a neuron is transmitting through its synapses; such a pulse is usually called *a spike*, or *action potential*.

In Figure 5.1 (a drawing by the Spanish Nobel prize winner Ramón y Cajal, one of the pioneers of neuroscience around 1900, reproduced from [33]), an example of a neural action potential is schematically given altogether with the main parts of a neuron – the cell itself (soma), the axon, the dendrites (a filamentous bush around the soma, where the synapses are established with the endbulbs of the axons of other neurons).

The neuronal signals consist of short electrical pulses (that can be observed as suggested in the figure by placing a fine electrode close to the soma or on the axon of a neuron) having an amplitude of about 100 mV and typically a duration of 1-2 ms. The form of the pulse does not change as the action potential propagates along the axon. A sequence of such impulses which occur at regular or irregular intervals is called a *spike train*. Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, *it is the number and the timing of spikes what matter*.

So, the size and the shape of a spike is independent of the input of the neuron, but the *time* when a neuron fires depends on its input.

Action potentials in a spike train are usually well separated. Even with very strong input, it is impossible to excite a second spike during or immediately after a first one. The minimal distance between two spikes defines the refractory period of the neuron.

The closeness of the axon of a neuron with the dendrites of another neuron is called a synapse. The most common type of synapse in the vertebrate brain is a chemical synapse. When an action potential arrives at a synapse, it triggers a complex chain of bio-chemical processing steps that lead to a release of neurotransmitter from the presynaptic neuron into the postsynaptic neuron. As soon as transmitter molecules have reached the postsynaptic side, they will be detected

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

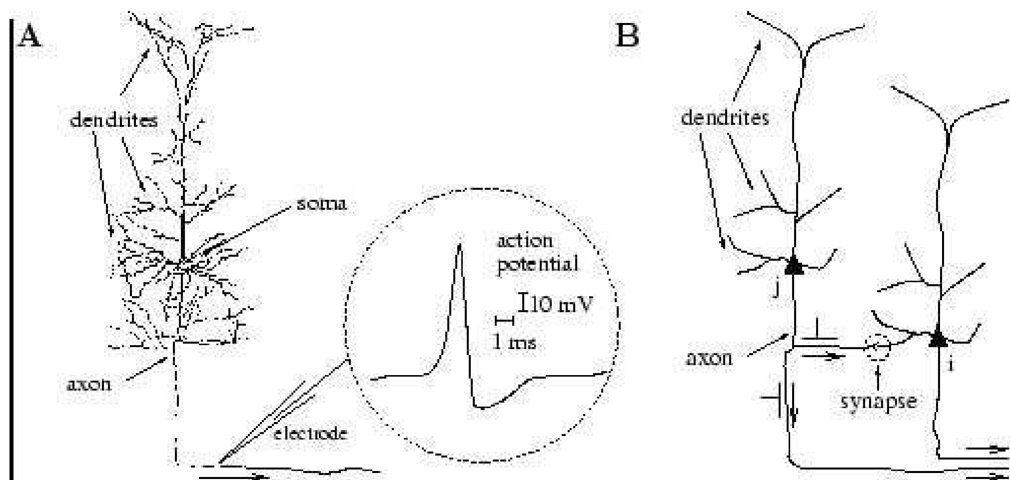


Figure 5.1: A neuron and its spiking

by specialized receptors in the postsynaptic cell membrane, and through specific channels, the ions from the extracellular fluid flow into the target cell. The ion influx, in turn, leads to a change of the membrane potential at the postsynaptic site so that, in the end, the chemical signal is translated into an electrical response. The voltage response of the postsynaptic neuron to a presynaptic action potential is called the postsynaptic potential.

In the following section, we will capture some of these ideas in the framework of neural-like P systems (defined in Section 2.4.4) as existing in the membrane computing literature, adapting the definition to the case of spiking.

5.2 The Initial Model

We pass from the definition of a neural-like P system to a model which makes explicit the restriction to work only with spikes. This means that we have to use only one impulse, hence symbol, identifying a generic electrical neural impulse, a "quanta" of electricity used in neural interaction. We will also remove the states of neurons; the firing will be controlled by the number of spikes present in a neuron and by the time elapsed since the previous spiking, in a very simple way; we will also relax the definition of successful computations, removing the halting condition (but adding a sort of simplicity condition: the output neuron can spike only twice).

Specifically, we consider a *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, in the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

   where:

   a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;

   b) R_i is a finite set of *rules* of the following two forms:

      (1) $E/a^r \rightarrow a; t$, where E is a regular expression over O, $r \geq 1$, and $t \geq 0$;

      (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \rightarrow a; t$ of type (1) from R_i;

3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* among neurons);

4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron*.

The rules of type (1) are *firing* (we also say *spiking*) *rules*: provided that the contents of the neuron (the number of spikes present in it) is described by the regular expression E (we return immediately to this aspect), r spikes are consumed (this corresponds to the result of the quotient of language $L(E)$ with respect to a^r, thus motivating the notation E/a^r from the firing rules), the neuron is fired, and it produces a spike which will be sent to other neurons after t time units (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized).

We have here two important actions which can take place in a step: getting fired and spiking.

A neuron gets fired when using a rule $E/a^r \rightarrow a; t$, and this is possible only if the neuron contains n spikes such that $a^n \in L(E)$ and $n \geq r$. This means that the regular expression E "covers" exactly the contents of the neuron. (For instance, a rule $a(aa)^+/a^3 \rightarrow a; 1$ can be applied to a neuron which contains seven copies of a, because $a^7 \in L(a(aa)^+) = \{a^{2n+1} \mid n \geq 1\}$, and then only four spikes remain in the neuron; now, the rule cannot be applied again – in general, the rule $a(aa)^+/a^3 \rightarrow a; 1$ cannot be applied if the neuron contains any even number of spikes.)

This is an important detail concerning the use of spiking rules, and we stress it again, especially because it contrasts the "standard" way of using rules in P systems, by rewriting parts of a multiset, in the maximally parallel mode both locally (in each compartment) and globally (at the level of the system). Here, at the level of each neuron we work in a sequential mode, with at most one rule used in each step, covering all spikes present in the neuron. Still, we have a maximal parallelism at the level of the system, in the sense that in each step *all* neurons which can evolve (use a rule) have to do it. We will come back to this aspect.

Now, about spiking. The use of a rule $E/a^r \rightarrow a; t$ in a step q means firing in step q and spiking in step $q + t$. That is, if $t = 0$, then the spike is produced

ISBN:978-84-691-9877-3/DL:T-127-2009

immediately, in the same step when the rule is used. If $t = 1$, then the spike will leave the neuron in the next step, and so on. In the interval between using the rule and releasing the spike, the neuron is assumed *closed* (in the refractory period), hence it cannot receive further spikes, and, of course, cannot fire again. This means that if $t \geq 1$ and another neuron emits a spike in any moment $q, q + 1, \ldots, q + t - 1$, then its spike will not pass to the neuron which has used the rule $E/a^r \to a; t$ in step q. In the moment when the spike is emitted, the neuron can receive new spikes (it is now free of internal electricity and can receive new electrical impulses). This means that if $t = 0$, then no restriction is imposed, the neuron can receive spikes in the same step when using the rule. Similarly, the neuron can receive spikes in moment t, in the case $t \geq 1$.

If a neuron σ_i spikes, its spike is replicated in such a way that one spike is sent to *all* neurons σ_j such that $(i, j) \in syn$, *and* σ_j is open at that moment. If a neuron σ_i fires and either it has no outgoing synapse, or all neurons σ_j such that $(i, j) \in syn$ are closed, then the spike of neuron σ_i is lost; the firing is allowed, it takes place, but it produces no spike.

The rules of type (2) are *forgetting rules*: s spikes are simply removed ("forgotten") when applying $a^s \to \lambda$. Like in the case of spiking rules, the left hand side of a forgetting rule must "cover" the contents of the neuron, that is, $a^s \to \lambda$ is applied only if the neuron contains exactly s spikes.

As defined above, the neurons can contain several rules, without restrictions about their left hand sides. More precisely, it is allowed to have two spiking rules $E_1/a^{r_1} \to a; t_1$, $E_2/a^{r_2} \to a; t_2$ with $L(E_1) \cap L(E_2) \neq \emptyset$ (but not forgetting rules $a^s \to \lambda$ with $a^s \in L(E_i), i = 1, 2$). This leads to a non-deterministic way of using the rules. If we use the spiking neural P systems in a generative mode (starting from the initial configuration, we evolve non-deterministically, and collect all results of all successful computations – we define immediately the used terms), then we cannot avoid the non-determinism (deterministic systems will compute only singleton sets). In the accepting mode of using our systems, as considered in Section 5.2.2, the non-determinism is no longer necessary.

However, we have imposed a *minimal determinism-like restriction* (we can also consider this as a *coherence* condition: either firing or forgetting, without being possible to chose among these two actions): no forgetting rule can be interchanged with a spiking rule. Thus, only in the case of spiking we allow branchings.

As suggested above, the rules are used in the non-deterministic manner, in a synchronous way at the level of the system: in each step, all neurons which can use a rule, of any type, spiking or forgetting, have to evolve, using a rule.

A spike emitted by a neuron i will pass immediately to all neurons j such that $(i, j) \in syn$ and are open, that is, the transmission of a spike takes no time, the spikes are available in the receiving neurons already in the next step.

The distribution of spikes in neurons and the states of neurons corresponding to the spiking intervals specified by the last rules used in each neuron (the open-close status and the time since the neurons were closed, depending on the rule

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

used) define the configuration of the system. The initial configuration is defined by the number of initial spikes, $n_1, \ldots, n_m$, with all neurons being open (no rule was used before). Formally, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron $i = 1, 2, \ldots, m$ contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps; with this notation, the initial configuration is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$.

Using the rules in this way, we pass from a configuration of the system to another configuration; such a step is called a *transition*. For two configuration C_1, C_2 of Π we denote by $C_1 \Longrightarrow C_2$ the fact that there is a direct transition from C_1 to C_2 in Π. The reflexive and transitive closure of the relation $\Longrightarrow$ is denoted by $\Longrightarrow^*$. A sequence of transitions, starting in the initial configuration, is called a *computation*.

With a computation we can associate several results. One possibility is the standard one in membrane computing: to consider only halting computations (reaching a configuration where no rule can be used) and to count the number of spikes present in the output neuron in the halting configuration, or sent to the environment by the output neuron during a halting computation. However, this is not in the style of spiking neurons, that is why we will consider here outputs related to the time when certain events take place. One idea is to take into account the moments when the output neuron, that with label i_0, spikes (*not* when it fires), and already we have two possibilities: if neuron i_0 spikes at times $t_1, t_2, \ldots$, then (i) either the set of numbers $t_1, t_2, \ldots$ can be considered as computed by Π, or (ii) the set of intervals between spikes, $t_s - t_{s-1}, s \geq 2$, can be the set computed by Π. Another possibility is to take a sequence of symbols/bits 0 and 1 as the result of a computation, with 0 associated with a moment when the output neuron does not spike and 1 associated with a spiking step. Finite and also infinite sequences of bits can be obtained in this way.

All these possibilities look very attractive, some of them will be considered below but their systematic study remains as a research topic. In what follows, we consider a further possibility, rather relaxed: we do not care whether or not the computation halts, but we only request that the output neuron spikes exactly twice during the computation. Then, the number of steps elapsed between the two spikes is the number computed by the system along that computation.

We denote by $N_2(\Pi)$ the set of numbers computed in this way by a system Π, with the subscript 2 reminding of the way the result of a computation is defined, and by $Spik_2P_m(rule_k, cons_p, forg_q)$ the family of all sets $N_2(\Pi)$ computed as above by spiking neural P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^r \rightarrow a; t$ having $r \leq p$, and all forgetting rules $a^s \rightarrow \lambda$ having $s \leq q$. When one of the parameters m, k, p, q is not bounded, then it is replaced with $*$.

5.2.1 Examples

We illustrate here the previous definitions with several examples, most of them also useful later.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

The first example concerns the system Π_1 given in a graphical form in Figure 5.2 – and in this way we also introduce a standard way to pictorially represent a configuration of an SN P system, in particular, the initial configuration. Specifically, each neuron is represented by a "membrane" (a circle or an oval), marked with a label and having inside both the current number of spikes (written explicitly, in the form a^n for n spikes present in a neuron) and the evolution rules; the synapses linking the neurons are represented by arrows; besides the fact that the output neuron will be identified by its label, i_0, it is also suggestive to draw a short arrow which exits from it, pointing to the environment.

Using this example, we also introduce the following **convention:** if a spiking rule is of the form $E/a^r \to a;t$, with $L(E) = \{a^r\}$ (this means that such a rule is applied when the neuron contains exactly r spikes – and all these spikes are consumed), then we will write this rule in the simpler form $a^r \to a;t$. Another simplification we adopt concerns the fact that we often refer to a neuron σ_i by its label, thus saying "neuron i" instead of "neuron σ_i".

In the system Π_1 we have three neurons, with labels 1, 2, 3; neuron 3 is the output one. In the initial configuration we have spikes in neurons 1 and 3, and these neurons fire already in the first step. The spike of neuron 3 exits the system, so the number of steps from now until the next spiking of neuron 3 is the number computed by the system. After firing, neuron 3 remains empty, so it cannot spike again before receiving a new spike. In turn, neuron 2 cannot fire until collecting exactly k spikes.

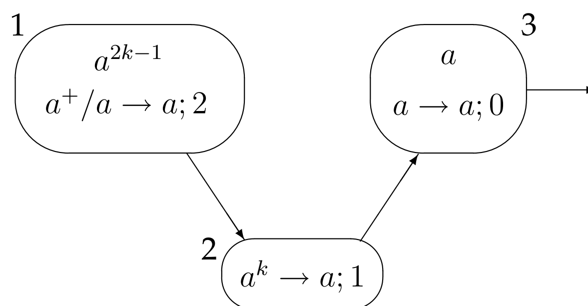


Figure 5.2: A simple example of an SN P system

After firing, neuron 1 will be closed/blocked for the next two steps; in the third step it will release its spike, sending it to neuron 2, and in step 3 will fire again. Thus, neuron 1 fires in every third step, consuming one of the spikes: any number $n \geq 1$ of spikes is "covered" by the regular expression a^+. In the step $3k$, neuron 2 will receive the kth spike emitted by neuron 1, hence in the next moment, $3k+1$, it will fire. The delay between firing and spiking is of one time unit for neuron 2, hence its spike will reach neuron 3 in step $3k+2$, meaning that neuron 3 spikes again in step $3k+3$. Therefore, the computed number is $(3k+3)-1 = 3k+2$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

The computation continues until consuming (and thus moving to neuron 2) all spikes from neuron 1, hence further $3(k-1)$ steps. However, neurons 2 and 3 will never fire again. If we have at least one more spike in neuron 1, then neuron 2 accumulates again k spikes, will fire for the second time, and the spike sent to neuron 3 will allow this neuron to spike for the third time. Such an event would make the computation unacceptable, we will get then no result.

The next example is presented in Figure 5.3 – we denote this SN P system by Π_2.

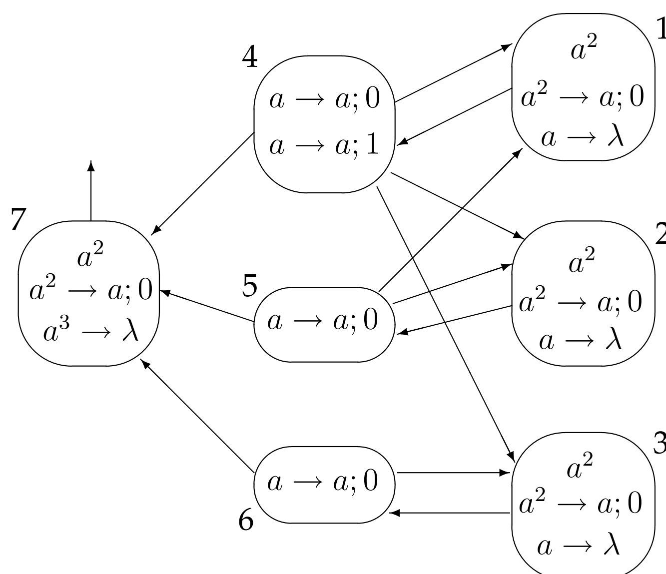


Figure 5.3: An SN P system generating all even natural numbers

In the beginning, only neurons 1, 2, 3, and 7 (which is the output neuron) contain spikes, hence they fire in the first step – and spike immediately. In particular, the output neuron spikes, hence we have to count the number of steps until the next spike, to define the result of the computation.

Note that in the first step we cannot use the forgetting rule $a \to \lambda$ in neurons 1, 2, 3, because we have more than one spike present in each neuron.

The spikes of neurons 1, 2, 3 will pass to neurons 4, 5, 6. In step 2, neurons 1, 2, 3 contain no spike inside, hence will not fire, but neurons 4, 5, 6 fire. Neurons 5, 6 have only one rule, but neuron 4 behaves non-deterministically, choosing between the rules $a \to a; 0$ and $a \to a; 1$. Assume that for $m \geq 0$ steps we use here the first rule. This means that three spikes are sent to neuron 7, while each of neurons 1, 2, 3 receives two spikes. In step 3, neurons 4, 5, 6 cannot fire, but all neurons 1, 2, 3 fire again. After receiving the three spikes, neuron 7 uses its forgetting rule and gets empty again. These steps can be repeated arbitrarily many times.

In order to fire again neuron 7, we have to use sometimes the rule $a \to a; 1$ of neuron 4. Assume that this happens in step t (it is easy to see that $t = 2m + 2$).

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

This means that at step t only neurons 5, 6 emit their spikes. Each of neurons 1, 2, 3 receives only one spike – and forgets it in the next step, $t+1$. Neuron 7 receives two spikes, and fires again, thus sending the second spike to the environment. This happens in moment $t+1 = 2m+2+1$, hence the computed number is $2m+2$ for some $m \geq 0$. The spike of neuron 4 (the one "prepared-but-not-yet-emitted" there by using the rule $a \rightarrow a;1$ in step t) will reach neurons 1, 2, 3, and 7 in step $t+1$, hence it can be used only in step $t+2$; in step $t+2$ neurons 1, 2, 3 forget their spikes and the computation halts. The spike from neuron 7 remains unused, there is no rule for it. Note that we cannot avoid using the forgetting rules $a \rightarrow \lambda$ from neurons 1, 2, 3: without such rules, the spikes of neurons 5, 6 from step t will wait unused in neurons 1, 2, 3 and, when the spike of neuron 4 will arrive, we will have two spikes, hence the rules $a^2 \rightarrow a;0$ from neurons 1, 2, 3 would be enabled again and the system will spike again.

Table 5.1 presents the computation of number 4 by the system Π_2; in each step, for each neuron we indicate in the first line the used rule and, in the second line, the spikes present in the neuron, with a dash used when no rule is applied and/or no spike is present; an exclamation mark indicates the spikes of the output neuron; the newly received spikes have a subscript which indicates their originating neuron.

Table 5.1: A computation in the system from Figure 5.3

Step / Neuron	0	1	2	3	4	5	6
1		$a^2 \rightarrow a;0$	—	$a^2 \rightarrow a;0$	—	$a \rightarrow \lambda$	$a \rightarrow \lambda$
	aa	—	a_4a_5	—	a_5	a_4	—
2		$a^2 \rightarrow a;0$	—	$a^2 \rightarrow a;0$	—	$a \rightarrow \lambda$	$a \rightarrow \lambda$
	aa	—	a_4a_5	—	a_5	a_4	—
3		$a^2 \rightarrow a;0$	—	$a^2 \rightarrow a;0$	—	$a \rightarrow \lambda$	$a \rightarrow \lambda$
	aa	—	a_4a_6	—	a_6	a_4	—
4		—	$a \rightarrow a;0$	—	$a \rightarrow a;1$	—	—
	—	a_1	—	a_1	—	—	—
5		—	$a \rightarrow a;0$	—	$a \rightarrow a;0$	—	—
	—	a_2	—	a_2	—	—	—
6		—	$a \rightarrow a;0$	—	$a \rightarrow a;0$	—	—
	—	a_3	—	a_3	—	—	—
7		$a^2 \rightarrow a;0\,!$	—	$a^3 \rightarrow \lambda$	—	$a^2 \rightarrow a;0\,!$	—
	aa	—	$a_4a_5a_6$	—	a_5a_6	a_4	a

We formally conclude that:

$$N_2(\Pi_2) = \{2n \mid n \geq 1\} \in Spik_2P_7(rule_2, cons_2, forg_3).$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

The next example is given both in a pictorial way, in Figure 5.4, and formally:

$$\begin{aligned}
\Pi_3 &= (\{a\}, \sigma_1, \sigma_2, \sigma_s, syn, 3), \text{ with} \\
\sigma_1 &= (2, \{a^2/a \to a; 0, \ \ a \to \lambda\}), \\
\sigma_2 &= (1, \{a \to a; 0, \ \ a \to a; 1\}), \\
\sigma_3 &= (3, \{a^3 \to a; 0, \ \ a \to a; 1, \ \ a^2 \to \lambda\}), \\
syn &= \{(1,2), \ (2,1), \ (1,3), \ (2,3)\}.
\end{aligned}$$

This system works as follows. All neurons can fire in the first step, with neuron 2 choosing non-deterministically between its two rules. Note that neuron 1 can fire only if it contains two spikes; one spike is consumed, the other remains available for the next step.

Both neurons 1 and 2 send a spike to the output neuron, 3; these two spikes are forgotten in the next step. Neurons 1 and 2 also exchange their spikes; thus, as long as neuron 2 uses the rule $a \to a; 0$, the first neuron receives one spike, thus completing the needed two spikes for firing again.

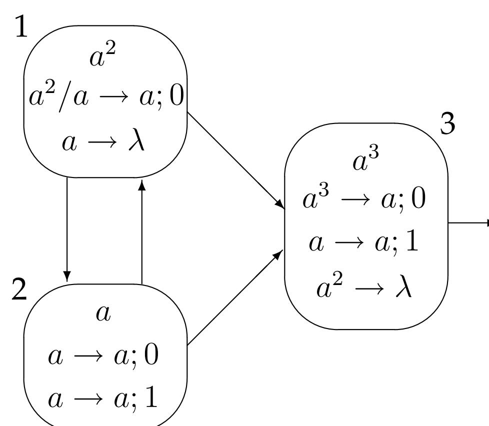


Figure 5.4: An SN P system generating all natural numbers greater than 1

However, at any moment, starting with the first step of the computation, neuron 2 can choose to use the rule $a \to a; 1$. On the one hand, this means that the spike of neuron 1 cannot enter neuron 2, it only goes to neuron 3; in this way, neuron 2 will never work again because it remains empty. On the other hand, in the next step neuron 1 has to use its forgetting rule $a \to \lambda$, while neuron 3 fires, using the rule $a \to a; 1$. Simultaneously, neuron 2 emits its spike, but it cannot enter neuron 3 (it is closed this moment); the spike enters neuron 1, but it is forgotten in the next step. In this way, no spike remains in the system. The computation ends with the expelling of the spike from neuron 3. Because of the waiting moment imposed by the rule $a \to a; 1$ from neuron 3, the two spikes of this neuron cannot be consecutive, but at least two steps must exist in between.

Thus, we conclude that (remember that number 0 is ignored)

$$N_2(\Pi_3) = \mathbf{N} - \{1\} \in Spik_2P_3(rule_3, cons_3, forg_2).$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

At the price of using one more neuron, we can compute all natural numbers. Such a system is given, without further details, in Figure 5.5 (it is denoted by Π_4 and its output neuron is 4).

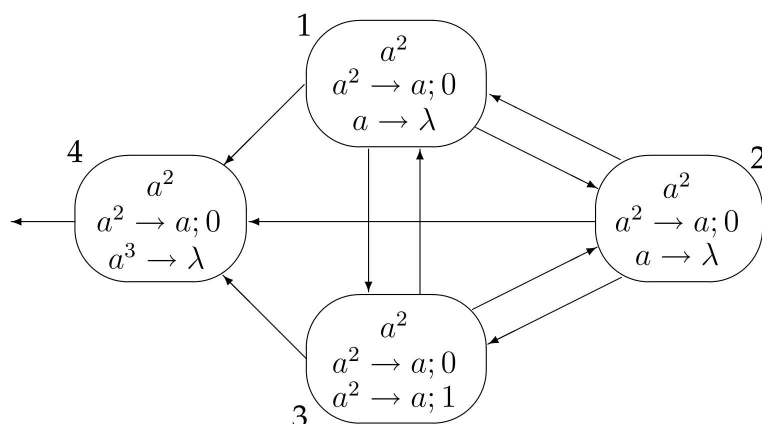


Figure 5.5: An SN P system generating all natural numbers

These last three examples will be useful in the next sections. Moreover, many of the proofs which follow are based on explicit constructions, hence further examples can be found in the sequel.

5.2.2 The Computational Power of SN P Systems

A Characterization of NFIN

In the continuation of the examples before, where an SN P system was presented with three neurons which computes an infinite set of numbers, let us examine the power of systems with one or two neurons only.

We start with the observation that by using any type of rules, spiking or forgetting rules, in a neuron σ_i we diminish the number of spikes from σ_i. In general, the number of spikes can increase in the whole system, because the spikes are replicated in the case of multiple synapses starting from a given neuron. However, if we have only one neuron in the system, then no replication is possible, hence each computation lasts at most as many steps as the number of spikes initially present in the neuron.

Consequently, SN P systems with only one neuron can only compute finite sets of numbers. Interesting enough, *any finite set can be computed by a one-neuron system*. Indeed, let us take a finite set of numbers, $F = \{n_1, n_2, \ldots, n_k\}$, $n_i \geq 1, 1 \leq i \leq k$, and construct the system Π_F with only one neuron, containing initially two spikes, and the following rules:

$$a^2/a \to a; 0,$$
$$a \to a; n_i - 1, \text{ for each } i = 1, 2, \ldots, k.$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

The system spikes in the first step by means of the rule $a^2/a \to a; 0$, then, because one spike remains inside, it fires again the next step, using any of the rules $a \to a; n_i$. This means that the next spike is sent out in step $n_i + 1$, hence $N_2(\Pi_F) = F$.

What about systems consisting of two neurons? One of them should be the output one, and the output neuron can spike only twice. This means that this neuron can increase the number of spikes in the system at most with two. The other neuron does not have a synapse to itself, hence, like in the case of single-neurons systems, it cannot use more rules than the number of spikes initially present in it, maybe plus two, those possibly received from the output neuron. This means that all computations are of a bounded length, hence the set of numbers computed by the system is again finite.

We synthesize these observations in the form of a theorem, mainly in view of the third example from the previous section: this is the best result of this type (in what concerns the number of neurons):

Theorem 5.2.1 $NFIN = Spik_2P_1(rule_*, cons_1, forg_0) = Spik_2P_1(rule_*, cons_*, forg_*) = Spik_2P_2(rule_*, cons_*, forg_*)$.

From these very small systems, let us now jump to the most general case, without any restriction on the number of neurons, or on other parameters.

Computational Completeness

The next inclusions follow directly from the definitions, with the inclusion in NRE provable in a straightforward manner (or, we can invoke for it the Turing-Church thesis):

Lemma 5.2.1 $Spik_2P_m(rule_k, cons_p, forg_q) \subseteq Spik_2P_{m'}(rule_{k'}, cons_{p'}, forg_{q'}) \subseteq Spik_2P_*(rule_*, cons_*, forg_*) \subseteq NRE$, *for all* $m' \geq m \geq 1$, $k' \geq k \geq 1$, $p' \geq p \geq 1$, $q' \geq q \geq 0$.

Surprisingly enough, taking into account the restrictive form of our systems, the spiking neural P systems prove to be computationally universal:

Theorem 5.2.2 $Spik_2P_*(rule_k, cons_p, forg_q) = NRE$ *for all* $k \geq 2, p \geq 3, q \geq 3$.

Proof In view of Lemma 5.2.1, we only have to prove the inclusion $NRE \subseteq Spik_2P_*(rule_2, cons_3, forg_3)$. To this aim, we use the characterization of NRE by means of register machines.

Let $M = (m, I, l_0, l_h)$ be a register machine, having the properties specified in Section 2.2.3: the result of a computation is the number from register 1 and this register is never decremented during the computation.

We construct a spiking neural P system Π as follows.

Instead of specifying all technical (and difficult to follow) details of the construction, we present the three main *types of modules* of the system Π, with the

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

neurons, their rules, and their synapses represented graphically. All neurons are initially empty, with the single exception of the neuron with label l_0 (the label of the initial instruction of M), which contains exactly two spikes (two copies of a).

What we want to do is to have Π constructed in such a way (1) to simulate the register machine M, and (2) to have its output neuron spiking only twice, at an interval of time which corresponds to a number computed by M.

In turn, simulating M means to simulate the ADD instructions and the SUB instructions. Thus, we will have a type of modules associated with ADD instructions, one associated with SUB instructions, and one dealing with the spiking of the output neuron (a FIN module). The modules of the three types are given in Figures 5.6, 5.7, 5.8, respectively. The neurons appearing in these figures have labels l_1, l_2, l_3, as in the instructions from I, labels $1, 2, \ldots, m$ associated with the registers of M, l'_1, l''_1, l''_1 associated with the label l_1 identifying ADD and SUB instructions from I, b_r (from "before r", because this neuron is used by all ADD instructions when sending a spike to neuron r), $f_1, f_2, f_3, f_4, f_5, f_6$, used by the FIN module, as well as out, labeling the output neuron.

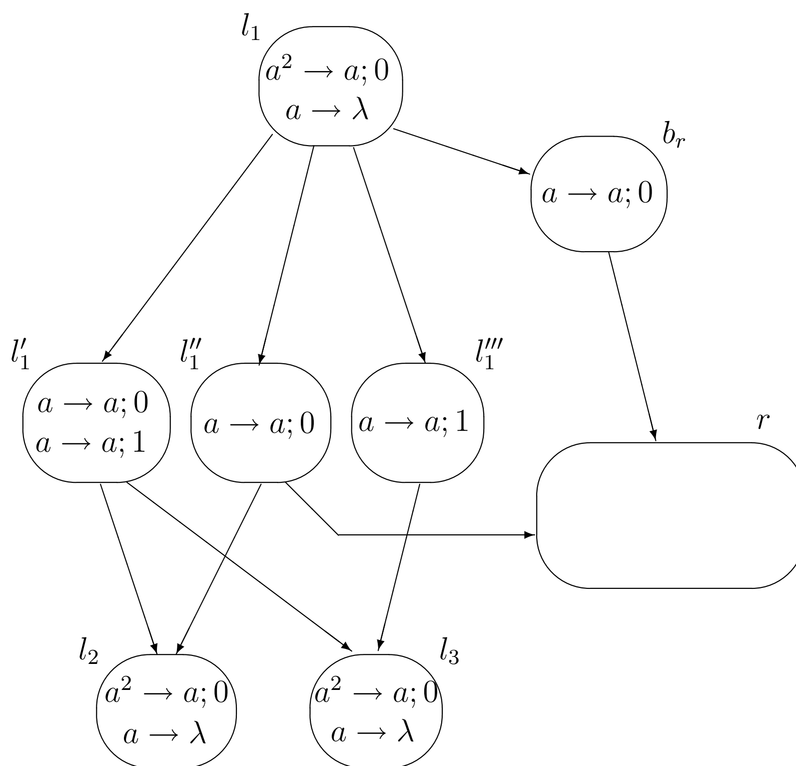


Figure 5.6: Module ADD (simulating $l_1 : (\texttt{ADD}(r), l_2, l_3)$)

Before describing the work of these modules, let us remember that the labels are uniquely associated with the instructions of M, hence each label precisely identifies one instruction, either an ADD or a SUB one, with the halting label having a special situation – it will be dealt with by the FIN module.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Simulating an ADD instruction $l_1 : (\texttt{ADD}(r), l_2, l_3)$ – module ADD (Figure 5.6).

The initial instruction, that labeled with l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an instruction $l_1 : (\texttt{ADD}(r), l_2, l_3)$, with two spikes present in neuron l_1 (like in the initial configuration) and no spike in any other neuron, except those neurons associated with the registers. Having two spikes inside, neuron l_1 gets fired. Its spike will simultaneously go to four neurons, l'_1, l''_1, l'''_1, and b_r.

In the next step, both neurons l''_1 and b_r will send a spike to neuron r, the one which corresponds to register r of M. In this way, the contents of neuron r increases by two. In each moment, if register r has value n, then neuron r will contain $2n$ spikes. In Figure 5.6, neuron r contains no rules, but, as we will see immediately, the neurons associated with registers have two rules each, used when simulating the SUB instructions, but both these rules need an odd number of spikes to be applied (this is true also for the module FIN, which only deals with the neuron associated with register 1). Therefore, during the simulation of an ADD instruction, neuron r just increases by 2 its contents, and never fires.

Now, the problem is to pass non-deterministically to one of the instructions with labels l_2 and l_3, that is, in our system we have to ensure the firing of neurons l_2 or l_3, non-deterministically choosing one of them. To this aim, we use the non-determinism of the rules in neuron l'_1. One of them will be used, thus consuming the unique spike existing here. If we use the rule $a \to a; 0$, then both neurons l_2, l_3 receive a spike from l'_1. For l_3 this is the unique spike it receives now (note that neuron l'''_1 fires now but its spike will leave one step later), hence in the next step the forgetting rule of neuron l_3 should be used. Instead, neuron l_2 receives two spikes, one from neuron l'_1 and one from neuron l''_1, hence in the next step it is fired.

However, if instead of rule $a \to a; 0$ we use the rule $a \to a; 1$ of neuron l'_1 (note that the only difference is the time of spiking), then it is l_2 which receives only one spike, and immediately it "forgets" it, while in the next step neuron l_3 receives two spikes, and fires.

Therefore, from firing neuron l_1, we pass to firing non-deterministically one of neurons l_2, l_3, while also increasing by 2 the number of spikes from neuron r.

Simulating a SUB instruction $l_1 : (\texttt{SUB}(r), l_2, l_3)$ – module SUB (Figure 5.7).

Let us examine now Figure 5.7, starting from the situation of having two spikes in neuron l_1 and no spike in other neurons, except neuron r, which holds an even number of spikes (half of this number is the value of the corresponding register r). The spike of neuron l_1 goes immediately to three neurons, l'_1, l''_1, and r. Neuron l'_1 will send in the next step a spike to neuron l_2, while neuron l''_1 will send a spike to neuron l_3 one step later.

A similar situation appears in neuron r: because it contains now an odd number of spikes, it gets fired, and it will either spike immediately, if the first rule is used ($a(aa)^+/a^3 \to a; 0$), or one step later, if the second rule is used ($a \to a; 1$). In turn, the first rule is used if and only if neuron r contains at least three spikes; one spike has just came from neuron l_1, hence at least two existed in the neuron,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

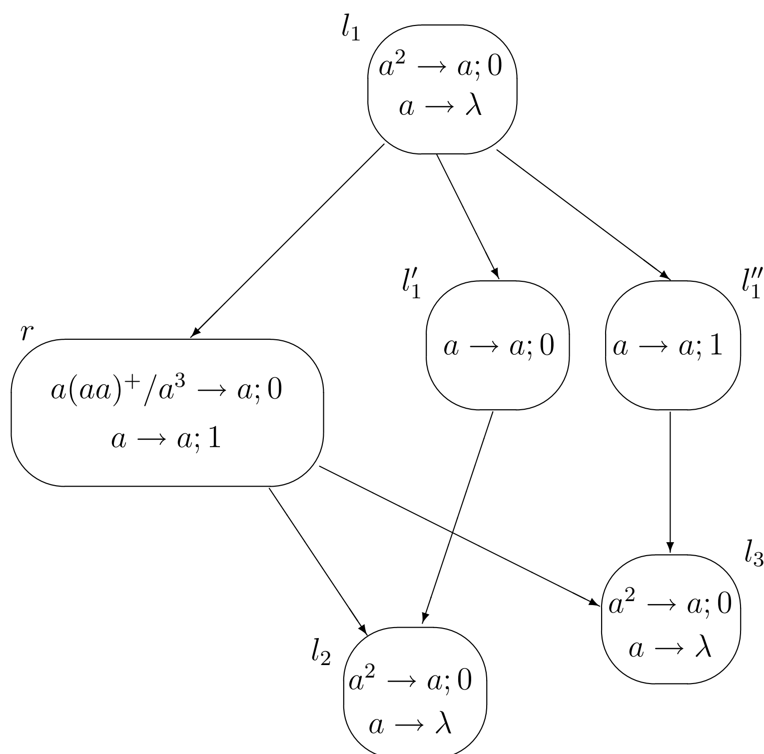


Figure 5.7: Module SUB (simulating $l_1 : (\texttt{SUB}(r), l_2, l_3)$)

which means that the value of register r was at least one. The two spikes which reach neuron l_2 make this neuron fire, as requested by simulating the SUB instruction. The spike from neuron l_3 will be removed by the local forgetting rule. If in neuron r there is only one spike (this corresponds to the case when register r is empty), then the second rule is used, hence the neuron spikes at the same time with l''_1, in the next step. This means that neuron l_2 receives only one spike (and removes it), while neuron l_3 receives two, and fires.

The simulation of the SUB instruction is correct, we started from l_1 and we ended in l_2 if the register was non-empty and decreased by one, and in l_3 if the register was empty.

Note that there is no interference between the neurons used in the ADD and the SUB instructions, other than correctly firing the neurons l_2, l_3 which may label instructions of the other kind. In particular, the ADD instructions do not use any rule for handling the spikes of neurons $1, 2, \ldots, m$. The only neurons used by several rules are those which correspond to registers and neuron b_r, used as a sort of interface before sending a spike to neuron r; then, each neuron r associated with a register which is subject of a SUB instruction sends a spike to several, possibly to all, neurons with labels $l \in lab(M)$ – but only one of these neurons also receives at the same time a spike from the corresponding neurons l'_1, l''_2, hence only the correct neuron fires, all others forget immediately the unique spike.

ISBN:978-84-691-9877-3/DL:T-127-2009

Ending a computation – module FIN (Figure 5.8).

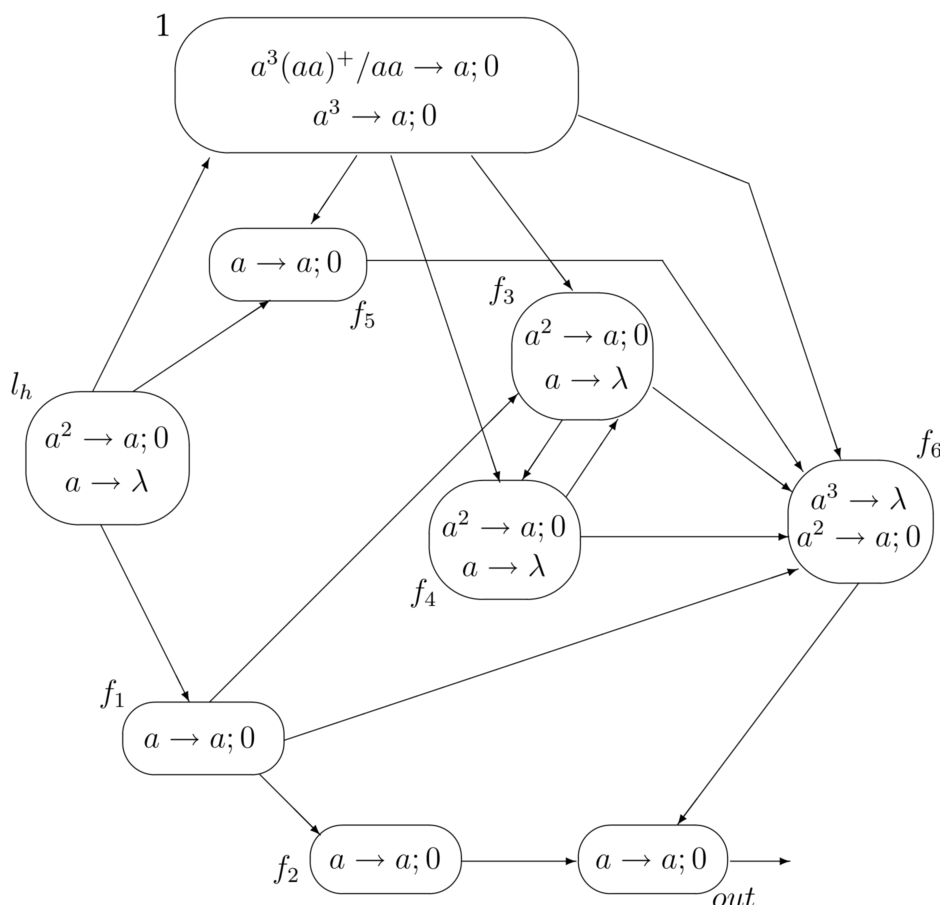


Figure 5.8: Module FIN (ending the computation)

Assume now that the computation in M halts, which means that the halting instruction is reached. For Π this means that the neuron l_h gets two spikes and fires. At that moment, neuron 1 contains $2n$ spikes, for n being the contents of register 1 of M. The spike of neuron l_h reaches immediately neurons 1, f_1, f_5. This means that neuron 1 contains now an odd number of spikes, and it can fire. It is important to remember that this neuron was never involved in a SUB instruction, hence it does not contain any rule as those from Figure 5.7.

Let t be the moment when neuron l_h fires.

At moment $t+1$, neurons f_1, f_5, and 1 fire and all of them spike immediately. The spiking of neuron 1 means subtracting one from the value of the associated register: by using the rule $a^3(aa)^+/a^2 \to a; 0$, two copies of a are consumed (hence the number of spikes remains odd, the rule can be used again next step). The spike of neuron 1 goes to four neurons, f_3, f_4, f_5, and f_6. This means that in step $t+1$ neuron f_6 receives three spikes (from neurons f_1, f_5, and 1), which are im-

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

mediately forgotten.

In step $t+2$, neurons f_2, f_3, f_5, and 1 fire and spike; again neuron f_6 receives three spikes, which are forgotten immediately. The spike from neuron f_2 reaches the output neuron, out, which in step $t+3$ will fire and spike. This is the first spike of the output neuron. The number of steps from this spike to the next one is the number computed by the system.

Let us see how the neurons f_3, f_4 interplay: in each step, each of them receives a spike from neuron 1. We start with neuron f_3 fired first; it sends a spike to neuron f_6 and one to the companion neuron f_4. This means that in the next step neuron f_4 is fired (he has received one spike from 1 and one from f_3), while neuron f_3, having only the spike from neuron 1, removes it. In the next step the roles of neurons f_3, f_4 are interchanged. This means that in each step one of them fires and sends a spike to the other (and one to neuron f_6), while the other only forgets one spike.

Then, let us observe that neuron 1 sends in each moment two spikes towards neuron f_6, one reaching immediately the target, the other one with one step delay, because it passes through neuron f_5. In turn, neuron f_5 never contains more than one spike, because the spike of neuron l_h reaches it in the first step and those from neuron 1 in the subsequent steps.

This means that the process of removing two spikes from neuron 1 continues, iteratively, without having neuron f_6 spiking, until using the rule $a^3 \to a; 0$. This is the last time when neuron 1 fires, hence this is step $t+n$ (for $2n$ being the initial contents of neuron 1). In the next step, $t+n+1$, one of neurons f_3, f_4 still fires, because it has two spikes, and the same with neuron f_5. No other neuron fires in this step. This means that neuron f_6 receives only two spikes, and this makes it spike in the next step, $t+n+2$. This is the only spiking in this step, all other neurons are empty.

In step $t+n+3$ also the output neuron spikes, and this ends the computation, the system contains no spike.

The interval between the two spikes of neuron out is $(t+n+3)-(t+3)=n$, exactly the value of register 1 of M in the moment when its computation halts. Consequently, $N_2(\Pi) = N(M)$ and this completes the proof. □

The previous construction contains only a few neurons which do not spike immediately, but their role in the correct functioning of the system is essential. Also the forgetting rules are crucial in this proof. Can these features be avoided (without decreasing the power of SN P system)? This problem was (affirmatively) solved in [35]. Another (standard) open problem is whether or not the parameters used in the theorem are optimal, or they can be improved. Of course, systems with only one rule in each neuron are deterministic, hence for such systems we have to look for other types of results of computations, e.g., the infinite sequence of bits marking the spiking of the output neuron. However, in what concerns the number of neurons behaving non-deterministically, a spectacular (especially by its metaphoric interpretation) result can be obtained:

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Corollary 5.2.1 *Each set from NRE can be computed by an SN P system Π having only one neuron with rules which can be used non-deterministically.*

Proof It is enough to observe that we have non-deterministic neurons only in the module ADD – this is the case with neuron l_1'. Let us "unify" all these neurons for all ADD instructions, in the form of a neuron l_{ndet} containing the rules

$$a \to a; 0, \quad a \to a; 1,$$

with synapses (l_1, l_{ndet}) for all instructions $l_1 : (\mathtt{ADD}(r), l_2, l_3)$ in the register machine we want to simulate, and (l_{ndet}, l) for all $l \in H$. When starting to simulate any instruction $l_1 : (\mathtt{ADD}(r), l_2, l_3)$, neuron l_{ndet} receives a spike, fires, and either spikes immediately, or in the next step. The spike is sent to all neurons σ_l, $l \in lab(M)$, but it meets another spike only in one neuron: l_2 if l_{ndet} spikes immediately and l_3 if it spikes in the next step. In all neurons different from these neurons, the spike is forgotten. The system obtained in this way is clearly equivalent with the one constructed in the proof of Theorem 5.2.2 and it contains only one non-deterministic neuron. □

This result can have a nice interpretation: it is sufficient for a "brain" (in the form of an SN P system) to have only one neuron which behaves non-deterministically in order to achieve "complete (Turing) creativity".

Now, in what concerns the number of spikes consumed for firing and the number of forgotten spikes, the problem remains whether or not the value 3 from the theorem can be decreased. Finally, it would be interesting to have universality results for systems with a bounded number of neurons, maybe paying in other parameters, such as the number of rules in neurons.

The previous construction can easily be modified – actually, simplified – in order to obtain an universality proof for the case where the result of a computation is defined as the number of spikes present in the output neuron in the last configuration of a halting computation (in this case we cannot avoid imposing the halting condition, because we have to make sure that no further spike will be added to the output): we just replace the module FIN with the module from Figure 5.9.

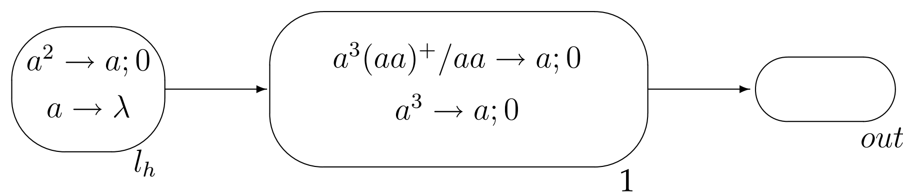


Figure 5.9: Module FIN in the case of counting the number of the spikes

When the halting instruction of the register machine M is reached, the neuron labeled l_h sends a spike to neuron 1, which has now $2n + 1$ spikes inside, and

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

can start to send spikes to neuron out. Exactly as in the proof of Theorem 5.2.2, the spiking of neuron 1 corresponds to decreasing by one the first register of M, hence the number of spikes accumulated in neuron out is equal to the value of register 1 in the end of a computation of M.

If we add the rule $a \rightarrow a; 0$ to neuron out, then in each step, after receiving a spike from neuron 1, neuron out spikes, hence in this way we obtain the result in the environment.

Spiking Neural P Systems Working in the Accepting Mode

An SN P system can be also used in the accepting mode. We consider here the following way of introducing the number to be accepted, again in the spirit of spiking neurons, with the time as the main data support: the special neuron i_0 is used now as an input neuron, which can receive spikes from the environment of the system (in the graphical representation an incoming arrow will indicate the input neuron); we assume that exactly two spikes are entering the system; the number n of steps elapsed between the two spikes is the one analyzed; if, after receiving the two spikes, the system halts (not necessarily in the moment of receiving the second spike), then the number n is accepted. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by a system Π, and by $Spik_{acc}P_m(rule_k, cons_p, forg_q)$ the family of sets of this form corresponding to the family $Spik_2P_m(rule_k, cons_p, forg_q)$.

In the accepting mode we can impose the restriction that in each neuron, in each time unit at most one rule can be applied, hence that the system behaves deterministically. When considering only deterministic SN P systems, the notation $Spik_{acc}P_m(rule_k, cons_p, forg_q)$ will get a letter "D" in front of it.

Of course, inclusions as those in Lemma 5.2.1 are valid both for deterministic and non-deterministic accepting SN P systems, and we also have the inclusion $DSpik_{acc}P_m(rule_k, cons_p, forg_q) \subseteq Spik_{acc}P_m(rule_k, cons_p, forg_q)$, for all m, k, p, q (numbers or $*$). A counterpart of Theorem 5.2.2 is also true:

Theorem 5.2.3 *$DSpik_{acc}P_*(rule_k, cons_p, forg_q) = NRE$ for all $k \geq 2, p \geq 3, q \geq 2$.*

Proof The proof is a direct consequence of the proof of Theorem 5.2.2. Namely, we start from a deterministic register machine M and we construct the SN P system Π as in the proof of Theorem 5.2.2 – with changes which will be immediately mentioned –, as well as a further module, called INPUT, which takes care of initializing the work of Π. This time Π has no object inside, and the same is true with the new module.

The module INPUT is indicated in Figure 5.10. Its functioning is rather clear. Because the system contains no spike inside, no rule can be applied. When a spike enters neuron in, this neuron sends a spike to all neighbors c_1, c_2, c_3, c_4 (they are new neurons). Neurons c_1, c_2 do nothing, they just wait for a second spike. Neurons c_3, c_4 spike immediately, sending together two spikes to neuron 1 (it corresponds to the first register of M), as well as to each other. This means that in each step each of c_3, c_4 fires, hence in each step the contents of neuron 1 increases

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

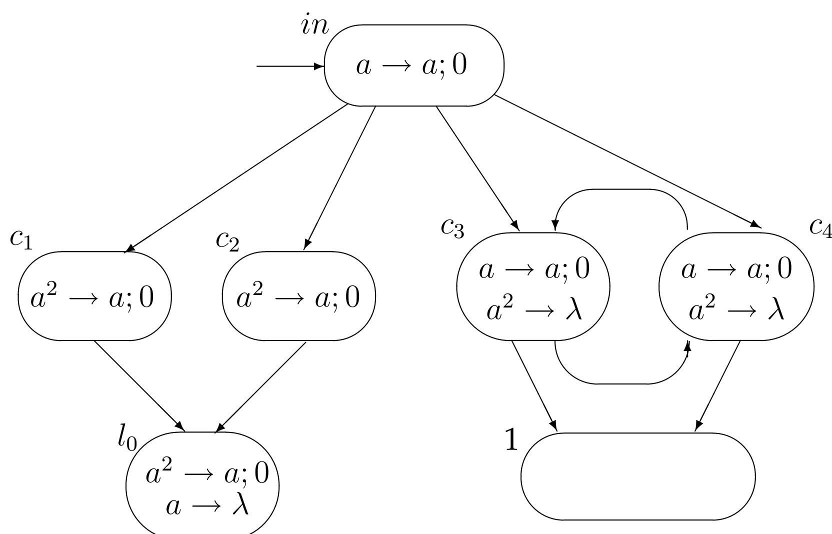


Figure 5.10: Module INPUT (initializing the computation)

again with two spikes. If at some moment, at n steps after the first spike coming to the system, a second spike enters neuron in coming from the environment, then this neuron spikes again. For neurons c_1, c_2 this entails the firing, and thus neuron l_0 gets the necessary spikes to fire, while for neurons c_3, c_4 this means the end of work, because the spikes are erased by the rules $a^2 \to \lambda$. Therefore, on the one hand, the contents of neuron 1 remains $2n$, on the other hand, neuron l_0 triggers the simulation of a computation in M, starting with the instruction labeled with l_0, in recognizing the number n.

Note that this is consistent with the way the system Π from the proof of Theorem 5.2.2 works: the neuron l_0 starts by having two spikes inside, and the contents of register 1 is double the number to analyze.

Now, we start using modules ADD and SUB associated with the register machine M, with modules ADD constructed for instructions of the form l_1 : $(\texttt{ADD}(r), l_2)$. This means that the module ADD is now much simpler than in Figure 5.6, namely, it looks like in Figure 5.11.

The functioning of this modified ADD module is obvious, hence we omit the details.

The modules SUB remain unchanged, while the module FIN is simply removed, with the neuron l_h remaining in the system, with no rule inside. Thus, the computation will stop if and only if the computation in M stops. The observation that the only forgetting rule $a^s \to \lambda$ with $s = 3$ was present in module FIN, which is no longer used, completes the proof. □

It is worth noting that both in this section and in the previous one, we use only firing rules with immediate spiking and with spiking at the next step. Furthermore, the regular expressions we have used (in the SUB and FIN modules)

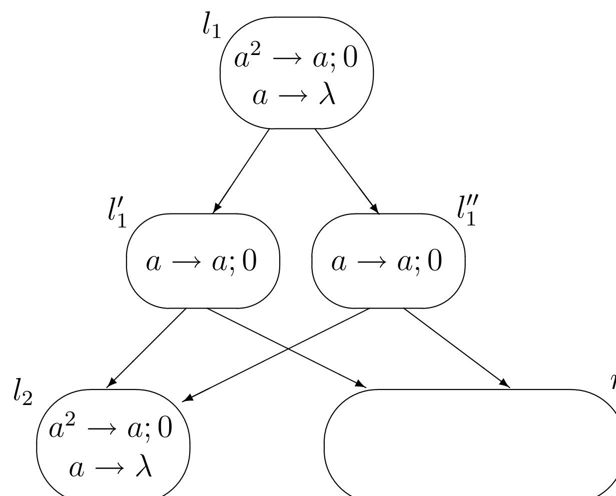


Figure 5.11: Module ADD in the deterministic case

are only meant to check the parity of the number of spikes present in the neuron. Both these parameters – the maximal delay in spiking and the complexity of the regular expressions used in rules – can be considered as complexity parameters of our systems (and the results above show that universality can be obtained even for rather simple systems from these points of view).

A Characterization of Semilinear Sets of Numbers

Let us now try to be more "realistic", not allowing the neurons to hold arbitrarily many spikes at the same time. This reminds of the sigmoidal function relating the input excitations to the neuron exciting: after a given threshold, the additional spikes do not matter, they are simply ignored.

Not very surprising (this reminds other characterizations of semilinear sets of numbers of or vectors of numbers in terms of P systems – see, e.g., [36]), but interesting by the proof, the SN P systems which have a bound on the number of spikes they hold during a computation generate exactly semilinear sets of natural numbers.

Let us denote by $Spik_2P_m(rule_k, cons_p, forg_q, bound_s)$ the family of sets of numbers $N_2(\Pi)$ computed by SN P systems Π with at most m neurons, using at most k rules in each neuron, consuming at most p and forgetting at most q spikes in each rule, and with at most s spikes present at any time in any neuron (if a computation reaches a configuration where a neuron accumulates more than s spikes, then it aborts, such a computation does not provide any result). As usual, we replace by $*$ the parameters which are not bounded ($bound_*$ will mean that we consider only SN P systems with a bound on the number of spikes present in any neuron, but this bound is not specified; when $bound_\alpha$ is simply missing from the notation, this will mean that the number of spikes in neurons can grow

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

arbitrarily, beyond any limit – like in the previous sections).

In this section we prove the following theorem:

Theorem 5.2.4 $SLIN_1 = Spik_2P_*(rule_k, cons_p, forg_q, bound_s)$, *for all* $k \geq 3, q \geq 3, p \geq 3$, *and* $s \geq 3$.

We start with the inclusion which is simpler to prove:

Lemma 5.2.2 $Spik_2P_*(rule_*, cons_*, forg_*, bound_*) \subseteq SLIN_1$.

Proof Take a system Π with a bound s on the number of spikes in each neuron. The number of neurons is given, their contents is bounded, the number of rules in neurons (hence the length of the refractory periods) is given, hence the number of configurations reached by Π is finite. Let $\mathcal{C}$ be their set, and let C_0 be the initial configuration of Π.

We construct the right-linear grammar $G = (N, \{a\}, (C_0, 0), P)$, where $N = \mathcal{C} \times \{0, 1, 2\}$, and P contains the following rules:

1. $(C, 0) \rightarrow (C', 0)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron does not spike;

2. $(C, 0) \rightarrow (C', 1)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron spikes;

3. $(C, 1) \rightarrow a(C', 1)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron does not spike;

4. $(C, 1) \rightarrow a(C', 2)$, for $C, C' \in \mathcal{C}$ such that there is a transition $C \Longrightarrow C'$ in Π during which the output neuron spikes;

5. $(C, 2) \rightarrow \lambda$, for $C \in \mathcal{C}$ if there is a halting computation $C \Longrightarrow^* C'$ in Π during which the output neuron never spikes, or there is an infinite computation starting in configuration C during which the output neuron of Π never spikes.

The way of controlling the derivation by the two components of the nonterminals in N ensures the fact that $N_2(\Pi)$ is the length set of the regular language $L(G)$, hence it is semilinear.

It is worth noting that the construction of grammar G is effective, because the conditions involved in defining the rules – including those from step 5 – can be decided algorithmically. □

The opposite inclusion is based on the observation that any semilinear set of numbers is the union of a finite set with a finite number of arithmetical progressions. Now, a finite set is the union of a finite number of singleton sets. Thus, it suffices to prove the closure under union and the fact that singleton sets and arithmetical progressions are in $Spik_2P_*(rule_3, cons_3, forg_3, bound_3)$, and we will do this in the following series of lemmas, whose conjunction – together with Lemma 5.2.2 – proves the theorem.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Lemma 5.2.3 *Each singleton* $\{n\}, n \geq 1$, *is in* $Spik_2P_1(rule_2, cons_1, forg_0, bound_2)$.

Proof Take the system with only one neuron, containing initially two spikes, and two rules:

$$a^2/a \to a; 0, \qquad a \to a; n-1.$$

The first spike exits in step 1, the second one in step $2+(n-1)$, hence the computed number is n. □

Lemma 5.2.4 *Each arithmetical progression* $\{ni \mid i \geq 1\}, n \geq 3$, *is in* $Spik_2P_{n+2}(rule_3, cons_3, forg_2, bound_3)$.

Proof For given n as in the statement of the lemma, we consider the SN P system in Figure 5.12.

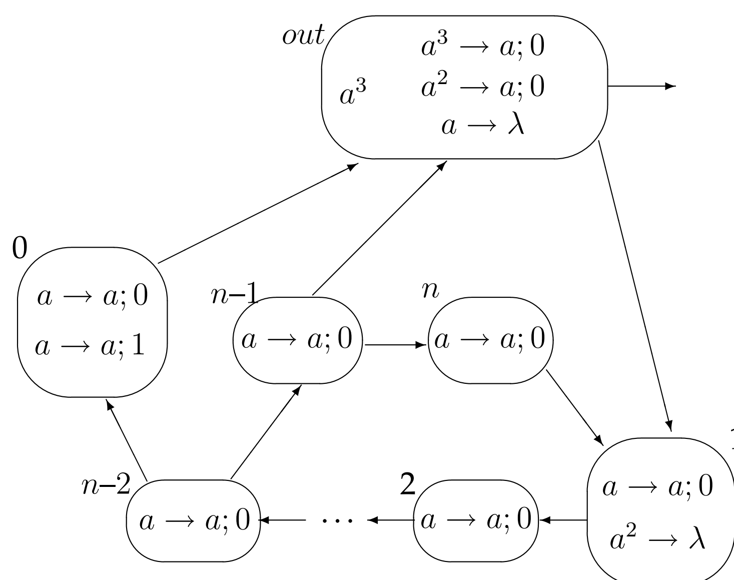


Figure 5.12: An SN P system generating an arithmetical progression

The neuron *out* spikes in step 1. The spike emitted by it goes along the path $1, 2, \ldots, n-2$ until getting doubled when passing from neuron $n-2$ to neurons $n-1$ and 0. Both these last neurons get fired. As long as neurons 0 and $n-1$ spike in different moments (because neuron 0 uses the rule $a \to a; 1$), no further spike exits the system (neuron *out* gets only one spike and forgets it immediately), and one passes along the cycle of neurons $1, 2, \ldots, n-1, n$ again and again. If neurons 0 and $n-1$ spike at the same time (neuron 0 uses the rule $a \to a; 0$), then the system spikes again – hence in a moment of the form $ni, i \geq 1$. The spike of neuron *out* arrives at the same time in neuron 1 with the spike of neuron n, and this halts the computation, because of the rule $a^2 \to \lambda$, which consumes

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

the spikes present in the system. Consequently, the system computes the "pure" arithmetical progression $\{ni \mid i \geq 1\}$. □

However, we have to compute not only "pure" progressions, but also of the form $\{r+ni \mid i \geq 1\}$, for $n \geq 1$ and $r \geq 1$. This is ensured by the following general result:

Lemma 5.2.5 *If* $Q \in Spik_2P_m(rule_k, cons_p, forg_q, bound_s)$ *and* $r \geq 1$*, then* $\{x + r \mid x \in Q\} \in Spik_2P_{m+1}(rule_k, cons_p, forg_q, bound_s)$*, for all* $m \geq 1, k \geq 2, p \geq 3, q \geq 0, s \geq 3$.

Proof Having a system Π, generating a given set Q, we add to this system a further neuron, labeled with out', with a synapse from the output neuron of Π to this new neuron; the new neuron has initially a^2 inside, and the rules

$$a^3 \to a; 0, \qquad a \to a; r.$$

Let Π' be the system obtained in this way, with the output neuron out'. If the system Π spikes at moments t_1 and t_2, then Π' spikes at the moments $t_1 + 1$ and $t_2 + 1 + r$, hence if Π computes the number $t_2 - t_1$, then Π' computes the number $(t_2 + 1 + r) - (t_1 + 1) = (t_2 - t_1) + r$. □

We still have to consider two particular progressions, the one with step 2 and the one with step 1. The former case was already covered by the second example from Section 5.2.1 (Figure 5.3). The latter is the set **N**, which, from the fourth example from Section 5.2.1 (Figure 5.5) is know to be in $Spik_2P_4(rule_2, cons_2, forg_3, bound_2)$.

For proving the closure under union we need an auxiliary result. For an SN P system Π, let us denote by $spin(\Pi)$ the maximal number of spikes present in a neuron in the initial configuration of Π.

Lemma 5.2.6 *For every SN P system* Π *(not necessarily with a bound on the number of spikes present in its neurons), there is an equivalent system* Π' *such that* $spin(\Pi') = 1$*. The system* Π' *has* $spin(\Pi) + 1$ *additional neurons, all of them containing only one rule, of the form* $a \to a; 0$.

Proof Take an arbitrary SN P system Π and construct a system Π' as follows. We consider further $spin(\Pi) + 1$ neurons, with labels $0, 1, 2, \ldots, spin(\Pi)$ (we assume that these labels are not used also in Π). Neuron 0 is the only one in the whole system which contains any spike, namely one (we remove all spikes from the neurons of system Π). The old neurons have the same rules as in Π, while each new neuron contains only the rule $a \to a; 0$. From neuron 0 start synapses to all neurons $1, 2, \ldots, spin(\Pi)$. From these neurons we establish as many synapses to the neurons of Π as many spikes they have in the initial configuration of Π (precisely, if a neuron l of Π has n_l spikes in the initial configuration of Π, then we

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

establish the synapses (i, l), for all $1 \leq i \leq n_l$). Thus, after two steps, all neurons of Π' which correspond to neurons of Π contain exactly as many spikes as they contained in the initial configuration of Π. No synapse goes back to the new neurons, hence from now on the system works exactly as Π, that is, $N_2(\Pi') = N_2(\Pi)$. □

We can now complete the proof of Theorem 5.2.4, by proving the "union lemma":

Lemma 5.2.7 *If* $Q_1, Q_2 \in Spik_2P_m(rule_k, cons_p, forg_q, bound_s)$, *for some* $m \geq 1, k \geq 2$, $p \geq 2$, $q \geq 1$, $s \geq 2$, *then* $Q_1 \cup Q_2 \in Spik_2P_{2m+6}(rule_k, cons_p, forg_q, bound_s)$.

Proof Take two SN P systems Π_1, Π_2 in the normal form given by Lemma 5.2.6. Let in_1, in_2 be the labels of neurons in Π_1, Π_2, respectively, containing initially one spike. We construct a system Π as suggested in Figure 5.13.

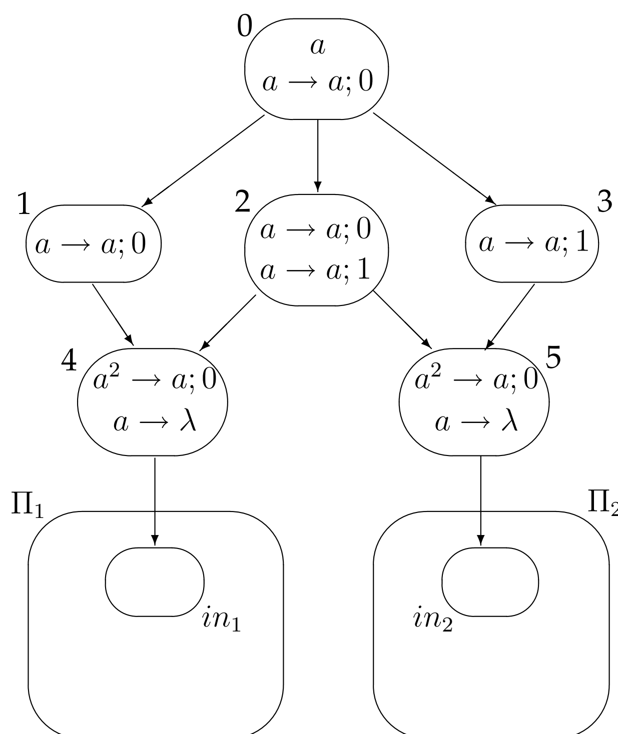


Figure 5.13: The idea of the union construction

In the system Π constructed in this way only neuron 0 contains a spike (those from neurons in_1, in_2 were removed. The non-deterministic functioning of neuron 2 will enable either the (sub)system Π_1, or the (sub)system Π_2, hence Π can compute whatever any of the two systems can compute. □

Note that also this lemma holds true for arbitrary systems, not only for the bounded ones.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Theorem 5.2.4 can be interpreted in the following way: even if we use time as a support for information, without using the classic workspace as a resource and without considering information also encoded in the number of objects used in the system, we cannot compute "too much" – precisely, we cannot go beyond the semilinear sets (the length sets of regular languages).

From Theorem 5.2.4, all closure properties of the family $SLIN_1$ are transferred to the family $Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$; for instance, we get the closure under sum, intersection, complement, and the non-closure under product. Still more precise results also follow from Lemmas 5.2.7 and 5.2.5, concerning the closure under union and the sum with a given number. A similar situation is met with respect to the non-closure under product:

Corollary 5.2.2 *There are sets* Q_1, Q_2 *in the family* $Spik_2P_3(rule_3, cons_3, forg_2, bound_3)$ *such that* $Q_1Q_2 = \{nm \mid n \in Q_1, m \in Q_2\}$ *is not in* $Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$.

Proof We take $Q_1 = Q_2 = \{2, 3, \dots\}$. From the third example in Section 5.2.1 we know that $Q_1 \in Spik_2P_3(rule_3, cons_3, forg_2, bound_3)$. Because Q_1Q_2 is the set of all composite numbers, and this set is not semilinear, from Theorem 5.2.4 it follows that $Q_1Q_2 \notin Spik_2P_*(rule_*, cons_*, forg_*, bound_*)$. □

We end this section with the following **general remark**: all SN P systems considered in the constructions from the proofs in this section and from Section 5.2.2 always halt (maybe a few steps) after the second spiking. Thus, all results above are valid also in the restrictive case of considering successful only halting computations (which spike exactly twice).

5.2.3 Remarks and Further Research

Starting from the definition of neural-like P systems and following the idea of spiking neurons from neurobiology, we have proposed a class of spiking neural P systems for which we have proved the universality in the general case and a characterization of semilinear sets of numbers in the case when the number of spikes is bounded.

These results are the basic ones in this area, and many research directions remain to be explored (several of them will be presented in the following sections), starting with considering further ideas from neurobiology. Inhibiting/de-inhibiting mechanisms, dynamic structures of synapses, changing the neurons themselves during their "lives" (maybe "damaging" ones and letting other neurons to replace them), assigning a more important role to the environment, learning and adapting to new inputs from the environment, and so on and so forth, can be issues to investigate.

Then, returning to the theoretical framework, the similarity with Petri nets is visible (both models move "tokens" across a network), and equally the differences (there is no delay in spiking, forgetting rules, counting states in Petri nets,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

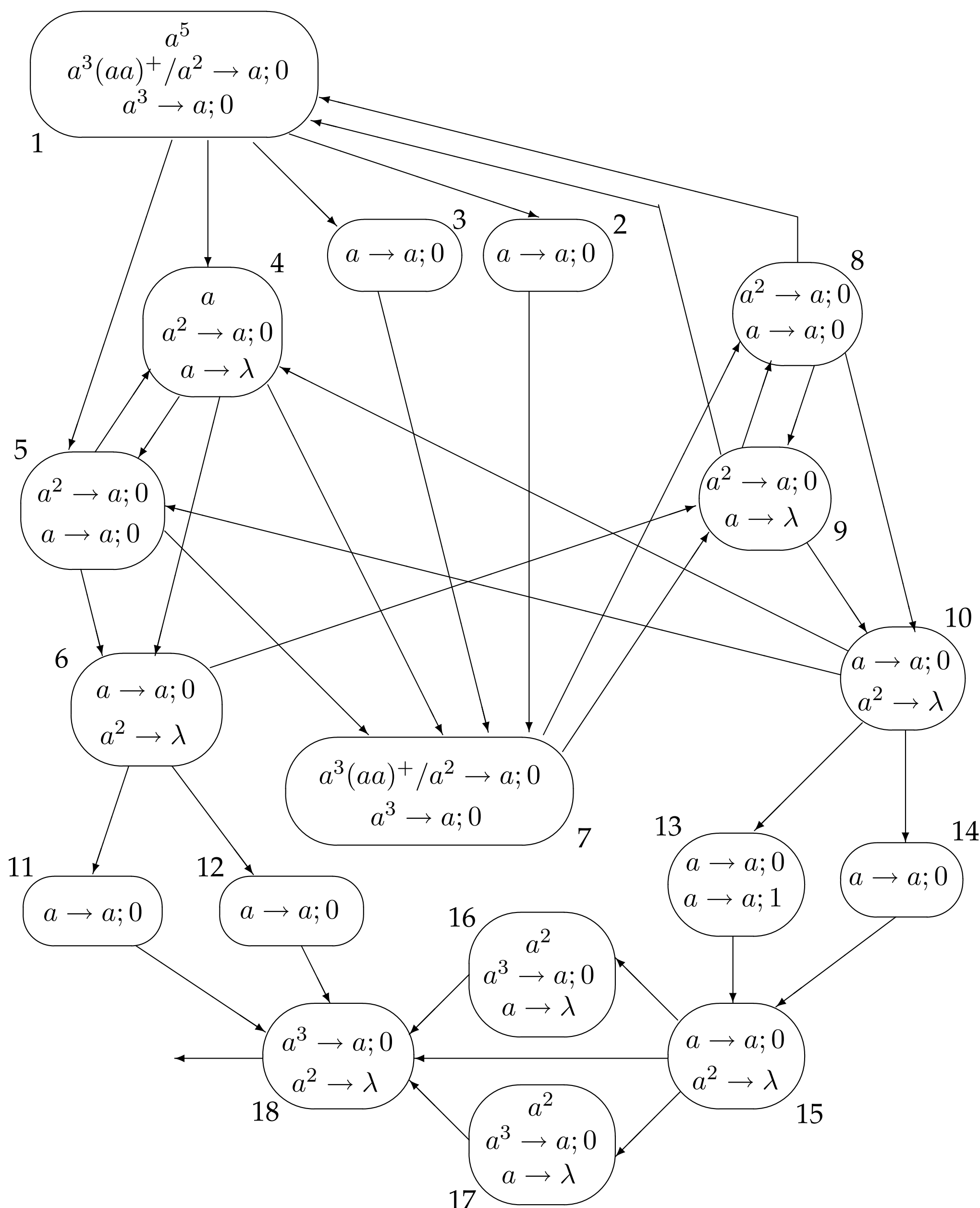


Figure 5.14: A "small" SN P system computing a non-semilinear set of numbers

neither universality for the basic classes of nets); each domain has probably to import ideas and results from the other one.

We conclude this section with a technical open problem. In Section 5.2.2 we have seen that two neurons can compute at most finite sets. How many neurons – with unbounded contents – suffice in order to compute a non-semilinear set of numbers?

Figure 5.14 gives a first answer, using 18 neurons. We start with 5 spikes in neuron 1, one in neuron 4, and two spikes in each neuron 16 and 17. Iteratively,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

the $2n+1$ spikes from neuron 1 are moved to neuron 7, doubling all but the last one spike, hence getting $4n+1$ spikes in neuron 7. The process is somewhat similar to that carried by the modules FIN and SUB from the proof of Theorem 5.2.2 ($2n$ is meant to represent the number n, which thus is doubled when passing from neuron 1 to neuron 7). From neuron 7, all spikes are then moved back to neuron 1. After each cycle, neurons 6 and 10 spike. As long as the rule $a \to a; 0$ is used in neuron 13, the spike of neuron 10 is forgotten on the way towards the output neuron. Similarly, the two spikes coming from neuron 6 to the output neuron are forgotten in neuron 18. When neuron 13 uses the rule $a \to a; 1$, neuron out spikes, once when receiving the signal from neuron 10 and once when receiving the signal from neuron 6, and this means that in between we have m steps, for $2m+1$ being the contents of neuron 1 just moved to neuron 7. We leave the details to the reader and we only mention that $N_2(\Pi) = \{2^n \mid n \geq 2\}$, hence this non-semilinear set belongs to $Spik_2P_{18}(rule_3, cons_3, forg_2)$.

Can these parameters be (significantly) decreased? Which is the smallest number of neurons sufficient for computing a non-semilinear set of numbers?

5.3 On String Languages Generated by SN P Systems

We continue the study of SNP systems by considering these computing devices as binary string generators: the set of spike trains of halting computations of a given system constitutes the language generated by that system. Specifically, with any computation (halting or not) we associate a *spike train*, the sequence of symbols 0 and 1 describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

More formally, let $\Pi = (O, \sigma_1, \dots, \sigma_m, syn, i_0)$ be an SN P system and let γ be a halting computation in Π, $\gamma = C_0 \Longrightarrow C_1 \Longrightarrow \dots \Longrightarrow C_k$ (C_0 is the initial configuration, and $C_{i-1} \Longrightarrow C_i$ is the ith step of γ). Let us denote by $bin(\gamma)$ the string $b_1 b_2 \dots b_k$, where $b_i \in \{0, 1\}$, and $b_i = 1$ if and only if the (output neuron of the) system Π sends a spike into the environment in step i of γ. We denote by B the binary alphabet $\{0, 1\}$, and by $COM(\Pi)$ the set of all halting computations of Π. Moreover, we define the language generated by Π by

$$L_{bin}(\Pi) = \{bin(\gamma) \mid \gamma \in COM(\Pi)\}.$$

In the next sections we will illustrate these definitions with a series of examples.

By $L_{bin}SNP_m(rule_k, cons_p, forg_q)$ we denote the family of languages $L_{bin}(\Pi)$, generated by systems Π with at most m neurons, each neuron having at most k rules, each of the spiking rules consuming at most p spikes, and each forgetting rule removing at most q spikes. As usual, a parameter m, k, p, q is replaced with $*$ if it is not bounded. If the underlying SN P systems are finite (i.e., contain only a bounded number of spikes), we denote the corresponding families of languages by $L_{bin}FSNP_m(rule_k, cons_p, forg_q)$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

5.3.1 An Illustrative Example

We consider a system with a simple architecture, but with an intricate behavior, given pictorially.

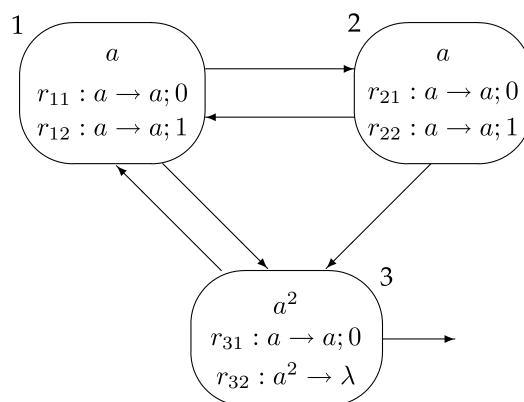


Figure 5.15: The initial configuration of the system Π_1

Figure 5.15 represents the initial configuration of the system Π_1. We have three neurons, labeled with 1, 2, 3, with neuron 3 being the output one. Each neuron contains two rules, with neurons 1 and 2 having the same rules (firing rules which can be chosen in a non-deterministic way, the difference between them being in the delay from firing to spiking), and neuron 3 having one firing and one forgetting rule. In the figure, the rules are labeled, and these labels are useful below.

The evolution of the system Π_1 can be analyzed on a transition diagram as that from Figure 5.16: because the system is finite, the number of configurations reachable from the initial configuration is finite, too, hence, we can place them in the nodes of a graph, and between two nodes/configurations we draw an arrow if and only if a direct transition is possible between them. In Figure 5.16 we have also indicated the rules used in each neuron, with the following conventions: for each r_{ij} we have written only the subscript ij, with *31* being written in italics, in order to indicate that a spike is sent out of the system at that step; when a neuron $i = 1, 2, 3$ uses no rule, we have written $i0$, and when it spikes (after being closed for one step), we write is.

The functioning of the system can easily be followed on this diagram, so that we only briefly describe it. We start with spikes in all neurons. Neurons 1 and 2 behave non-deterministically, choosing one of the two rules. As long as they use the rules $a \to a; 0$, the computation cycles in the initial configuration: neurons 1 and 2 exchange spikes, while neuron 3 forgets its two spikes. If both neurons use the second rule the rule $a \to a; 1$, then both neurons fire, but do not spike immediately, and we reach the configuration $\langle 0/1, 0/1, 0/0 \rangle$. In the next step, both neurons spike, and we return to the initial configuration. When only one of neurons 1, 2 uses the rule $a \to a; 0$ (and therefore spikes immediately) and the other

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

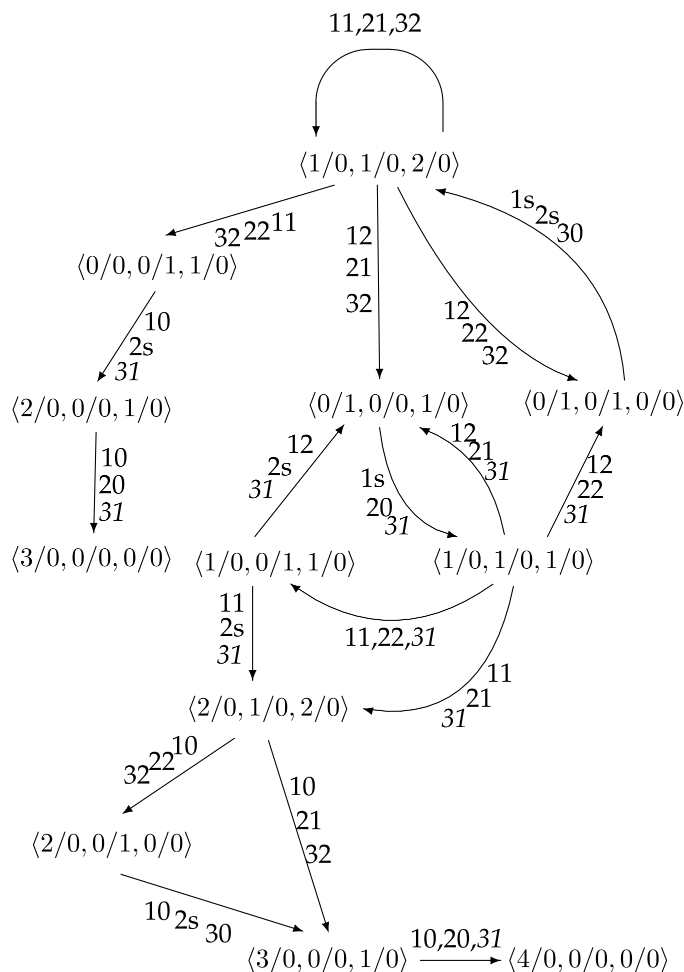


Figure 5.16: The transition diagram of system Π_1

one uses the rule $a \to a; 1$ (and fires, but does not spike, hence, is closed and cannot receive spikes), then only one spike arrives in neuron 3, and in the next step neuron 3 as well as the neuron having used the rule $a \to a; 1$ now send out a spike. If neuron 1 has used the rule $a \to a; 0$ and neuron 2 has used the rule $a \to a; 1$, we reach the configuration $\langle 2/0, 0/0, 1/0 \rangle$; neurons 1 and 2 now cannot apply a rule anymore, thus after one more spiking of neuron 3 we reach the halting configuration $\langle 3/0, 0/0, 0/0 \rangle$. If, conversely, neuron 1 uses the rule $a \to a; 1$ and neuron 2 uses the rule $a \to a; 0$, then we first get the configuration $\langle 0/1, 0/0, 1/0 \rangle$; in the next step, neuron 1 sends its spike to neurons 2 and 3, while neuron 3 also spikes, and "reloads" neuron 1. Then the computation can either run along the cycles depicted in the central part of the diagram from Figure 5.16, or it can reach again the initial configuration, or else it can reach the halting configuration $\langle 4/0, 0/0, 0/0 \rangle$ from the configuration $\langle 2/0, 1/0, 2/0 \rangle$ in two or three steps (as indicated in the bottom part of the diagram).

The transition diagram of a finite SN P system can be interpreted as the

ISBN:978-84-691-9877-3/DL:T-127-2009

representation of a non-deterministic finite automaton, with C_0 being the initial state, the halting configurations being final states, and each arrow being marked with 0 if in that transition the output neuron does not send a spike out, and with 1 if in the respective transition the output neuron spikes; in this way, we can identify the language generated by the system. In the case of the finite SN P system Π_1, the generated language is the following one: $L_{bin}(\Pi_1) = L((0^*0(11 \cup 111)^*110)^*0^*(011 \cup 0(11 \cup 111)^+(0 \cup 0^2)1))$.

We here do not present further examples, because many of the results in the next section are based on effective constructions of SN P systems.

5.3.2 The Language Generative Power of SN P Systems

In what follows we will see that the power of SN P systems used as language generators is rather "ex-centric": "easy" languages cannot be generated, but on the other hand some "hard" languages can be generated.

We start by pointing out an example of the first type.

Theorem 5.3.1 *No language of the form $L_{k,j} = \{0^k, 10^j\}$, for $k \geq 1$, $j \geq 0$, can be generated by an SN P system.*

Proof In order to generate a string 10^j, in the initial configuration the output neuron must contain at least one spike. In such a case, no string of the form 0^k can be generated: if $k = 1$, then we need a forgetting rule $a^r \to \lambda$ which can be applied at the same time with a spiking rule, and this is not possible, by definition (no forgetting rule can be interchanged with a spiking rule); if $k \geq 2$, then in the first step the output neuron should not use a rule, but this is not allowed by the way of defining the computations in a synchronized way. □

The simplest language of the form above is $L_{1,0} = \{0, 1\}$, which does not belong to $L_{bin}SNP_*(rule_*, cons_*, forg_*)$. The same argument works for any language of the form $\{0^k, 1x\}$, where x is an arbitrary string over the binary alphabet.

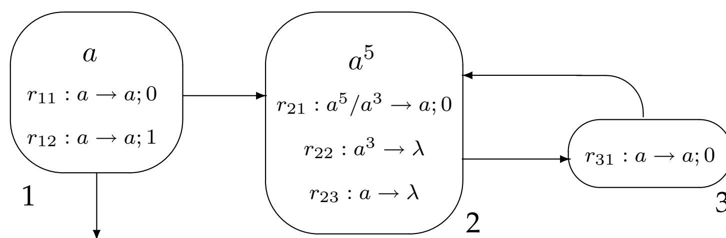


Figure 5.17: An SN P system (Π_2) generating the language $\{100, 01\}$

This does not at all mean that languages consisting of two words similar to those considered above (one word starting with 1 and one with 0) cannot be generated by our systems. An example is the language $\{100, 01\}$, which is generated

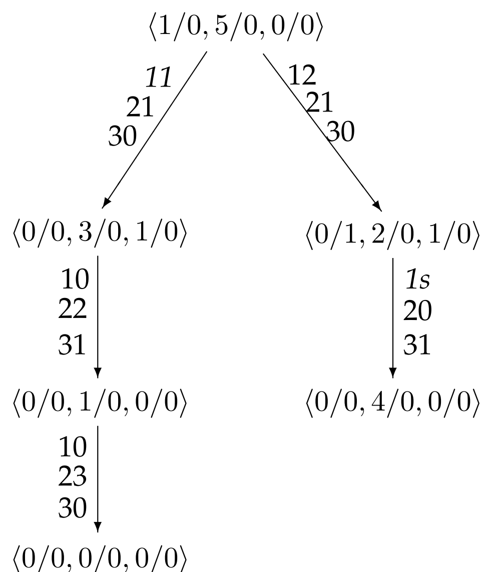


Figure 5.18: The transition diagram of the system Π_2

by the system Π_2 from Figure 5.17; for the reader's convenience, the transition diagram of system Π_2 is given in Figure 5.18.

We remain in the same area, of finite languages, mentioning the following three results.

Theorem 5.3.2 *If* $L = \{x\}$, $x \in B^+$, $|x|_1 = r \geq 0$, *then* $L \in L_{bin}FSNP_2(rule_{r+1}, cons_1, forg_0)$.

Proof Let us consider a string $x = 0^{n_1}10^{n_2}\dots 0^{n_r}10^{n_{r+1}}$, for $n_j \geq 0,\ 1 \leq j \leq r+1$ (if $x = 0^{n_1}$, then $r = 0$). The SN P system from Figure 5.19 generates the string x. The output neuron initially contains r spikes. At step 1, the rule $a^r/a \to a; n_1$ can be applied. One spike is removed and at step $n_1 + 1$ one spike is sent out. We continue in this way until using the rule $a/a \to a; n_r$, i.e., until exhausting the spikes; the last spike is sent out at step $\sum_{i=1}^{r} n_i + r$. The second neuron (not having a synapse with the output one) is meant to make the computation last exactly $|x|$ steps. Note that $|x| - (\sum_{j=1}^{r} n_j + r) = n_{r+1}$, therefore the system halts after generating n_{r+1} more occurrences of 0.

In the case $r = 0$, the system contains no rule in neuron 1, but there is one rule in neuron 2, that is why we have $rule_{r+1}$ in the theorem statement. □

Actually, modulo a supplementary final occurrence of 1, any finite language can be generated.

Theorem 5.3.3 *If* $L \in FIN$, $L \subseteq B^+$, *then* $L\{1\} \in L_{bin}FSNP_1(rule_*, cons_*, forg_0)$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

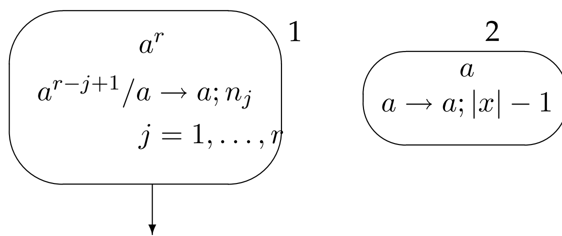


Figure 5.19: An SN P system generating a singleton language

Proof Let us assume that $L = \{x_1, x_2, \ldots, x_m\}$, with $|x_j 1| = n_j \geq 2, 1 \leq j \leq m$; denote $\alpha_j = \sum_{i=1}^{j} n_i$, for all $1 \leq j \leq m$. We write $x_j 1 = 0^{s_{j,1}} 1 0^{s_{j,2}} 1 \ldots 1 0^{s_{j,r_j}} 1$, for $r_j \geq 1$, $s_{j,l} \geq 0$, $1 \leq l \leq r_j$.

An SN P system which generates the language $L\{1\}$ is the following:

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \emptyset, 1), \\
\sigma_1 &= (\alpha_m + 1, R_1), \\
R_1 &= \{a^{\alpha_m+1}/a^{\alpha_m+1-\alpha_j} \to a; s_{j,1} \mid 1 \leq j \leq m\} \\
&\cup \{a^{\alpha_j-t+2}/a \to a; s_{j,t} \mid 2 \leq t < r_j - 1, 1 \leq j \leq m\} \\
&\cup \{a^{\alpha_j-r_j+2} \to a; s_{j,r_j} \mid 1 \leq j \leq m\}.
\end{aligned}$$

Initially, only a rule $a^{\alpha_m+1}/a^{\alpha_m+1-\alpha_j} \to a; s_{j,1}$ can be used, and in this way we non-deterministically chose the string x_j to generate. After $s_{j,1}$ steps, for some $1 \leq j \leq m$, we output a spike, hence, in this way the prefix $0^{s_{j,1}}1$ of the string x_j is generated. Because α_j spikes remain in the neuron, we have to continue with rules $a^{\alpha_j-t+2}/a \to a; s_{j,t}$, for $t = 2$, and then for the respective $t = 3, 4, \ldots, r_j - 1$; in this way we introduce the substrings $0^{s_{j,t}}1$ of x_j, for all $t = 2, 3, \ldots, r_j - 1$. The last substring, $0^{s_{j,r_j}}1$, is introduced by the rule $a^{\alpha_j-r_j+2} \to a; s_{j,r_j}$, which concludes the computation.

We observe that the rules which are used in the generation of a string $x_j 1$ cannot be used in the generation of a string $x_k 1$ with $k \neq j$. □

Corollary 5.3.1 *Every language $L \in FIN$, $L \subseteq B^+$, can be written in the form $L = \partial_1^r(L')$ for some $L' \in L_{bin}FSNP_1(rule_*, cons_*, forg_0)$.*

A sort of "mirror result" can be obtained, based on the idea used in the proof of Theorem 5.3.2.

Theorem 5.3.4 *If $L \in FIN$, $L \subseteq B^+, L = \{x_1, x_2, \ldots, x_n\}$, then $\{0^{i+3}x_i \mid 1 \leq i \leq n\} \in L_{bin}FSNP_*(rule_*, cons_1, forg_0)$.*

Proof For each $x_i \in L$, $x_i = 0^{n_{i,1}} 1 0^{n_{i,2}} 1 \ldots 1 0^{n_{i,r_i}} 1 0^{n_{i,r_i+1}}$, there is a system as in the proof of Theorem 5.3.2, consisting of a neuron which outputs spikes in the moments which correspond to the digits 1 of x_i, and a companion neuron which

ISBN:978-84-691-9877-3/DL:T-127-2009

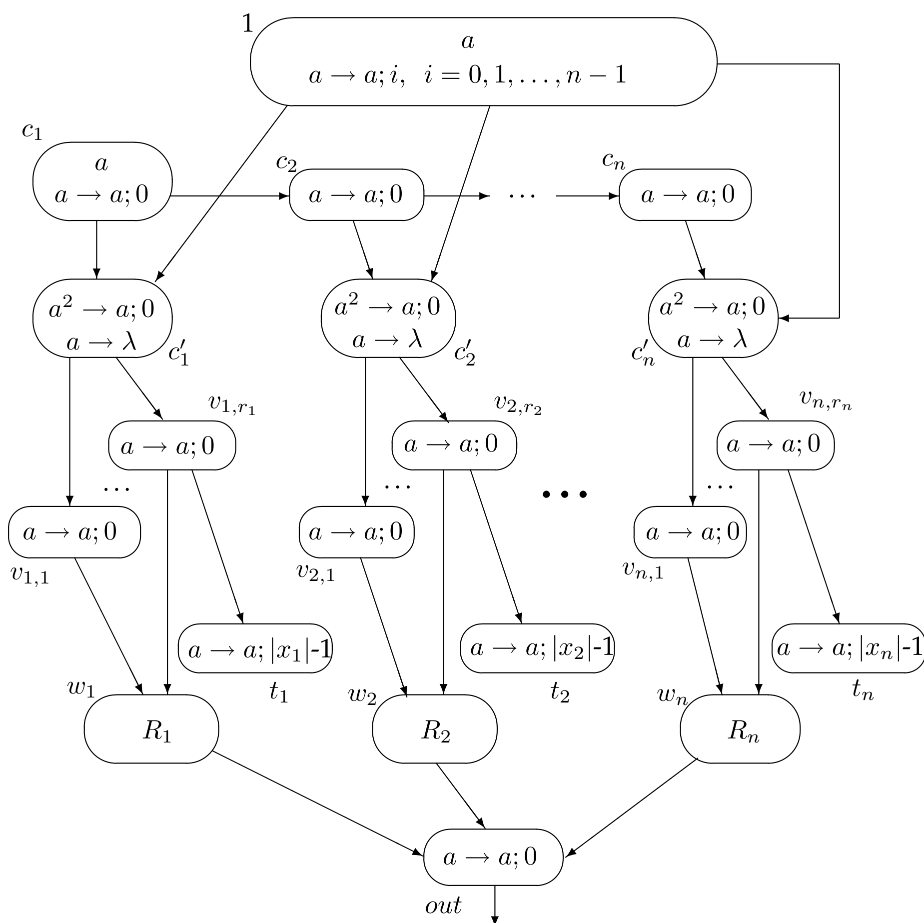


Figure 5.20: An SN P system generating a finite set, prefixed by zeroes

just makes sure that the computation lasts $|x_i|$ steps. We combine such subsystems, each one taking care of one string x_i, into a system as that from Figure 5.20.

In the bottom of Figure 5.20 there are the modules (w_i, t_i) for generating x_i, $1 \leq i \leq n$, as in Figure 5.19, except that there is no spike in the neurons w_i; the corresponding sets of rules are denoted by $R_1, R_2, \ldots, R_n$. If there is an i such that $|x_i|_1 = 0$ (note that in this case we have $R_i = \emptyset$), then we set $r_i = 1$ in the construction from Figure 5.20. The work of the system is triggered by neuron 1, which selects one of its rules to be applied non-deterministically in step 1: if the rule $a \to a; i$ is applied, then the module for generating x_{i+1} is activated, $i = 0, 1, , \ldots, n-1$. The neurons c_i, $1 \leq i \leq n$, count the time steps, so that at step i one spike is sent to c_{i+1} and one to c'_i. All these spikes are forgotten, except the one which arrives in c'_i at the same time with the spike emitted by neuron 1, and these spikes load the necessary number of spikes in the corresponding working module w_i, and also send one spike to the timing neuron t_i. As in the proof of Theorem 5.3.2, these two neurons ensure the generation of x_i – with $i+3$ occurrences of 0 in the left hand, corresponding to the $i+2$ steps necessary for the spike

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

of neuron 1 to reach the working neurons and to the step necessary for passing a spike from neuron w_i to neuron out. □

We now pass to investigating the relationships with the family of regular languages, and we start with a result already proved by the considerations above.

Theorem 5.3.5 *The family of languages generated by finite SN P systems is strictly included in the family of regular languages over the binary alphabet.*

Proof The inclusion follows from the fact that for each finite SN P system we can construct the corresponding transition diagram associated with the computations of the SN P system and then interpret it as the transition diagram of a finite automaton (with an arc labeled by 1 when the output neuron spikes and labeled by 0 when the output neuron does not spike) as already done in the example of Section 5.3.1. The strictness is a consequence of Theorem 5.3.1. □

However, each regular language, over any alphabet, not only on the binary one, can be represented in an easy way starting from a language in $L_{bin}FSNP_*(rule_*, cons_*, forg_*)$.

Theorem 5.3.6 *For any language $L \subseteq V^*$, $L \in REG$, there is a finite SN P system Π and a morphism $h : V^* \longrightarrow B^*$ such that $L = h^{-1}(L_{bin}(\Pi))$.*

Proof Let $V = \{a_1, a_2, \ldots, a_k\}$ and let $L \subseteq V^*$ be a regular language. Consider the morphism $h : V^* \longrightarrow B^*$ defined by

$$h(a_i) = 0^{i+1}1, \ 1 \leq i \leq k.$$

The language $h(L)$ is regular. Consider a right-linear grammar $G = (N, B, S, P)$ such that $L(G) = h(L)$ and having the following properties:

1. $N = \{A_1, A_2, \ldots, A_n\}, n \geq 1$, and $S = A_n$,
2. the rules in P are of the forms $A_i \rightarrow 0^s 1 A_j$ or $A_i \rightarrow 0^s 1$, for $s \in \{2, 3, \ldots, k+1\}, i, j \in \{1, 2, \ldots, n\}$.

A grammar with these properties can easily be found (the first property is a matter of renaming the nonterminals, the second property is ensured by the fact that the strings of $h(L)$ consist of blocks of the form $0^s 1$ for $2 \leq s \leq k+1$).

For uniformity, let us assume that there exists a further nonterminal, A_0, and that all terminal rules $A_i \rightarrow 0^s 1$ are replaced by $A_i \rightarrow 0^s 1 A_0$, hence, all rules have the generic form $A_i \rightarrow 0^s 1 A_j$. It is important to note that we always have at least two occurrences of 0 in the rules.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

We construct the following SN P system:

$$\begin{aligned}
\Pi &= (\{a\}, \sigma_1, \ldots, \sigma_{n+3}, n+3),\\
\sigma_1 &= (2, \{a^2/a \to a; 0,\ a^2/a \to a; 1,\ a \to \lambda\}),\\
\sigma_2 &= (2, \{a^2/a \to a; 0,\ a \to \lambda\}),\\
\sigma_i &= (0, \{a \to a; 0\}), i = 3, 4, \ldots, n+2,\\
\sigma_{n+3} &= (2n, \{a^{n+i}/a^{n+i-j} \to a; s \mid A_i \to 0^s 1 A_j \in P\}\\
&\cup\ \{a^j \to a; k+2 \mid 1 \leq j \leq n\}).
\end{aligned}$$

For an easier understandability, the system is also given graphically, in Figure 5.21.

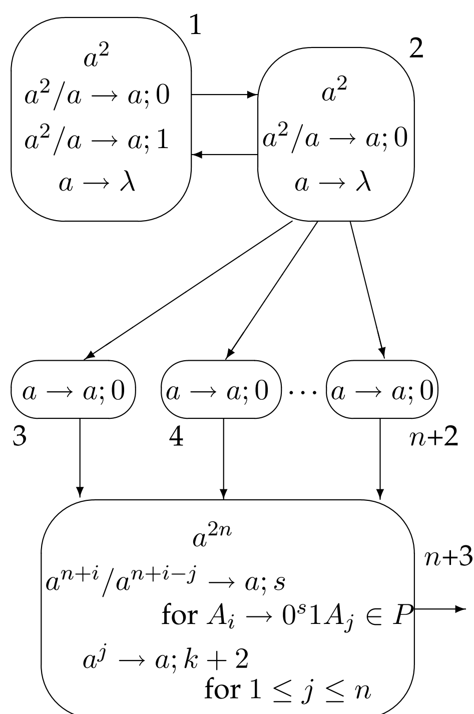


Figure 5.21: The SN P system from the proof of Theorem 5.3.6

The output neuron already fires in the first step, by a rule $a^{2n}/a^{2n-j} \to a; s$ associated with a rule $A_n \to 0^s 1 A_j$ from P (where A_n is the axiom of G) and its spike exits the system in step $s+1$. Because $s \geq 2$, the neuron $n+3$ is closed at least two steps, hence, no spike can enter from neurons $3, 4, \ldots, n+2$ in these steps (this is true in all subsequent steps of the computation when rules of G are simulated).

Neurons 1 and 2 are meant to continuously "reload" neuron $n+3$ with n spikes, through the intermediate "multiplier neurons" $3, 4, \ldots, n+2$: as long as neuron 1 uses the rule $a^2/a \to a; 0$, neurons 1 and 2 send to each other a spike,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

returning to the initial state, while neuron 2 also sends a spike to all neurons $3, 4, \ldots, n+2$. In each step, neuron 1 can however use the rule $a^2/a \to a; 1$. Simultaneously, neuron 2 spikes, but its spike does not enter neuron 1. In the next step, neuron 2 uses its forgetting rule $a \to \lambda$, and receives one spike from neuron 1. This spike is forgotten in the next step, and the work of neurons 1 and 2 ends. In this way, also the reloading of neuron $n+3$ stops.

Let us now return to the work of neuron $n+3$ and assume that we have $n+i$ spikes in it, for some $1 \leq i \leq n$ (initially, $i = n$). The only rule which can be used is $a^{n+i}/a^{n+i-j} \to a; s$, for $A_i \to 0^s 1 A_j \in P$. There remain j spikes; if the neuron receives n spikes from neurons $3, 4, \ldots, n+2$ in step $s+1$ (the spikes sent earlier are lost), then the output neuron ends the step $s+1$ with $n+j$ spikes inside. If $j \geq 1$, then the simulation of rules in G can be repeated.

If in the moment when a rule $a^{n+i}/a^{n+i-0} \to a; s$ is applied (i.e., a rule $A_i \to 0^s 1 A_0$ is simulated) the output neuron does not receive further spikes from neurons $3, 4, \ldots, n+2$, which means that neurons 1, 2 have finished their work, then no spike remains in the system and the computation halts. The generated string is one from $L(G) = h(L)$.

If the work of neurons 1 and 2 stops prematurely, i.e., in neuron $n+3$ we apply a rule $a^{n+i}/a^{n+i-j} \to a; s$ and no spike comes from neurons $3, 4, \ldots, n+2$ in step $s+1$, then the rule $a^j \to a; k+2$ is immediately applied, and the computation stops after producing the string $0^{k+2}1$, as a suffix of the generated string.

Similarly, if after using a rule $a^{n+i}/a^{n+i-0} \to a; s$ we still receive spikes from neurons $3, 4, \ldots, n+2$, then in the next step the rule $a^n \to a; k+2$ is used, and again the string $0^{k+2}1$ is introduced, possibly repeated several times before the computation halts.

Therefore, $h(L) \subseteq L_{bin}(\Pi)$ and $L_{bin}(\Pi) - h(L) \subseteq B^*\{0^{k+2}1\}^+$. Because a string containing a substring 0^{k+2} is not in $h(V^*)$ and because h is injective, we have $h^{-1}(L_{bin}(\Pi)) = L$. □

As expected, the power of SN P systems goes far beyond the regular languages. We first illustrate this assertion with an example, namely, the system Π_3 from Figure 5.22, for which we have $L_{bin}(\Pi_3) = \{0^{n+4}1^{n+4} \mid n \geq 0\}$; observe that due to the rule $a(aa)^+/a^2 \to a; 0$ in neuron 10 this SN P system is not finite.

The reader can check that in $n \geq 0$ steps when neuron 1 uses the rule $a^2/a \to a; 0$ the output neuron accumulates $2n+6$ spikes. When neuron 1 uses the rule $a^2/a \to a; 1$, one more spike will arrive in neuron 10 (in step $n+4$). In this way, the number of spikes present in neuron 10 becomes odd, and the rule $a(aa)^+/a^2 \to a; 0$ can be repeatedly used until only one spike remains; this last spike is used by the rule $a \to a; 0$, thus $n+4$ occurrences of 1 are produced.

Much more complex languages can be generated. First, the previous construction can be extended to non-context-free languages consisting of strings of the form $0^{n_1}1^{n_2}0^{n_3}$ with a precise relation between n_1, n_2, n_3. Then, languages with non-semilinear blocks in their strings can be generated.

Theorem 5.3.7 $L_{bin}SNP_{22}(rule_3, cons_3, forg_3) - MAT \neq \emptyset$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

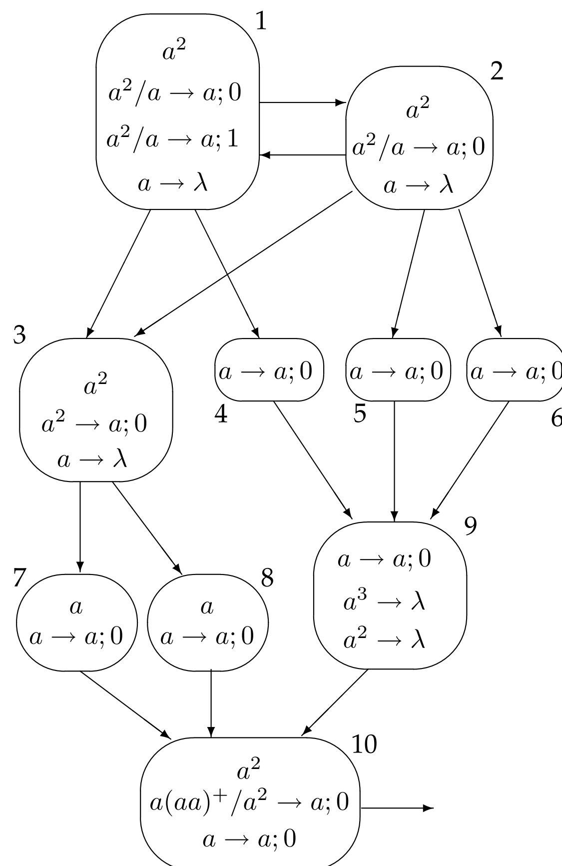


Figure 5.22: An SN P system generating a non-regular language

Proof In Figure 5.14 from Section 5.2.3 one considers an SN P system Π (with 18 neurons, of the complexity described by $rule_3, cons_3, forg_3$) which produces all spike trains of the form $0^k 1 0^{2^n} 1 0 w$, for any $n \geq 2$, and some $k \geq 1$ and an infinite binary sequence w. To this system we add a *halting module* as that suggested in Figure 5.23, which waits until receiving two spikes from the output neuron of Π, then sends three spikes to neurons 6 and 10 of Π; these neurons play an important rôle in iterating the work of Π, but they cannot handle more than two spikes. In this way, the work of Π is blocked after producing two spikes. (The same effect is obtained if we send three spikes to all neurons of Π, except the output one, so the reader should not mind which is the precise role of neurons 6 and 10 in Π.) Thus, the obtained system, let us denote it by Π', will halt after sending out two spikes, hence, it generates a language $L_{bin}(\Pi')$ included in $\{0\}^*\{1 0^{2^n} 1 \mid n \geq 2\}\{0\}^*$ such that

$$((\{0\}^*\{1\}) \backslash L_{bin}(\Pi'))/(\{1\}\{0\}^*) = \{0^{2^n} \mid n \geq 1\}.$$

The family MAT is closed under right and left quotients by regular languages [22] and all one-letter matrix languages are regular [34], therefore $L_{bin}(\Pi') \notin MAT$. □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

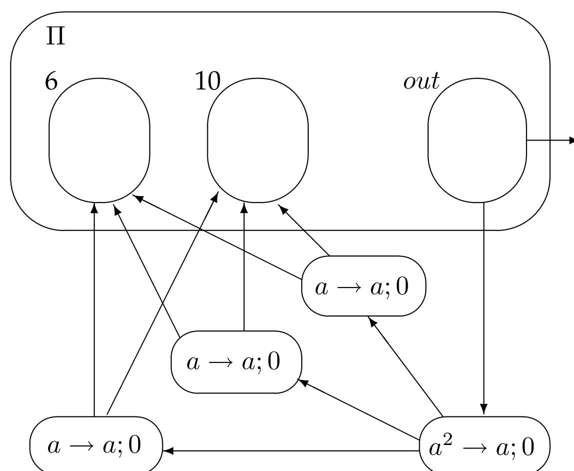


Figure 5.23: A halting module (as used in the proof of Theorem 5.3.7)

The strong restriction that in order to produce a string of length n we have to work exactly n steps (and the fact that the workspace used during these steps cannot increase exponentially) directly implies the fact that languages generated by SN P systems are recursive. Indeed, their membership problem can be solved in the following easy way: consider a string x and a system Π; start from the initial configuration of Π and construct the computation tree with $|x| + 1$ levels, then check whether there is a path in this tree which corresponds to a halting computation and which produces the string x. Therefore, we have the following result.

Theorem 5.3.8 $L_{bin}SNP_*(rule_*, cons_*, forg_*) \subset REC$.

We do not know whether this result can be improved to the inclusion $L_{bin}SNP_*(rule_*, cons_*, forg_*) \subset CS$.

However, a characterization of recursively enumerable languages is possible in terms of languages generated by SN P systems.

Theorem 5.3.9 *For every alphabet $V = \{a_1, a_2, \dots, a_k\}$ there are a morphism $h_1 : (V \cup \{b, c\})^* \longrightarrow B^*$ and a projection $h_2 : (V \cup \{b, c\})^* \longrightarrow V^*$ such that for each language $L \subseteq V^*$, $L \in RE$, there is an SN P system Π such that $L = h_2(h_1^{-1}(L_{bin}(\Pi)))$.*

Proof The two morphisms are defined as follows:

$$\begin{aligned}
h_1(a_i) &= 10^i 1, \text{ for } i = 1, 2, \dots, k, \\
h_1(b) &= 0, \\
h_1(c) &= 01, \\
h_2(a_i) &= a_i, \text{ for } i = 1, 2, \dots, k, \\
h_2(b) &= h_2(c) = \lambda.
\end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

For a string $x \in V^*$, let us denote by $val_k(x)$ the value in base $k+1$ of x (we use base $k+1$ in order to consider the symbols $a_1, \ldots, a_k$ as digits $1, 2, \ldots, k$, thus avoiding the digit 0 in the left hand of the string). We extend this notation in the natural way to sets of strings. Now consider a language $L \subseteq V^*$. Obviously, $L \in RE$ if and only if $val_k(L)$ is a recursively enumerable set of numbers. In turn, a set of numbers is recursively enumerable if and only if it can be accepted by a deterministic register machine. Let M be such a register machine, i.e., $N(M) = val_k(L)$.

We construct an SN P system Π performing the following operations (c_0 and c_1 are two distinguished neurons of Π, the first one being empty and the second one having three spikes in the initial configuration):

1. Output a spike in the first time unit.

2. For some $1 \leq i \leq k$, output no spike for i steps, but introduce the number i in neuron c_0; in the construction below, a number n is represented in a neuron by storing there $3n$ spikes, i.e., the previous task means introducing $3i$ spikes in neuron c_0.

3. When this operation is finished, output a spike (hence, up to now we have produced a string 10^i1).

4. Multiply the number stored in neuron c_1 (initially, we here have number 0) by $k+1$, then add the number from neuron c_0; specifically, if neuron c_0 holds $3i$ spikes and neuron c_1 holds $3m$ spikes, $m \geq 0$, we end this step with $3(m(k+1)+i)$ spikes in neuron c_1 and no spike in neuron c_0. In the meantime, the system outputs no spike (hence, the string was continued with a number of occurrences of 0; this number depends on the duration of the operation above, but it is greater than 1). When the operation is completed, output two spikes in a row (hence, the string is continued with 11).

5. Repeat from step 2, or, non-deterministically, stop the increase of spikes from neuron c_1 and pass to the next step.

6. After the last increase of the number of spikes from neuron c_1 we have got $val_k(x)$ for a string $x \in V^+$ such that the string produced by the system up to now is of the form $10^{i_1}10^{j_1}110^{i_2}10^{j_2}11 \ldots 110^{i_m}10^{j_m}$, for $1 \leq i_l \leq k$ and $j_l \geq 1$, for all $1 \leq l \leq m$, i.e., $h_1(x) = 10^{i_1}110^{i_2}1 \ldots 10^{i_m}1$. We now start to simulate the work of the register machine M in recognizing the number $val_k(x)$. During this process, we output no spike, but we output one if (and only if) the machine M halts, i.e., when it accepts the input number, which means that $x \in L$. After emitting this last spike, the system halts. Therefore, the previous string $10^{i_1}10^{j_1}110^{i_2}10^{j_2}11 \ldots 110^{i_m}10^{j_m}$ is continued with a suffix of the form 0^s1 for some $s \geq 1$.

From the previous description of the work of Π, it is clear that we stop, after producing a string of the form $y = 10^{i_1}10^{j_1}110^{i_2}10^{j_2}11 \ldots 110^{i_m}10^{j_m}0^s1$ as above, if

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

and only if $x \in L$. Moreover, it is obvious that $x = h_2(h_1^{-1}(y))$: we have $h_1^{-1}(y) = a_{i_1}b^{j_1-1}ca_{i_2}b^{j_2-1}c\ldots a_{i_m}b^{j_m+s-1}c$ (this is the only way to cover correctly the string x with blocks of the forms of $h_1(a_i), h_1(b), h_1(c)$); the projection h_2 simply removes the auxiliary symbols b, c.

Now, it remains to construct the system Π.

Instead of constructing it in all details, we rely on the fact that a register machine can be simulated by an SN P system, as already shown in the previous section – for the sake of completeness and because of some minor changes in the construction, we below recall the details of this simulation. Then, we also suppose that the multiplication by $k+1$ of the contents of neuron c_1 followed by adding a number between 1 and l is done by a register machine (with the numbers stored in neurons c_0, c_1 introduced in two specified registers); we denote this machine by M_0. Thus, in our construction, also for this operation we can rely on the general way of simulating a register machine by an SN P system. All other modules of the construction (introducing a number of spikes in neuron c_0, sending out spikes, choosing non-deterministically to end the string to generate and switching to the checking phase, etc.) are explicitly presented below.

A delicate problem which appears here is the fact that the simulations of both machines M_0 and M have to use the same neuron c_1, but the correct work of the system (the fact that the instructions of M_0 are not mixed with those of M) will be explained below.

The overall appearance of Π is given in Figure 5.24, where M_0 indicates the subsystem corresponding to the simulation of the register machine $M_0 = (m_0, I_0, l_{0,0}, l_{h,0})$ and M indicates the subsystem which simulates the register machine $M = (m, I, l_0, l_h)$. Of course, we assume $lab(M_0) \cap lab(M) = \emptyset$.

We start with spikes in neurons 6, 7, 8, and 18 (besides the three spikes from neuron c_1), hence, we spike in the first step. As long as neurons 6, 7 do not receive a spike from neuron 8, they spike and send a spike to each other and three spikes to neuron c_0.

If neuron 8 starts by using some rule $a \to a; i-1$, $1 \leq i \leq k$, then after $i-1$ steps a spike is sent from neuron 8 to all neurons 6, 7 (which stop working), 4, 5 (which load two spikes in neuron $l_{0,0}$, thus starting the simulation of the register machine M_0), and 17; from here, the spike goes to the output neuron 18, which spikes exactly in the moment when the simulation of M_0 starts.

Now, the subsystem corresponding to the register machine M_0 works a number of steps (at least one); after a while the computation in M_0 stops, by activating the neuron $l_{h,0}$ (this neuron will get two spikes in the end of the computation and will spike; see below). This neuron sends a spike to both neurons 9 and 10.

Neuron 9 is the one which non-deterministically chooses to continue the string (the case of using the rule $a \to a; 0$) or to stop growing the string and to pass to checking whether it is in our language (the case of using the rule $a \to a; 1$). If both neurons 9 and 10 spike immediately, then neuron 14 fires, but neuron 15 forgets the two spikes.

Neuron 14 sends a spike to the output neuron and one to neuron 16. In the

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

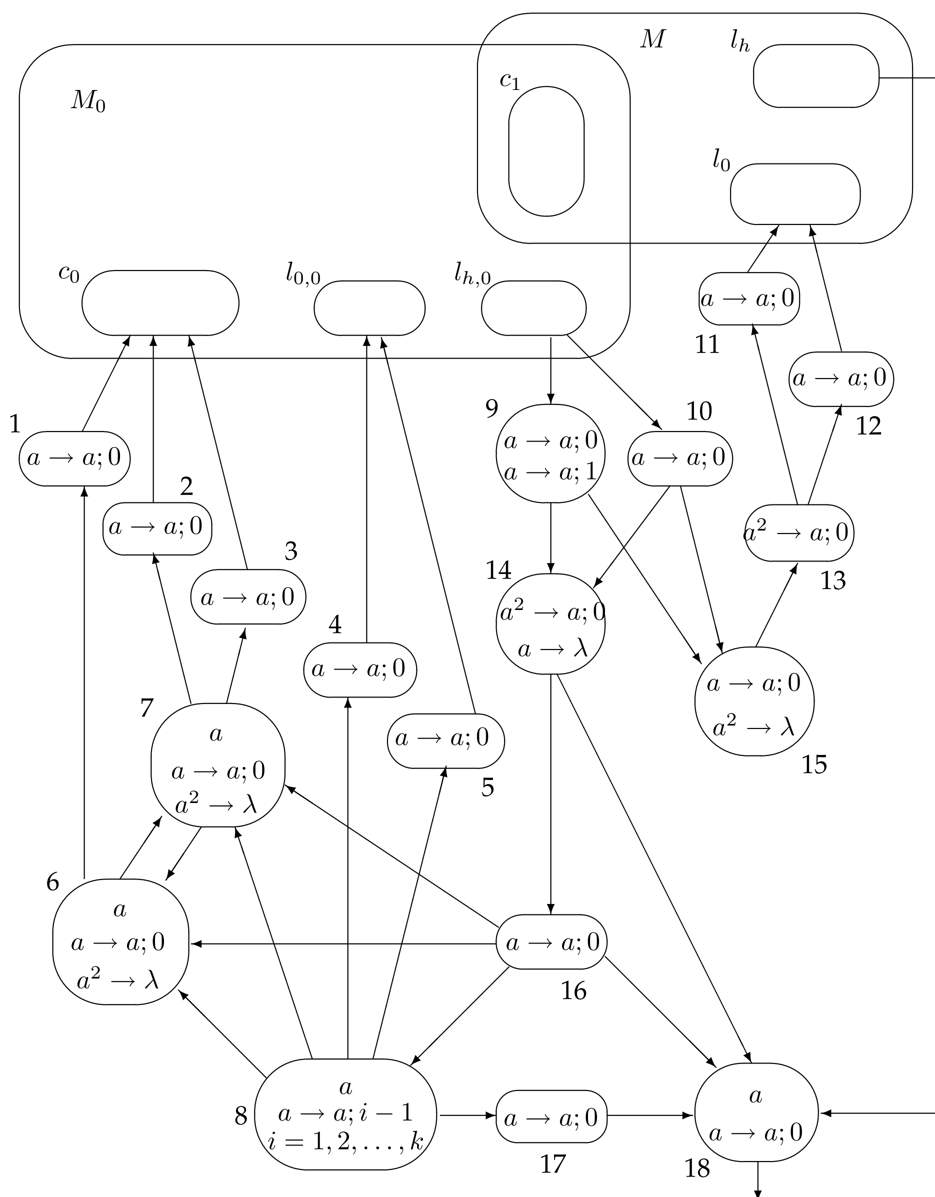


Figure 5.24: The structure of the SN P system from the proof of Theorem 5.4.6

next step, besides sending a spike outside, the system returns neurons 6, 7, 8, and 18 to the initial state (having one spike). This means that a sequence of two spikes are sent out, and the process continues by introducing another substring 0^i1 in the string produced by the system.

When neuron 9 uses the rule $a \to a; 1$, the spikes of neurons 10 and 9 arrive, in this order, in neuron 15, which spikes and sends the spikes to neuron 13. This neuron waits for the two spikes, and, after having both of them, spikes and thus sends two spikes to l_0, the initial label of the register machine M. We start simulating the work of this machine, checking whether or not the number stored in

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

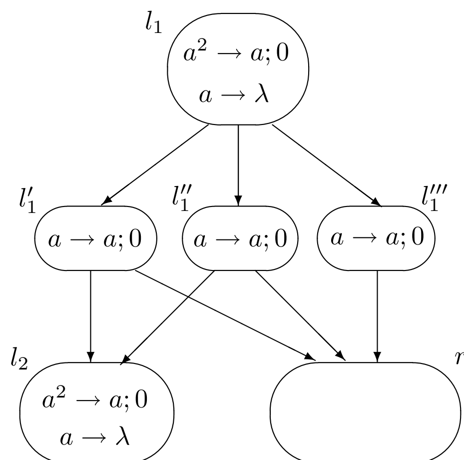


Figure 5.25: Module ADD (simulating $l_1 : (\mathtt{ADD}(r), l_2)$)

neuron c_1 belongs to $val_k(L)$. In the affirmative case, neuron l_h is activated, it sends a spike to the output neuron of the system, and the computation stops. If the number stored in neuron c_1 is not accepted, then the computation continues forever, hence, the system does not produce a string.

In order to complete the proof we have to show how the two register machines are simulated, using the common neuron c_1 but without mixing the computations. To this aim, we consider the modules ADD and SUB from Figures 5.25 and 5.26. Neurons are associated with each label of the machine (they fire if they have two spikes inside and forget a single spike), with each register (with $3t$ spikes representing the number t from the register), and there also are additional neurons with primed labels – it is important to note that all these additional neurons have distinct labels.

The simulation of an ADD instruction is easy, we just add three spikes to the respective neuron; no rule is needed in the neuron. The instructions SUB of machine M_0 are simulated by modules as in the left side of Figure 5.26 and those of M by modules as in the right hand of the figure. The difference is that the rules for M_0 fire for a content of the neuron described by the regular expression $(a^3)^+a$ while the rules for M fire for a content of the neuron described by the regular expression $(a^3)^+a^2$ (that is why the module for M has two additional neurons, g_1', g_2'). This ensures the fact that the rules of M_0 are not used instead of those of M or vice versa. It is also important to note that the neurons corresponding to the labels of the register machines need two spikes to fire, hence, the unique spike sent by neuron c_1 (and by other neurons involved in subtraction instructions) to other neurons than the correct ones identified by the instruction are immediately forgotten. □

The previous theorem given a characterization of recursively enumerable languages, because the family RE is closed under direct and inverse morphisms.

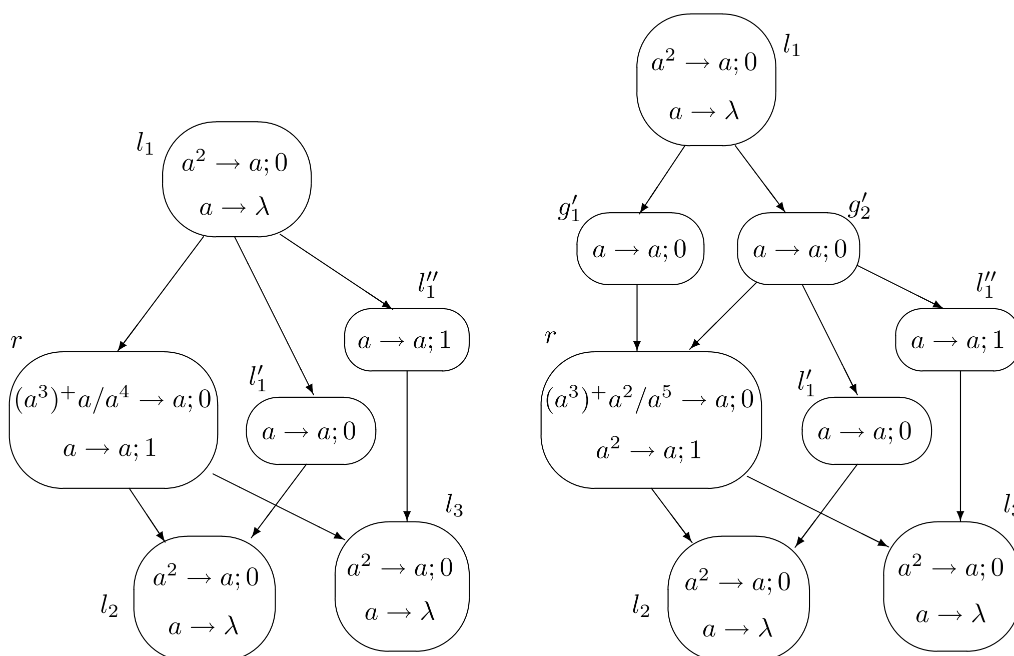


Figure 5.26: Module SUB (simulating $l_1 : (\texttt{SUB}(r), l_2, l_3)$)

Corollary 5.3.2 *The family $L_{bin}SNP_*(rule_*, cons_*, forg_*)$ is incomparable with all families of languages FL such that $FIN \subseteq FL \subset RE$ (even if we consider only languages over the binary alphabet) which are closed under direct and inverse morphisms.*

The system Π constructed in the proof of Theorem 5.4.6 depends on the language L, while the morphisms h_1, h_2 only depend on the alphabet V. Can this property be reversed, taking the system Π depending only on the alphabet V and the morphisms (or other stronger string mappings, such as a gsm mapping) depending on the language? A possible strategy of addressing this question is to use a characterization of RE languages starting from fixed languages, such as the twin-shuffle language [23] or the copy languages [24].

5.3.3 Remarks and Future Research

We have considered the natural question of using spiking neural P systems as language generators, and we have investigated their power with respect to families in the Chomsky hierarchy. Several topics remain to be investigated, mainly, concerning possible changes in the definition of SN P systems, starting with considering different types of rules. For instance, what about using forgetting rules of the form $E/a^r \to \lambda$, with E being a regular expression like in firing rules? Another extension is to have rules of the form $E/a^c \to a^p; d$, with $p \geq 1$ (at least in the output neuron): when we output i spikes in one moment, we can record a symbol b_i in the generated string, and in this way we produce strings over ar-

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

bitrary alphabets, not only on the binary one. This possibility is investigated in Section 5.5 below.

Another variant of interest is to consider a sort of rough-set-like rule, of the form $(s, S)/a^c \to a^p; d$, with $0 \leq s \leq c \leq S \leq \infty$. The meaning is that if the number of spikes from the neuron is k and $s \leq k \leq S$, then the rule fires and c spikes are consumed. This reminds the lower and the upper approximations of sets in rough sets theory [76], [77] and it is also well motivated from the neurobiology point of view (it reminds the sigmoid function of neuron exciting). It is interesting to note that all rules of the form $a^n/a^c \to a; d$ are of this type: just take $(n, n)/a^c \to a; d$. Thus, all examples and results until Theorem 5.3.6, including this theorem, are valid also for SN P systems with rough-set-like rules. Are there such systems able to generate non-regular languages? This is a question worth to be considered.

It is also of interest to see whether or not languages from other families can be represented starting from languages generated by specific classes of SN P systems and using various operations with languages (as we have done here with the regular languages – Theorem 5.3.6, and with recursively enumerable languages – Theorem 5.3.9). For instance, are there such representations – maybe using other operations and/or restricted/extended variants of SN P systems – for other families of languages from the Chomsky hierarchy, such as CF and CS?

5.4 Trace Languages Associated to SN P Systems

We continue our investigation of SN P systems by incorporating them the notion investigated in Chapter 3: the languages of traces generated by a special spike.

Now, we distinguish a spike by "marking" it and we follow its path through the neurons of the system, thus obtaining a language.

In contrast to the previous spiking models, the current one will miss the output neuron, but we allowe synapses $(i, 0)$ for any neuron σ_i; actually, in the trace case investigated here we do not really need such synapses, but we consider them just because such links with the environment are "realistic".

Consider an SN P system of degree $m \geq 1$, $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn)$ defined as above, and distinguish one of the neurons of the system as the *input* one (thus, we add a further component, in, to the system description, with $in \in \{1, 2, \ldots, m\}$). In the initial configuration of the system we "mark" one spike from this neuron – the intuition is that this spike has a "flag" – and we follow the path of this flag during the computation, *recording the labels of the neurons where the flag is present in the end of each step*. Actually, for neuron σ_i we consider the symbol b_i in the trace string. (When presenting the initial configuration of the system, the number n_{in}, of spikes present in the input neuron, is written in the form n'_{in}, to indicate that the marked spike is here.)

The previous definition contains many delicate points which need clarifications – and we use a simple example to do this.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Assume that in neuron σ_i we have three spikes, one of them marked; we write aaa' to represent them. Assume also that we have a spiking rule $aaa/aa \to a; 0$. When applied, this rule consumes two spikes, one remains in the neuron and one spike is produced and sent along the synapses going out of neuron σ_i. Two cases already appear: the marked spike is consumed or not. If not consumed, it remains in the neuron. If consumed, then the flag passes to the produced spike. Now, if there are two or more synapses going out of neuron σ_i, then again we can have a branching: only one spike is marked, hence only on one of the synapses (i, j) we will have a marked spike and that synapse is non-deterministically chosen (on other synapses we send non-marked spikes). If σ_j is an open neuron, then the marked spike ends in this neuron. If σ_j is a closed neuron, then the marked spike is lost, and the same happens if $j = 0$ (the marked spike exits in the environment). Anyway, if the marked spike is consumed, at the end of this step it is no longer present in neuron i; it is in neuron σ_j if $(i, j) \in syn$ and neuron σ_j is open, or it is removed from the system in other cases.

Therefore, if in the initial configuration of the system neuron σ_i contains the marked spike, then the trace can start either with b_i (if the marked spike is not consumed) or with b_j (if the marked spike was consumed and passed to neuron σ_j); if the marked spike is consumed and lost, then we generate the empty string, which is ignored in our considerations. Similarly in later steps.

If the rule used is of the form $aaa/aa \to a; d$, for some $d \geq 1$, and the marked spike is consumed, then the newly marked spike remains in neuron σ_i for d steps, hence the trace starts/continues with b_i^d. Similarly, if no rule is used in neuron σ_i for k steps, then the trace records k copies of b_i.

If a forgetting rule is used in the neuron where the marked spike is placed, then the trace string stops (and no symbol is recorded for this step).

Therefore, when considering the possible branchings of the computation, we have to take into account the non-determinism not only in using the spiking rules, but also in consuming the marked spike and in sending it along one of the possible synapses.

The previous discussion has, hopefully, made clear what we mean by *recording the labels of the neurons where the flag is present in the end of each step*, and why we have chosen the end of a step and not the beginning: in the latter case, all traces would start with the same symbol, corresponding to the input neuron, which is a strong – and artificial – restriction.

In the next section, we will illustrate all these points by examining in detail an example.

Anyway, we take into account only halting computations: irrespective whether or not a marked spike is still present in the system, the computation should halt (note that it is possible that the marked spike is removed and the computation still continues for a while – but this time without adding further symbols to the trace string).

For an SN P system Π we denote by $T(\Pi)$ the language of all strings describing the traces of the marked spike in all halting computations of Π. Then, we

denote by $TSNP_m(rule_k, cons_p, forg_q)$ the family of languages $T(\Pi)$, generated by systems Π with at most m neurons, each neuron having at most k rules, each of the spiking rules consuming at most p spikes, and each forgetting rule removing at most q spikes. As usual, a parameter m, k, p, q is replaced with $*$ if it is not bounded.

5.4.1 Two Examples

We consider here two examples, both illuminating the previous definitions and relevant for the intricate work of SN P systems as language generators by means of traces; indications on the power of these devices are also obtained in this way.

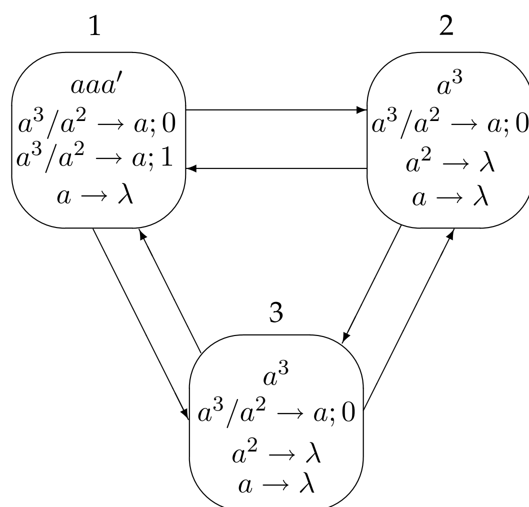


Figure 5.27: The initial configuration of system Π_1

We start with a system already having a complex behavior, the one whose initial configuration is given in Figure 5.27 (we denote it by Π_1), with the marked spike indicated by a prime; synapses of the form $(i, 0)$ are indicated by arrows pointing to the environment.

We have three neurons, labeled with 1, 2, 3, with neuron σ_1 being the input one. Each neuron contains three rules, and only neuron σ_1 has two spiking rules, but the non-determinism of the system is considerable, due to the possible traces of the marked spike.

The evolution of the system Π_1 can be analyzed on a transition diagram as that from Figure 5.28: because the number of configurations reachable from the initial configuration is finite, we can place them in the nodes of a graph, and between two nodes/configurations we draw an arrow if a direct transition is possible between them.

It should be noted in Figure 5.28 an important detail: when presenting a configuration of the system where there is a marked spike, it is no longer sufficient to indicate only the number of spikes and the open status of neurons, but we also

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

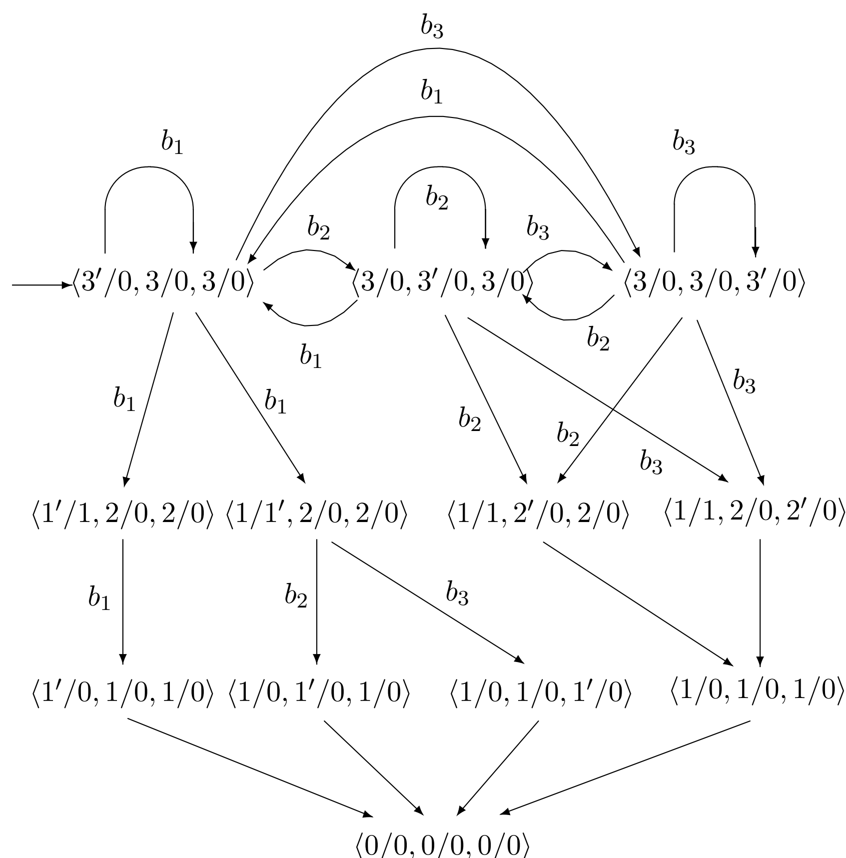


Figure 5.28: The transition diagram of system Π_1

have to indicate the place of the marked spike (if the system still contains such a spike). This is done by priming either the number of spikes from the neuron where the marked spike is, or by priming the number of steps until the neuron is open, in the case when a marked spike was produced by means of a spiking rule with delay.

In Figure 5.28 we have also indicated on the arrows the symbol introduced in the trace string by that arrow (this is b_j, where σ_j is the neuron where the marked spike arrives in the end of this step). Thus, following the marked arrows, we can construct the language of all traces.

Let us follow on this diagram some of the possible traces. As long as neuron σ_1 uses the rule $a^3/a^2 \to a; 0$, the marked spike circulates among neurons $\sigma_1, \sigma_2, \sigma_3$ and the computation continues. Note that the marked spike can be consumed or not; in the first case it moves to one of the partner neurons, non-deterministically chosen, in the latter case it remains in the neuron where it is placed.

When neuron σ_1 uses the rule $a^3/a^2 \to a; 1$, then the computation passes to the halting phase – in the diagram from Figure 5.28, we leave the upper level and we pass to the next level of configurations. If the marked spike was in neuron σ_1, it is or not consumed; this is the case when reaching the configurations $\langle 1'/1, 2/0, 2/0\rangle$, $\langle 1/1', 2/0, 2/0\rangle$: all neurons consume two spikes, but neurons σ_2 and σ_3 exchange

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

one spike, hence they end the step with two spikes inside; neuron σ_1 has only one spike inside (it is closed, does not accept spikes from neurons σ_2 and σ_3) and one ready to be emitted, and either one of them can be the marked one.

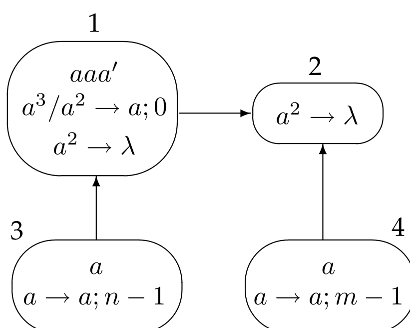


Figure 5.29: The initial configuration of system Π_2

In each case, after two more steps the computation halts with no spike in the system.

Thus, the traces start with an arbitrary string x over $\{b_1, b_2, b_3\}$, and end with one of the strings $b_1b_1, b_1b_2, b_1b_3, b_2, b_3$, depending on the last symbol of x.

We continue with a simpler example, the system Π_2 given in Figure 5.29. It generates the language $T(\Pi_2) = \{b_1^n, b_2^m\}$, for $n, m \geq 1$. In the first step, neuron σ_1 consumes or does not consume the marked spike, thus keeping it inside or sending it to neuron σ_2. One spike remains in neuron σ_1 and one is placed in neuron σ_2. Simultaneously, neurons σ_3 and σ_4 fire, and they spike after $n-1$ and $m-1$ steps, respectively. Thus, in steps n and m, neurons σ_1 and σ_2, respectively, receive one more spike, which is forgotten in the next step together with the spike existing there.

Note that n and m can be equal or different.

5.4.2 The Power of SN P Systems as Trace Generators

The following inclusions are direct consequences of the definitions:

Lemma 5.4.1 $TSNP_m(rule_k, cons_p, forg_q) \subseteq TSNP_{m'}(rule_{k'}, cons_{p'}, forg_{q'}) \subseteq TSNP_*(rule_*, cons_*, forg_*) \subseteq RE$, *for all* $1 \leq m \leq m'$, $1 \leq k \leq k'$, $1 \leq p \leq p'$, $1 \leq q \leq q'$.

We pass now to investigating the relationship with the families of languages from Chomsky hierarchy, starting with a counterexample result.

Theorem 5.4.1 *There are singleton languages not in* $TSNP_*(rule_*, cons_*, forg_*)$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Proof Let us consider the language $L = \{b_1b_2b_1b_3\}$. In order to generate it, at least three neurons should be used, with labels 1, 2, 3. Moreover, the synapses $(1, 2), (2, 1), (1, 3)$ are necessary. The existence of the synapses $(1, 2), (2, 1)$ makes possible also the trace $b_1b_2b_1b_2$: the marked spike exists after the third step and it can go non-deterministically to each neuron σ_2 and σ_3. Thus, $T(\Pi)$ cannot be a singleton, for an SN P system Π such that $b_1b_2b_1b_3 \in T(\Pi)$. □

Clearly, this reasoning can be applied to any string of the form $w = w_1b_ib_jw_2b_ib_kw_3$ with $j \neq k$, and to any language which contains a string like w but not also the string $w_1b_ib_jw_2b_ib_jw_3$.

Let us mention now a result already proved by the considerations related to the first example from the previous section (this introduces another complexity parameter, of a dynamical nature: the number of spikes present in the neurons during a computation).

Theorem 5.4.2 *The family of trace languages generated by SN P systems by means of computations with a bounded number of spikes present in their neurons is strictly included in the family of regular languages.*

Proof The inclusion follows from the fact that the transition diagram associated with the computations of an SN P system which use a bounded number of spikes is finite and can be interpreted as the transition diagram of a finite automaton, as already done in Section 5.4.1. The fact that the inclusion is proper is a consequence of Theorem 5.4.1. □

However, modulo some simple squeezing operations, SN P systems with a bounded number of spikes inside can generate any regular language:

Theorem 5.4.3 *For each language $L \subseteq V^*, L \in REG$ there is an SN P system Π such that each neuron from any computation of Π contains a bounded number of spikes, and $L = h(\partial_c^r(T(\Pi))$ for some coding h, and symbol c not in V; actually, $T(\Pi) \in TSNP_{m_L}(rule_{k_L}, cons_{p_L}, forg_0)$, for some constants m_L, k_L, p_L depending on language L.*

Proof Let us assume that $V = \{b_1, b_2, \dots, b_n\}$, and take a regular grammar $G = (N, V, S, P)$ generating the language L. Without loss of the generality, we may assume that each rule $A \to b_iB$ from P has $A \neq B$. If this is not the case with the initial set of rules, then we proceed as follows. Take a rule $A \to b_iA$; we consider a new nonterminal, A', and we replace the rule $A \to b_iA$ with $A \to b_iA'$, then we also add the rule $A' \to b_iA$ as well as all rules $A' \to b_jB$ for each $A \to b_jB \in P$. Let us continue to denote by G the obtained grammar. It is clear that this change does not modify the language generated by G, but it diminishes by one the number of rules having the same nonterminal in both sides. Continuing in this way, we eliminate all rules of this form.

Assume that G contains k rules of the form $A \to b_iB$ (from the previous discussion, we know that $A \neq B$). We construct the SN P system Π as follows.

The set of neurons is the following:

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

1. $\sigma_{\langle S\rangle} = (1', \{a \to a; 0\})$,
2. $\sigma_{\langle i,B\rangle} = (0, \{a^j \to a; 0 \mid 1 \leq j \leq k-1\})$, for each rule $A \to b_i B$ of G,
3. $\sigma_{\langle i\rangle} = (0, \{a^j \to a; 0 \mid 1 \leq j \leq k-1\}$, for each rule $A \to b_i$ of G,
4. $\sigma_f = (0, \{a^j \to a; 0 \mid 1 \leq j \leq k-1\})$,
5. $\sigma_{c_i} = (0, \{a \to a; 0\})$, for $i = 1, 2, \ldots, k$,
6. $\sigma_1 = (2, \{a^2/a \to a; 0,\ \ a^2/a \to a; 1\})$,
7. $\sigma_2 = (2, \{a^2/a \to a; 0,\ \ a \to \lambda\})$,
8. $\sigma_3 = (0, \{a \to a; 0,\ \ a^2 \to \lambda\})$.

Among these neurons, we take the following synapses:

$$\begin{array}{rcl} syn & = & \{(\langle S\rangle, \langle i, A\rangle) \mid S \to b_i A \in P\} \\ & \cup & \{(\langle i, A\rangle, \langle j, B\rangle \mid A \to b_j B \in P, 1 \leq i \leq n\} \\ & \cup & \{(\langle i, A\rangle, \langle j\rangle) \mid A \to b_j \in P, 1 \leq i \leq n\} \\ & \cup & \{(\langle S\rangle, \langle j\rangle) \mid S \to b_j \in P\} \\ & \cup & \{(\langle i\rangle, f) \mid 1 \leq i \leq n\} \\ & \cup & \{(1,2), (2,1), (1,3), (2,3)\} \cup \{(3, c_i) \mid 1 \leq i \leq k\} \\ & \cup & \{(c_i, \langle j, A\rangle),\ (c_i, \langle j\rangle \mid 1 \leq i \leq k, 1 \leq j \leq n, A \in N\}. \end{array}$$

We start with a single spike in the system, marked, placed in neuron $\sigma_{\langle S\rangle}$; the input neuron spikes in the first step.

The neurons $\sigma_1, \sigma_2, \sigma_3, \sigma_f, \sigma_{c_1}, \ldots, \sigma_{c_k}$ will be discussed separately below.

Among neurons with labels of the form $\langle S\rangle, \langle i, A\rangle$, and $\langle j\rangle$, we have synapses only if the labels of these neurons are linked by a rule of the grammar. If there are several rules of the form $A \to b_{i_1} B_1, \ldots, A \to b_{i_r} B_r$, then the spike emitted by each neuron $\sigma_{\langle j,A\rangle}, 1 \leq j \leq n$, goes to all neurons $\sigma_{\langle i_t,B_t\rangle}, 1 \leq t \leq r$ – and the marked spike can take any of these choices. Conversely, we can have several (actually, at most $k-1$) synapses coming to the same neuron, if more rules have the same right hand member. Thus, in any neuron we can collect at most $k-1$ spikes in a step. All of them are immediately consumed, hence never the number of spikes from any neuron becomes greater than $k-1$. Clearly, the marked spike can follow a derivation in G.

Because the grammar G can contain cycles (for instance, pairs of rules of the form $A \to b_i B, B \to b_i A$), the system Π can contain pairs of self-sustaining neurons, hence the computation in Π could not stop, even if a derivation in G was correctly simulated (this is due to the fact that the spikes leaving a neuron are replicated to all neurons to which we have a synapse).

In order to prevent such "wrong" evolutions, we use the "halting module", composed of the neurons $\sigma_1, \sigma_2, \sigma_3, \sigma_f, \sigma_{c_1}, \ldots, \sigma_{c_k}$: neurons σ_1, σ_2 exchange spikes

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

for an arbitrary number of steps; when σ_1 uses the rule $a^2/a \to a; 1$, their interplay stops, but neuron σ_3 fires and sends a spike to all neurons $\sigma_{c_i}, 1 \leq i \leq k$. These neurons will send spikes to all neurons $\sigma_{\langle j,A\rangle}$, $\sigma_{\langle j\rangle}$, and $\sigma_{\langle S\rangle}$ of the system, thus halting their evolution (they do not have rules for handling more than $k-1$ spikes). The computation halts.

What is not ensured yet by the construction of Π is the fact that the computation is not halted as above before finishing the derivation in G which we want to simulate, that is, with the marked spike present inside the system. This aspect is handled by the squeezing mechanism: we take the alphabet

$$U = \{b_{\langle i,A\rangle} \mid 1 \leq i \leq n, A \in N\} \cup \{b_{\langle i\rangle} \mid 1 \leq i \leq n\},$$

the symbol $c = b_f$, and the coding $h : U^* \longrightarrow V^*$ defined by

$$h(b_{\langle i,A\rangle}) = b_i,\ 1 \leq i \leq n, A \in N,$$
$$h(b_{\langle i\rangle}) = b_i,\ 1 \leq i \leq n.$$

The right derivative with respect to b_f selects from $T(\Pi)$ only those traces for which the marked spike reaches the "final" neuron σ_f, which is accessible only from a "terminal" neuron $\sigma_{\langle i\rangle}$, hence the marked spike has followed a complete derivation in G. Then, the coding h renames the symbols, thus delivering exactly the strings of $L(G)$.

Clearly, we can determine the constants m_L, k_L, p_L as in the statement of the theorem, depending on the size of grammar G, and this observation completes the proof. □

A related result can be obtained by slightly changing the previous proof.

Theorem 5.4.4 *For each regular language $L \subseteq V^*$ there is an SN P system Π such that each neuron from any computation of Π contains a bounded number of spikes, and $L = h(T(\Pi) \cap U_1^* U_2)$, for some coding h and alphabets U_1, U_2.*

As expected, also non-regular languages can be generated – of course, by using systems whose computations are allowed to use an unbounded number of spikes in the neurons. We skip the proof of the next result.

Theorem 5.4.5 $TSNP_{12}(rule_2, cons_2, forg_1) - REG \neq \emptyset$.

Actually, languages "far from being regular" can be also generated:

Theorem 5.4.6 *Every unary language $L \in RE$ can be written in the form $L = h(L') = (d_1^* \backslash L') \cap d_2^*$, where $L' \in TSNP_*(rule_2, cons_3, forg_3)$, and h is a projection.*

Proof This result is a consequence of the fact that SN P systems can simulate register machines. Specifically, as proved in Section 5.2.2, starting from a register machine M, we construct an SN P system Π which halts its computation with $2n$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

spikes in a specified neuron σ_{out} if and only if n can be generated by the register machine M; in the halting moment, a neuron σ_{l_h} of Π associated with the label of the halting instruction of M gets two spikes and fires. The neuron σ_{out} contains no rule used in the simulation of M (the corresponding register is only incremented, but never decremented – see the details of the construction from Section 5.2.2).

Now, consider a language $L \subseteq b_2^*, L \in RE$. There is a register machine M such that $n \in N(M)$ if and only if $b_2^n \in L$. Starting from such a machine M, we construct the system Π as in Section 5.2.2, having the properties described above. We append to the system Π six more neurons, as indicated in Figure 5.30. There is a marked spike in neuron σ_1, and it will stay here during all the simulation of M. In the moment when neuron σ_{l_h} of Π spikes, its spike goes both to neuron σ_{out} and to neuron σ_1.

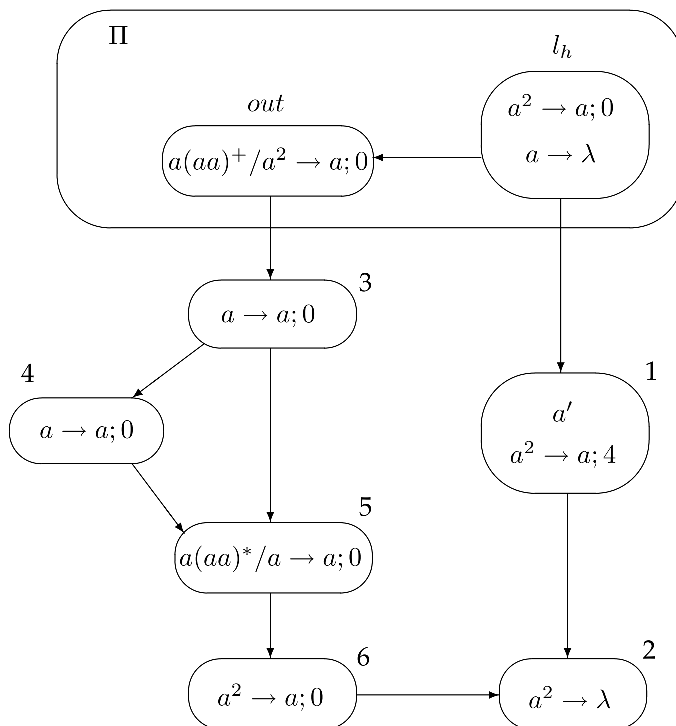


Figure 5.30: The SN P system from the proof of Theorem 5.4.6

Neurons $\sigma_3, \sigma_4, \sigma_5, \sigma_6$ send a spike to neuron σ_2 only when neuron σ_{out} has finished its work (this happens after n steps of using the rule $a(aa)^+/a^2 \to a; 0$, for $2n$ being the contents of neuron σ_{out} in the moment when neuron σ_{l_h} spikes).

The marked spike leaves neuron σ_1 four steps after using the rule $a^2 \to a; 4$, hence five steps after the spiking of neuron σ_{l_h}. This means that the marked spike waits in neuron σ_2 exactly n steps. When the spike of neuron σ_6 reaches neuron σ_2, the two spikes present here, the marked one included, are forgotten.

Thus, the traces of the marked spike are of the form $b_1^r b_2^n$, for some $r \geq 1$ and $n \in N(M)$. By means of the left derivative with the regular language b_1^* we

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

can remove prefixes of the form b_1^k and by means of the intersection with b_2^* we ensure that the maximal prefix of this form is removed. Similarly, the projection $h : \{b_1, b_2\}^* \longrightarrow \{b_1, b_2\}^*$ defined by $h(b_1) = \lambda$, $h(b_2) = b_2$, removes all occurrences of b_1. Consequently, $L = (b_1^* \backslash T(\Pi)) \cap b_2^* = h(T(\Pi))$.

The system Π from Section 5.2.2 (Theorem 5.2.2 there) has the rule complexity described by $rule_2, cons_3, forg_3$; one sees that the construction from Figure 5.30 does not increase these parameters. □

In what concerns the number of neurons, we do not have a bound for the language L' from the previous theorem.

Corollary 5.4.1 *Each family $TSNP_*(reg_k, cons_p, forg_q)$ with $k \geq 2, p \geq 3, q \geq 3$, is incomparable with each family of languages FL which (i) contains the singleton languages, (ii) is closed under left quotients with regular languages and intersection with regular languages, or under projections, and (iii) does not contain all unary recursively enumerable languages.*

5.4.3 Remarks and Further Research

We have considered here the possibility of generating a language by means of a spiking neural P system by following the traces of a distinguished spike during a halting computation. The power of such devices seems to be rather large – Theorem 5.4.6 – but we do not have a precise characterization of the obtained families of languages. Theorem 5.4.6 also raises the natural question whether RE languages over arbitrary alphabets can be represented starting from trace languages of SN P systems.

The main difficulty in handling the trace languages of SN P systems is, in the previous setup, the non-determinism of the marked spike evolution. A possible way to better control the marked spike is to consider spiking rules of the forms $E'/a^c \rightarrow a; d$ and $E/a'a^c \rightarrow a'; d$, with the meaning that the prime indicates whether or not the rule consumes the marked spike: this is not the case when the prime is attached to E, but this happens if the prime marks one of the consumed spikes, and hence also the produced spike. This removes the non-determinism induced by the use of the marked spike when applying the rules, but still non-determinism remains in what concerns the choice of synapses toward neighboring neurons (and this was the basis of the easy counterexample from Theorem 5.4.1). How also this non-determinism can be removed remains as a research topic.

5.5 Spiking Neural P Systems with Extended Rules

In this section we consider SN P systems with rules allowed to introduce zero, one, or more spikes at the same time – they are called *extended* SN P systems. The

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

(mathematical) motivation comes both from constructing small universal systems and from generating strings.

An *extended spiking neural P system* of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

where all components are as above, but the rules of neurons are of the form $E/a^c \rightarrow a^p$, where E is a regular expression over a and $c \geq 1, p \geq 0$, with the restriction $c \geq p$.

A rule $E/a^c \rightarrow a^p$ is applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule can *fire*, and its application means consuming (removing) c spikes (thus only $k - c$ remain in σ_i) and producing p spikes, which will exit immediately the neuron.

Note that we do not consider here a delay between firing and spiking (i.e., rules of the form $E/a^c \rightarrow a^p; d$, with $d \geq 0$), because we do not need this feature in the proofs below, but such a delay can be introduced in the usual way. (As a consequence, here the neurons are always open.) Moreover, because $p = 0$ means producing no spike, rules as above also cover the case of forgetting rules, which are now of a generalized form, with a regular expression controlling their application.

If a rule $E/a^c \rightarrow a^p$ has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a^p$. If all rules $E/a^c \rightarrow a^p$ have $L(E) = \{a^c\}$, then we say that the system is *finite*.

The spikes emitted by a neuron σ_i go to all neurons σ_j such that $(i, j) \in syn$, i.e., if σ_i has used a rule $E/a^c \rightarrow a^p$, then each neuron σ_j receives p spikes.

As usual, if several rules can be used at the same time, then the one to be applied is chosen non-deterministically.

During the computation, a configuration of the system is described by the number of spikes present in each neuron; thus, the initial configuration is described by the numbers $n_1, n_2, \ldots, n_m$.

We denote by $N_2(\Pi)$ the set of numbers generated by an extended SN P system in the form of the number of steps between the first two steps of a computation when spikes are emitted into environment, and by $Spik_2SN^eP_m(rule_k, cons_p, prod_q)$ the family of sets $N_2(\Pi)$ generated by SN P systems with at most m neurons, at most k rules in each neuron, consuming at most p and producing at most q spikes. Any of these parameters is replaced by $*$ if it is not bounded. (The superscript e points out the fact that we work with *extended* rules.)

5.5.1 Extended SN P Systems as Number Generators

Because non-extended SN P systems – without delay; see [35] – are already computationally universal, this result is directly valid also for extended systems. However, the construction on which the proof is based is much simpler in the

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

extended case and it is also instructive for the way the small universal systems are found, that is why we briefly present it.

Theorem 5.5.1 $NRE = Spik_2SN^eP_*(rule_4, cons_5, prod_2)$.

Proof The proof of the similar result from Section 5.2.2 is based on constructing an SN P system Π which simulates a given register machine M. The idea is that each register r has associated a neuron σ_r, with the value n of the register represented by $2n$ spikes in neuron σ_r. Also, each label of M has a neuron in Π, which is "activated" when receiving two spikes. We do not recall other details from the proof, and we pass directly to presenting – in Figures 5.31, 5.32, and 5.33 – modules for simulating the ADD and the SUB instructions of M, as well as an OUTPUT module, in the case of using extended rules.

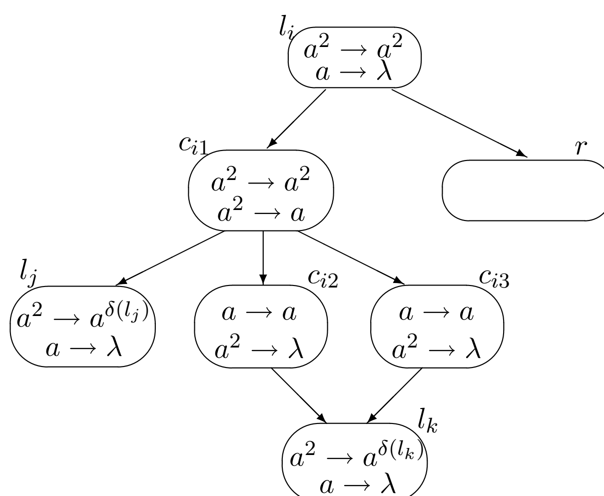


Figure 5.31: Module ADD, for simulating an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$

Because the neurons associated with labels of ADD and SUB instructions have to produce different numbers of spikes, in the neurons associated with "output" labels of instructions we have written the rules in the form $a^2 \to a^{\delta(l)}$, with $\delta(l) = 1$ for l being the label of a SUB instruction and $\delta(l) = 2$ if l is the label of an ADD instruction.

Because l_i precisely identifies the instruction, the neurons $c_{i\alpha}$ are distinct for distinct instructions. However, an interference between SUB modules appears in the case of instructions SUB which operate on the same register r: synapses $(r, c_{is}), (r, c_{i's}), s = 4, 5$, exist for different instructions $l_i : (\texttt{SUB}(r), l_j, l_k)$, $l_{i'} : (\texttt{SUB}(r), l_{j'}, l_{k'})$. Neurons $\sigma_{c_{i's}}, \sigma_{c_{i'5}}$ receive 1 or 2 spikes from σ_r even when simulating the instruction with label l_i, but they are immediately forgotten (this is the role of rules $a \to \lambda$, $a^2 \to \lambda$ from neurons $\sigma_{c_{i4}}, \sigma_{c_{i5}}$ from Figure 5.32).

The task of checking the functioning of the modules from Figures 5.31, 5.32, 5.33 is left to the reader. □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

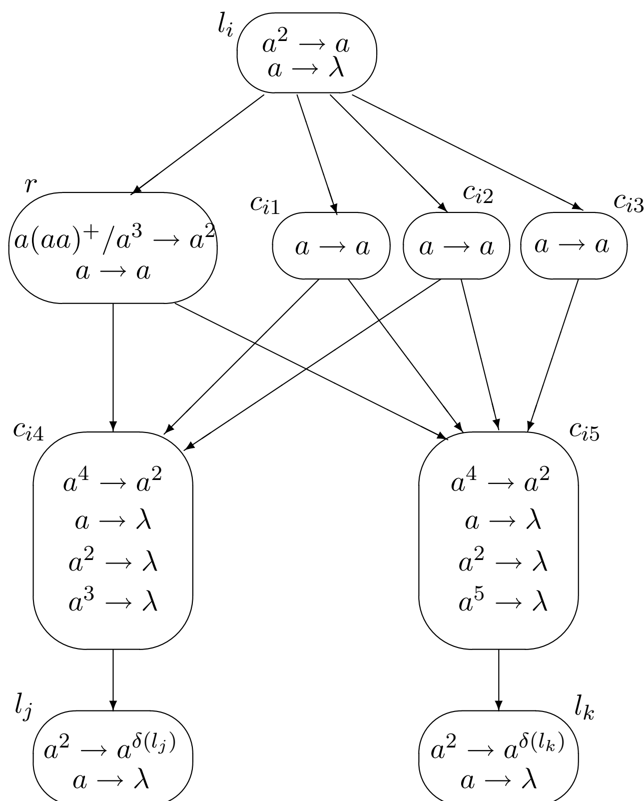


Figure 5.32: Module SUB, for simulating an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$

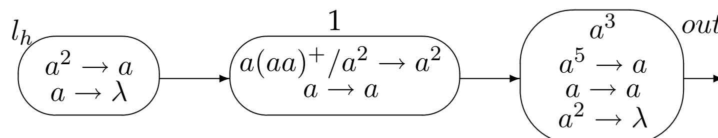


Figure 5.33: Module OUTPUT

5.5.2 Languages in the Restricted Case

We pass now to considering extended SN P systems as language generators, starting with the restricted case, when the system outputs a symbol in each computation step. Specifically, with a step when a system Π sends out $i \geq i$ spikes we associate the symbol b_i (thus, when no spike is emitted, we produce the symbol b_0). The set of strings of this type associated with all halting computations in Π is denoted by $L_{res}(\Pi)$ - with *res* pointing out the fact that in each step one symbol is produced. Another possibility, much more flexible , is to .... b_0 a "no symbol", i.e., to introduce no symbol in the steps when no spike is produced. In this case we denote the generated language by $L_\lambda(\Pi)$.

Then, by $L_\alpha SNP_m(rule_k, cons_p, prod_q)$ we denote the family of all languages $L_\alpha(\Pi)$ generated in mode $\alpha \in \{res, \lambda\}$, by extended SN P systems with at most m

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

neurons, at most k rules in each neuron, consuming at most p and producing at most q spikes.

In all considerations below, we work with the alphabet $V = \{b_1, b_2, \ldots, b_s\}$, for some $s \geq 1$. By a simple renaming of symbols, we may assume that any given language L over an alphabet with at most s symbols is a language over V. When a symbol b_0 is also used, it is supposed that $b_0 \notin V$.

A Characterization of FIN

As we have seen before, SN P systems with standard rules cannot generate all finite languages, but extended rules help in this respect.

Lemma 5.5.1 $L_\alpha SN^e P_1(rule_*, cons_*, prod_*) \subseteq FIN$, $\alpha \in \{res, \lambda\}$.

Proof In each step, the number of spikes present in a system with only one neuron decreases by at least one, hence any computation lasts at most as many steps as the number of spikes present in the system at the beginning. Thus, the generated strings have a bounded length. □

Lemma 5.5.2 $FIN \subseteq L_\alpha SN^e P_1(rule_*, cons_*, prod_*)$, $\alpha \in \{res, \lambda\}$.

Proof Let $L = \{x_1, x_2, \ldots, x_n\} \subseteq V^*$, $n \geq 1$, be a finite language, and let $x_i = x_{i,1} \ldots x_{i,r_i}$ for $x_{i,j} \in V, 1 \leq i \leq n, 1 \leq j \leq r_i = |x_i|$. Denote $l = \max\{r_i \mid 1 \leq i \leq n\}$. For $b \in V$, define $index(b) = i$ if $b = b_i$. Define $\alpha_j = ls \sum_{i=1}^{j} |x_i|$, for all $1 \leq j \leq n$.

An SN P system that generates L is shown in Figure 5.51.

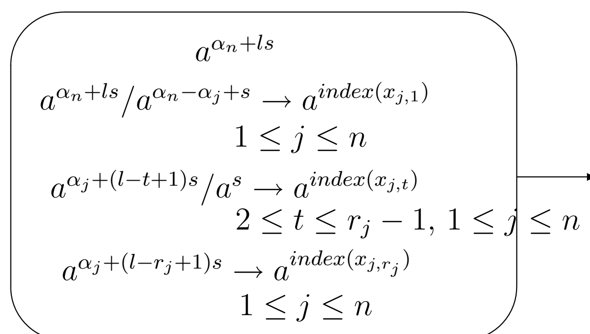


Figure 5.34: An extended SN P system generating a finite language

Initially, only a rule $a^{\alpha_n+ls}/a^{\alpha_n-\alpha_j+s} \to a^{index(x_{j,1})}$ can be used, and in this way we non-deterministically chose the string x_j to generate. This rule outputs the necessary number of spikes for $x_{j,1}$. Then, because $\alpha_j + (l-1)s$ spikes remain in the neuron, we have to continue with rules $a^{\alpha_j+(l-t+1)s}/a^s \to a^{index(x_{j,t})}$, for $t = 2$, and then for the respective $t = 3, 4, \ldots, r_j - 1$; in this way we introduce $x_{j,t}$, for all

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$t = 2, 3, \ldots, r_j - 1$. In the end, the rule $a^{\alpha_j+(l-r_j+1)s} \to a^{index(x_{j,r_j})}$ is used, which produces x_{j,r_j} and concludes the computation.

It is easy to see that the rules which are used in the generation of a string x_j cannot be used in the generation of a string x_k with $k \neq j$. Also, in each rule the number of spikes consumed is not less than the number of spikes produced. The system Π never outputs zero spikes, hence $L_{res}(\Pi) = L_\lambda(\Pi) = L$. □

Theorem 5.5.2 $FIN = L_\alpha SN^eP_1(rule_*, cons_*, prod_*), \alpha \in \{res, \lambda\}$.

This characterization is sharp in what concerns the number of neurons, because of the following result:

Proposition 5.5.1 $L_\alpha SN^eP_2(rule_2, cons_3, prod_3) - FIN \neq \emptyset$, $\alpha \in \{res, \lambda\}$.

Proof The SN P system Π from Figure 5.35 generates the infinite language $L_{res}(\Pi) = L_\lambda(\Pi) = b_3^*b_1\{b_1, b_3\}$. □

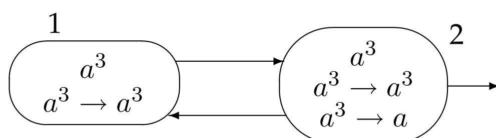


Figure 5.35: An extended SN P system generating an infinite language

Representations of Regular Languages

Such representations are obtained in Section 5.3 starting from languages of the form $L_{bin}(\Pi)$, but in the extended SN P systems, regular languages can be represented in an easier and more direct way.

Theorem 5.5.3 *If* $L \subseteq V^+$, $L \in REG$, *then* $\{b_0\}L \in L_{res}SN^eP_4(rule_*, cons_*, prod_*)$.

Proof Consider a regular grammar $G = (N, V, S, P)$ such that $L = L(G)$, where $N = \{A_1, A_2, \ldots, A_n\}$, $n \geq 1$, $S = A_n$, and the rules in P are of the forms $A_i \to b_kA_j$, $A_i \to b_k$, $1 \leq i, j \leq n$, $1 \leq k \leq s$.

Then $\{b_0\}L$ can be generated by the SN P system shown in Figure 5.36.

In each step, neurons σ_1 and σ_2 will send $n + s$ spikes to neuron σ_3, provided that they receive spikes from neuron σ_3. Neuron σ_3 fires in the first step by a rule $a^{2n+s}/a^{2n-j+s} \to a^k$ (or $a^{2n+s} \to a^k$) associated with a rule $A_n \to b_kA_j$ (or $A_n \to b_k$) from P, produces k spikes and receives $n + s$ spikes from neuron σ_2. In the meantime neuron σ_4 does not spike, hence it produces the symbol b_0, and receives spikes from neuron σ_3, therefore in the second step it generates the first symbol of the string.

ISBN:978-84-691-9877-3/DL:T-127-2009

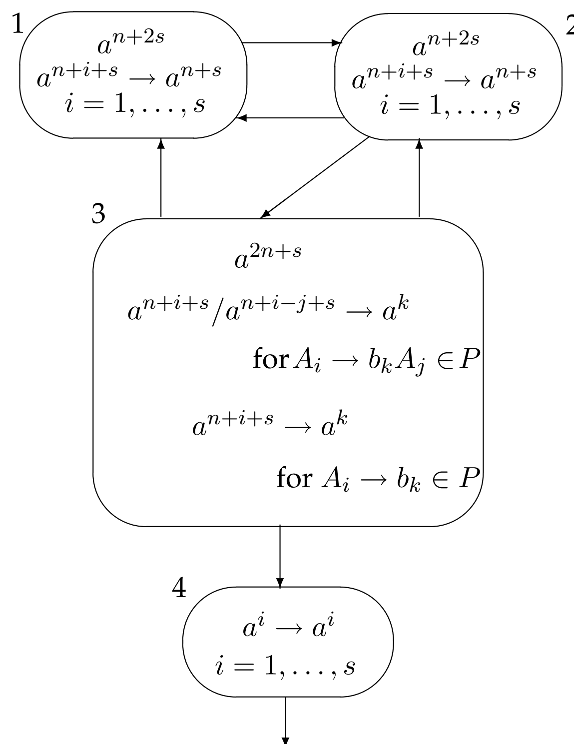


Figure 5.36: The SN P system from the proof of Theorem 5.5.3

Assume in some step t, the rule $a^{n+i+s}/a^{n+i-j+s} \to a^k$, for $A_i \to b_k A_j$, or $a^{n+i+s} \to a^k$, for $A_i \to b_k$, is used, for some $1 \leq i \leq n$, and $n+s$ spikes are received from neuron σ_2.

If the first rule is used, then k spikes are produced, $n+i-j+s$ spikes are consumed and j spikes remain in neuron σ_3. Then in step $t+1$, we have $n+j+s$ spikes in neuron σ_3, and a rule for $A_j \to b_k A_l$ or $A_j \to b_k$ can be used. In step $t+1$ neuron σ_3 also receives $n+m$ spikes from σ_2. In this way, the computation continues, unless the second rule is used.

If the second rule is used, then k spikes are produced, all spikes are consumed, and $n+m$ spikes are received in neuron σ_3. Then, in the next time step, neuron σ_3 receives $n+m$ spikes, but no rule can be used, so no spike is produced. At the same time, neuron σ_4 fires using spikes received from neuron σ_3 in the previous step, and then the computation halts.

In this way, all the strings in $\{b_0\}L$ can be generated. □

Corollary 5.5.1 *Every language $L \in REG, L \subseteq V^+$, can be written in the form $L = \partial^l_{b_0}(L')$ for some $L' \in L_{res}SN^eP_4(rule_*, cons_*, prod_*)$.*

One neuron in the previous representation can be saved, by adding the extra symbol in the right hand end of the string.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Theorem 5.5.4 *If* $L \subseteq V^+$, $L \in REG$, *then* $L\{b_0\} \in L_{res}SN^eP_3(rule_*, cons_*, prod_*)$.

Proof The proof is based on a construction similar to the one from the proof of Theorem 5.5.3. Specifically, starting from a regular grammar G as above, we construct a system Π as in Figure 5.37, for which we have $L_{res}(\Pi) = L\{b_0\}$. We leave the task to check this assertion to the reader. □

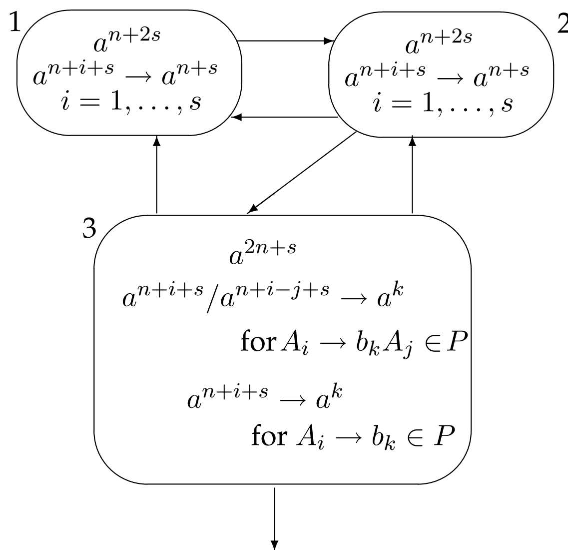


Figure 5.37: The SN P system for the proof of Theorem 5.5.4

Corollary 5.5.2 *Every language* $L \in REG, L \subseteq V^+$, *can be written in the form* $L = \partial^r_{b_0}(L')$ *for some* $L' \in L_{res}SN^eP_3(rule_*, cons_*, prod_*)$.

Going Beyond REG

We do not know whether the additional symbol b_0 can be avoided in the previous theorems (hence whether the regular languages can be directly generated by SN P systems in the restricted way), but such a result is not valid for the family of minimal linear languages (generated by linear grammars with only one nonterminal symbol).

Lemma 5.5.3 *The number of configurations reachable after* n *steps by an extended SN P system of degree* m *is bounded by a polynomial* $g(n)$ *of degree* m.

Proof Let us consider an extended SN P system $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$ of degree m, let n_0 be the total number of spikes present in the initial configuration of Π, and denote $\alpha = \max\{p \mid E/a^c \to a^p \in R_i, 1 \leq i \leq m\}$ (the maximal number of spikes produced by any of the rules of Π). In each step of a computation, each neuron σ_i consumes some c spikes and produces $p \leq c$ spikes; these spikes are

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

sent to all neurons σ_j such that $(i, j) \in syn$. There are at most $m - 1$ synapses $(i, j) \in syn$, hence the p spikes produced by neuron σ_i are replicated in at most $p(m - 1)$ spikes. We have $p(m - 1) \leq \alpha(m - 1)$. Each neuron can do the same, hence the maximal number of spikes produced in one step is at most $\alpha(m - 1)m$. In n consecutive steps, this means at most $\alpha(m - 1)mn$ spikes. Adding the initial n_0 spikes, this means that after any computation of n steps we have at most $n_0 + \alpha(m - 1)mn$ spikes in Π. These spikes can be distributed in the m neurons in less that $(n_0 + \alpha(m - 1)mn)^m$ different ways. This is a polynomial of degree m in n (α is a constant) which bounds from above the number of possible configurations obtained after computations of length n in Π. □

Theorem 5.5.5 *If $f : V^+ \longrightarrow V^+$ is an injective function, $card(V) \geq 2$, then there is no extended SN P system Π such that $L_f(V) = \{x\, f(x) \mid x \in V^+\} = L_{res}(\Pi)$.*

Proof Assume that there is an extended SN P system Π of degree m such that $L_{res}(\Pi) = L_f(V)$ for some f and V as in the statement of the theorem. According to the previous lemma, there are only polynomially many configurations of Π which can be reached after n steps. However, there are $card(V)^n \geq 2^n$ strings of length n in V^+. Therefore, for large enough n there are two strings $w_1, w_2 \in V^+, w_1 \neq w_2$, such that after n steps the system Π reaches the same configuration when generating the strings $w_1\, f(w_1)$ and $w_2\, f(w_2)$, hence after step n the system can continue any of the two computations. This means that also the strings $w_1\, f(w_2)$ and $w_2\, f(w_1)$ are in $L_{res}(\Pi)$. Due to the injectivity of f and the definition of $L_f(V)$ such strings are not in $L_f(V)$, hence the equality $L_f(V) = L_{res}(\Pi)$ is contradictory. □

Corollary 5.5.3 *The following languages are not in $L_{res}SN^eP_*(rule_*, cons_*, prod_*)$ (in all cases, $card(V) = k \geq 2$):*

$$L_1 = \{x\, mi(x) \mid x \in V^+\},$$
$$L_2 = \{xx \mid x \in V^+\},$$
$$L_3 = \{x\, c^{val_k(x)} \mid x \in V^+\}, c \notin V.$$

Note that language L_1 above is a non-regular minimal linear one, L_2 is context-sensitive non-context-free, and L_3 is non-semilinear. In all cases, we can also add a fixed tail of any length (e.g., considering $L_1' = \{x\, mi(x)z \mid x \in V^+\}$, where $z \in V^+$ is a given string), and the conclusion is the same – hence a result like that in Theorem 5.5.4 cannot be extended to minimal linear languages.

5.5.3 Languages in the Non-Restricted Case

As expected, the possibility of having intermediate steps when no output is produced is helpful, because this provides intervals for internal computations. In this way, we can get rid of the operations used in the previous sections when dealing with regular and with recursively enumerable languages.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Relationships with REG

Lemma 5.5.4 $L_\lambda SN^eP_2(rule_*, cons_*, prod_*) \subseteq REG$.

Proof In a system with two neurons, the number of spikes from the system can remain the same after a step, but it cannot increase: the neurons can consume the same number of spikes as they produce, and they can send to each other the produced spikes. Therefore, the number of spikes in the system is bounded by the number of spikes present at the beginning. This means that the system can pass through a finite number of configurations and these configurations can control the evolution of the system like states in a finite automaton. Consequently, the generated language is regular (see similar reasonings, with more technical details, in the previous sections). □

Lemma 5.5.5 $REG \subseteq L_\lambda SN^eP_3(rule_*, cons_*, prod_*)$.

Proof For the SN P system Π constructed in the proof of Theorem 5.5.4 (Figure 5.37) we have $L_\lambda(\Pi) = L(G)$. □

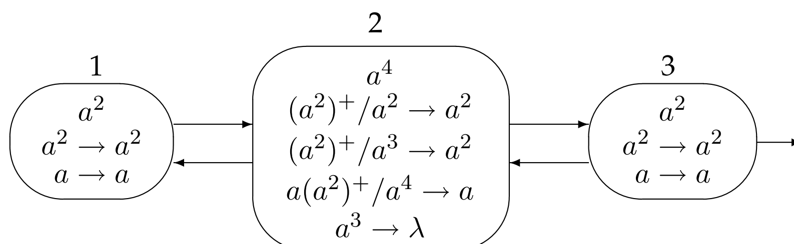


Figure 5.38: An SN P system generating a non-regular language

This last inclusion is proper:

Proposition 5.5.2 $L_\lambda SN^eP_3(rule_4, cons_4, prod_2) - REG \neq \emptyset$.

Proof The SN P system Π from Figure 5.38 generates the language $L_\lambda(\Pi) = \{b_2^n b_1^{n+1} \mid n \geq 1\}$. Indeed, for a number $n \geq 0$ of steps, neuron σ_2 consumes two spikes by using the rule $(a^2)^+/a^2 \to a^2$ and receives four from the other two neurons. After changing the parity of the number of spikes (by using the rule $(a^2)^+/a^3 \to a^2$), neuron σ_2 will continue by consuming four spikes (using the rule $a(a^2)^+/a^4 \to a$) and receiving only two. When only 3 spikes remain, the computation stops (the two further spikes received by σ_2 from σ_1 and σ_3 cannot fire again neuron σ_2). □

Corollary 5.5.4 $L_\lambda SN^eP_1(rule_*, cons_*, prod_*) \subset L_\lambda SN^eP_2(rule_*, cons_*, prod_*) \subset L_\lambda SN^eP_3(rule_*, cons_*, prod_*)$, *strict inclusions.*

ISBN:978-84-691-9877-3/DL:T-127-2009

Going Beyond CF

Actually, much more complex languages can be generated by extended SN P systems with three neurons.

Theorem 5.5.6 *The family* $L_\lambda SN^e P_3(rule_3, cons_6, prod_4)$ *contains non-semilinear languages.*

Proof The system Π from Figure 5.39 generates the language

$$L_\lambda(\Pi) = \{b_4^2 b_2 b_4^{2^2} b_2 \dots b_4^{2^n} b_2 \mid n \geq 1\}.$$

We start with $2 + 4 \cdot 2^0$ spikes in neuron σ_1. When moved from neuron σ_1 to neuron σ_3, the number of spikes is doubled, because they pass both directly from σ_1 to σ_3, and through σ_2. When all spikes are moved to σ_3, the rule $a^2 \rightarrow a$ of σ_1 should be used. With a number of spikes of the form $4m + 1$, neuron σ_3 cannot fire, but in the next step one further spike comes from σ_2, hence the first rule of σ_3 can now be applied. Using this rule, all spikes of σ_3 are moved back to σ_1 – in the last step we use the rule $a^2 \rightarrow a^2$, which makes again the first rule of σ_1 applicable.

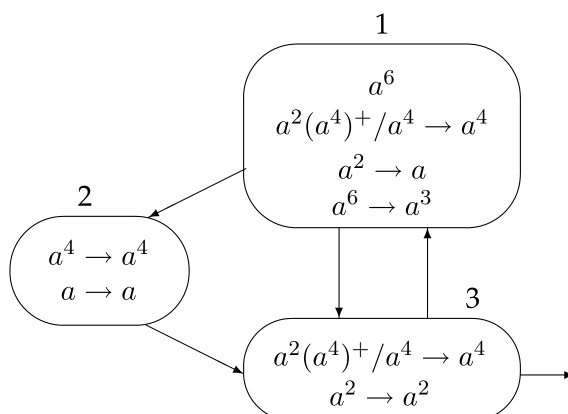


Figure 5.39: An SN P system generating a non-semilinear language

This process can be repeated any number of times. In each moment, after moving all but the last 6 spikes from neuron σ_1 to σ_3, we can also use the rule $a^6 \rightarrow a^3$ of σ_1, and this ends the computation: there is no spike in σ_1, neuron σ_2 cannot work when having 3 spikes inside, and the same with σ_3 when having $4m + 3$ spikes.

Now, one sees that σ_3 is also the output neuron and that the number of times of using the first rule of σ_3 is doubled after each move of the contents of σ_3 to σ_1. □

In this proof we made an essential use of the fact that no spike of the output neuron means no symbol introduced in the generated string. If we work

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

in the restricted case, then symbols b_0 are shuffled in the string, hence the non-semilinearity of the generated language is preserved, that is, the result also holds for the restricted case.

A Characterization of RE

If we do not bound the number of neurons, then a characterization of recursively enumerable languages is obtained.

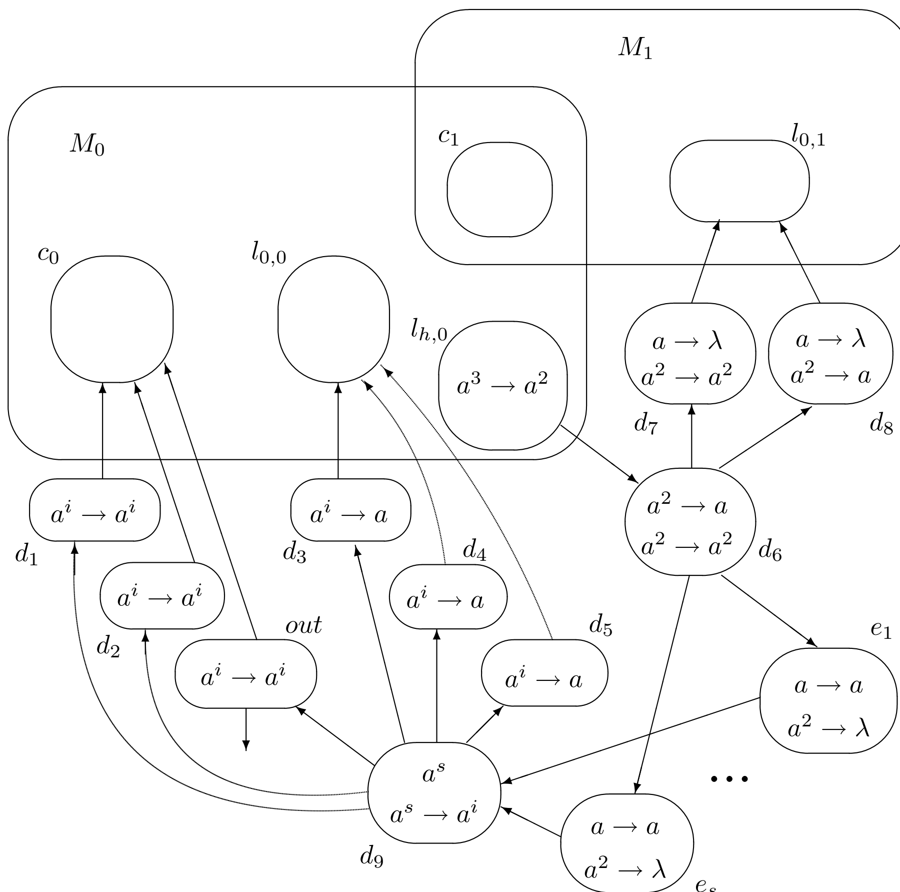


Figure 5.40: The structure of the SN P system from the proof of Lemma 5.5.6

As in Section 3.1, let us write s in front of a language family notation in order to denote the subfamily of languages over an alphabet with at most s symbols (e.g., $2RE$ denotes the family of recursively enumerable languages over alphabets with one or two symbols).

Lemma 5.5.6 $sRE \subseteq sL_\lambda SN^eP_*(rule_{s'}, cons_s, prod_s)$, *where* $s' = \max(s, 6)$ *and* $s \geq 1$.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Proof We follow here the same idea as in the proof of Theorem 5.3.9 from Section 5.3, adapted to the case of extended rules.

Take an arbitrary language $L \subseteq V^*, L \in RE$, $card(V) = s$. Obviously, $L \in RE$ if and only if $val_s(L) \in NRE$. In turn, a set of numbers is recursively enumerable if and only if it can be accepted by a deterministic register machine. Let M_1 be such a register machine, i.e., $N(M_1) = val_s(L)$.

We construct an SN P system Π performing the following operations (σ_{c_0} and σ_{c_1} are two distinguished neurons of Π, which are empty in the initial configuration):

1. Output i spikes, for some $1 \leq i \leq s$, and at the same time introduce the number i in neuron σ_{c_0}; in the construction below, a number n is represented in a neuron by storing there $3n$ spikes, hence the previous task means introducing $3i$ spikes in neuron σ_{c_0}.

2. Multiply the number stored in neuron σ_{c_1} (initially, we have here number 0) by $s+1$, then add the number from neuron σ_{c_0}; specifically, if neuron σ_{c_0} holds $3i$ spikes and neuron σ_{c_1} holds $3n$ spikes, $n \geq 0$, then we end this step with $3(n(s+1)+i)$ spikes in neuron σ_{c_1} and no spike in neuron σ_{c_0}. In the meantime, the system outputs no spike.

3. Repeat from step 2, or, non-deterministically, stop the increase of spikes from neuron σ_{c_1} and pass to the next step.

4. After the last increase of the number of spikes from neuron σ_{c_1} we have here $val_s(x)$ for a string $x \in V^+$. Start now to simulate the work of the register machine M_1 in recognizing the number $val_s(x)$. The computation halts only if this number is accepted by M_1, hence the string x produced by the system is introduced in the generated language only if $val_s(x) \in N(M_1)$.

In constructing the system Π we use the fact that a register machine can be simulated by an SN P system. Then, the multiplication by $s+1$ of the contents of neuron σ_{c_1} followed by adding a number between 1 and s is done by a computing register machine (with the numbers stored in neurons $\sigma_{c_0}, \sigma_{c_1}$ introduced in two specified registers); we denote by M_0 this machine. Thus, in our construction, also for this operation we can rely on the general way of simulating a register machine by an SN P system. All other modules of the construction (introducing a number of spikes in neuron σ_{c_0}, sending out spikes, choosing non-deterministically to end the string to generate and switching to the checking phase, etc.) are explicitly presented below.

The overall appearance of Π is given in Figure 5.40, where M_0 indicates the subsystem corresponding to the simulation of the register machine $M_0 = (m_0, H_0, l_{0,0}, l_{h,0}, I_0)$ and M_1 indicates the subsystem which simulates the register machine $M_1 = (m_1, H_1, l_{0,1}, l_{h,1}, I_1)$. Of course, we assume $H_0 \cap H_1 = \emptyset$.

We start with spikes only in neuron σ_{d_9}. We spike in the first step, non-deterministically choosing the number i of spikes to produce, hence the first letter

ISBN:978-84-691-9877-3/DL:T-127-2009

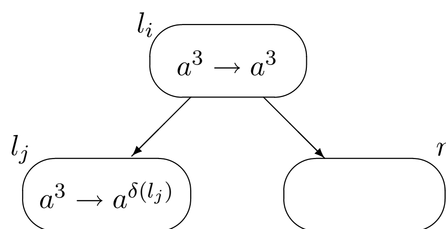


Figure 5.41: Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j)$)

b_i of the generated string. Simultaneously, i spikes are sent out by the output neuron, $3i$ spikes are sent to neuron σ_{c_0}, and three spikes are sent to neuron $\sigma_{l_{0,0}}$, thus triggering the start of a computation in M_0. The subsystem corresponding to the register machine M_0 starts to work, multiplying the value of σ_{c_1} with $s+1$ and adding i. When this process halts, neuron $\sigma_{l_{h,0}}$ is activated, and in this way two spikes are sent to neuron σ_{d_6}.

This is the neuron which non-deterministically chooses whether the string should be continued or we pass to the second phase of the computation, checking whether the produced string is accepted. In the first case, neuron σ_{d_6} uses the rule $a^2 \to a$, which makes neurons $\sigma_{e_1}, \ldots, \sigma_{e_m}$ spike; these neurons send m spikes to neuron σ_{d_9}, like in the beginning of the computation. In the latter case, one uses the rule $a^2 \to a^2$, which activates the neuron $\sigma_{l_{0,1}}$ by sending three spikes to it, thus starting the simulation of the register machine M_1. The computation stops if and only if $val_s(x)$ is accepted by M_1.

In order to complete the proof we need to show how the two register machines are simulated, using the common neuron σ_{c_1} but without mixing the computations. To this aim, we consider the modules ADD and SUB from Figures 5.41, 5.42, and 5.43. Like in Section 5.6, neurons are associated with each label of the machine (they fire if they have three spikes inside) and with each register (with $3t$ spikes representing the number t from the register); there also are additional neurons with labels c_{il} – it is important to note that all these additional neurons have distinct labels.

The simulation of an ADD instruction is easy, we just add three spikes to the respective neuron; no rule is needed in the neuron – Figure 5.41. The SUB instructions of machines M_0, M_1 are simulated by modules as in Figures 5.42 and 5.43, respectively. Note that the rules for M_0 fire for a content of the neuron σ_r described by the regular expression $(a^3)^+a$ and the rules for M_1 fire for a content of the neuron σ_r described by the regular expression $(a^3)^+a^2$. To this aim we use the rule $a^3 \to a^2$ in σ_{l_i} instead of $a^3 \to a$, while in σ_r we use the rule $(a^3)^+a^2/a^5 \to a^4$ instead of $(a^3)^+a/a^4 \to a^3$. This ensures the fact that the rules of M_0 are not used instead of those of M_1 or vice versa. In neurons associated with different labels of M_0, M_1 we have to use different rules, depending on the type of instruction simulated, that is why in Figures 5.41, 5.42, and 5.43 we have written again some rules in the form $a^3 \to a^{\delta(l)}$, as in Figures 5.49 and 5.50. Specifically, $\delta(l) = 3$ if l

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

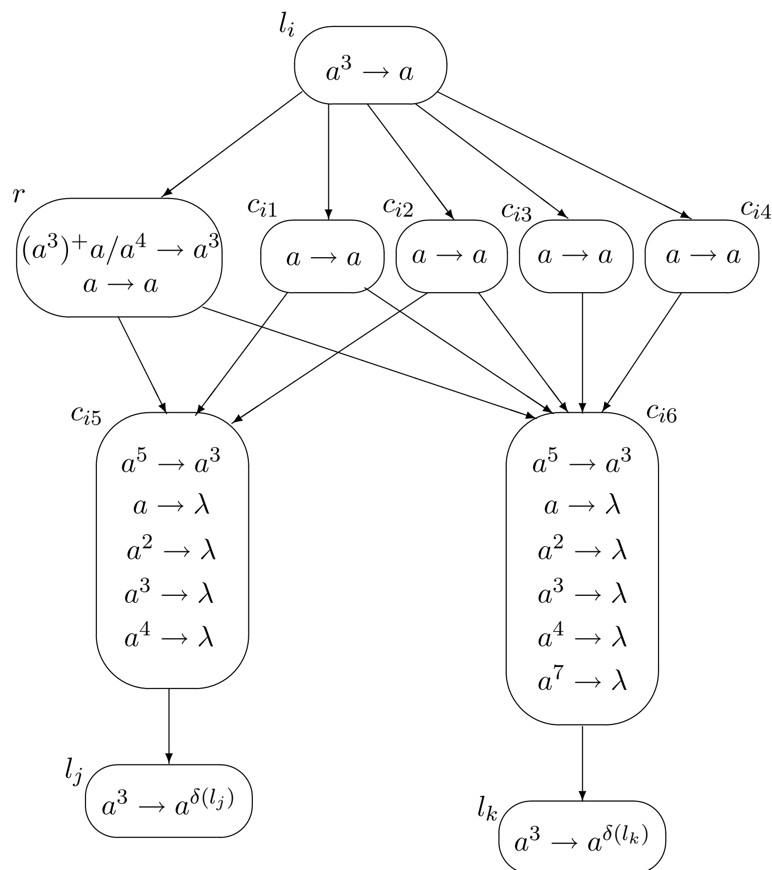


Figure 5.42: Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$) for machine M_0

labels an ADD instruction, $\delta(l) = 1$ or $\delta(l) = 2$ if l labels a SUB instruction of M_0 or of M_1, respectively, and, as one sees in Figure 10, we also take $\delta(l_{h,0}) = 2$.

With these explanations, the reader can check that the system Π works as requested, hence $L_\lambda(\Pi) = L$ (in Figures 5.42, 5.43 we have neurons with 6 rules, that is why $s' = \max(s, 6)$). □

Theorem 5.5.7 $RE = L_\lambda SN^eP_*(rule_*, cons_*, prod_*)$.

In the proof of Lemma 5.5.6, if the moments when the output neuron emits no spike are associated with the symbol b_0, then the generated strings will be shuffled with occurrences of b_0. Therefore, L is a projection of the generated language.

Corollary 5.5.5 *Every language $L \in RE, L \subseteq V^*$, can be written in the form $L = h(L')$ for some $L' \in L_{res}SN^eP_*(rule_*, cons_*, prod_*)$, where h is a projection on $V \cup \{b_0\}$ which removes the symbol b_0.*

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

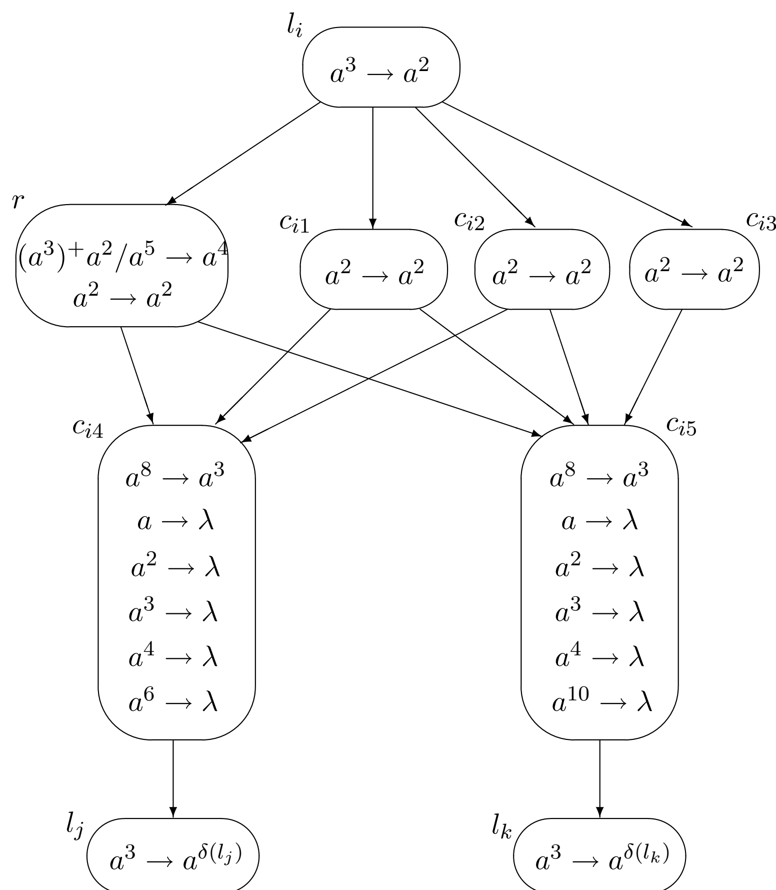


Figure 5.43: Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k))$ for machine M_1

5.5.4 Remarks and Further Research

We have investigated here the power of SN P systems with extended rules (rules allowing to introduce several spikes at the same time) both as number generators and as language generators. In the first case we have provided a simpler proof of a known universality result, in the latter case we have proved characterizations of finite and recursively enumerable languages, and representations of regular languages.

Finding characterizations (or at least representations) of other families of languages from Chomsky hierarchy and Lindenmayer area remains as a research topic. It is also of interest to investigate the possible hierarchy on the number of neurons, extending the result from Corollary 5.5.4, as well as to decrease the number of neurons from universal SN P systems.

In the next section, we produce small universal SN P systems.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

5.6 Two Small Universal SN P Systems

We have already seen that in both the generating and the accepting case, SN P systems are universal, they compute the Turing computable sets of numbers. We have also seen that the proofs are based on simulating register machines, which are known to be equivalent to Turing machines when computing (generating or accepting) sets of numbers, [60]. Small universal register machines are produced in [50].

In [50], the register machines are used for computing functions, with the universality defined as follows. Let $(\varphi_0, \varphi_1, \dots)$ be a fixed admissible enumeration of the set of unary partial recursive functions. A register machine M_u is said to be universal if there is a recursive function g such that for all natural numbers x, y we have $\varphi_x(y) = M_u(g(x), y)$. In [50], the input is introduced in registers 1 and 2, and the result is obtained in register 0 of the machine.

$$
\begin{array}{ll}
l_0 : (\mathtt{SUB}(1), l_1, l_2), & l_1 : (\mathtt{ADD}(7), l_0), \\
l_2 : (\mathtt{ADD}(6), l_3), & l_3 : (\mathtt{SUB}(5), l_2, l_4), \\
l_4 : (\mathtt{SUB}(6), l_5, l_3), & l_5 : (\mathtt{ADD}(5), l_6), \\
l_6 : (\mathtt{SUB}(7), l_7, l_8), & l_7 : (\mathtt{ADD}(1), l_4), \\
l_8 : (\mathtt{SUB}(6), l_9, l_0), & l_9 : (\mathtt{ADD}(6), l_{10}), \\
l_{10} : (\mathtt{SUB}(4), l_0, l_{11}), & l_{11} : (\mathtt{SUB}(5), l_{12}, l_{13}), \\
l_{12} : (\mathtt{SUB}(5), l_{14}, l_{15}), & l_{13} : (\mathtt{SUB}(2), l_{18}, l_{19}), \\
l_{14} : (\mathtt{SUB}(5), l_{16}, l_{17}), & l_{15} : (\mathtt{SUB}(3), l_{18}, l_{20}), \\
l_{16} : (\mathtt{ADD}(4), l_{11}), & l_{17} : (\mathtt{ADD}(2), l_{21}), \\
l_{18} : (\mathtt{SUB}(4), l_0, l_h), & l_{19} : (\mathtt{SUB}(0), l_0, l_{18}), \\
l_{20} : (\mathtt{ADD}(0), l_0), & l_{21} : (\mathtt{ADD}(3), l_{18}), \\
l_h : \mathtt{HALT}. &
\end{array}
$$

Figure 5.44: The universal register machine from [50]

The constructions from Section 5.2.2 do not provide a bound on the number of neurons, but such a bound can be found if we start from a specific universal register machine. We will use here the one with 8 registers and 23 instructions from [50] – for the reader convenience, this machine is recalled in Figure 5.44, in the notation and the setup introduced in the Section 5.2.2.

Theorem 5.6.1 *There is a universal SN P system with 84 neurons.*

Proof (Outline) We follow the way used in the first section of the chapter to simulate a register machine by an SN P system. This is done as follows: neurons are associated with each register (r) and with each label (l_i) of the machine; if a register contains a number n, then the associated neuron will contain $2n$ spikes; modules as in Figures 5.11 and 5.7 are associated with the ADD and the SUB instructions (each of these modules contains two neurons – with primed labels – which do not correspond to registers and labels of the simulated machine).

ISBN:978-84-691-9877-3/DL:T-127-2009

The work of the system is triggered by introducing two spikes in the neuron l_0 (associated with the starting instruction of the register machine). In general, the simulation of an ADD or SUB instruction starts by introducing two spikes in the neuron with the instruction label. We do not describe here in detail the (pretty transparent) way the modules from Figures 5.11 and 5.7 work, as it was previously described.

Starting with neurons 1 and 2 already loaded with $2g(x)$ and $2y$ spikes, respectively, and introducing two spikes in neuron l_0, we can compute in our system in the same way as M_u; if the computation halts, then neuron 0 will contain $2\varphi_x(y)$ spikes. What remains to do is to construct input and output modules, for reading a sequence of bits and introducing the right number of spikes in the neurons corresponding to registers 1 and 2, and, in the end of the computation, to output the contents of register 0. Modules of these types are given in Figures 5.45, 5.46, having seven and two additional neurons, respectively.

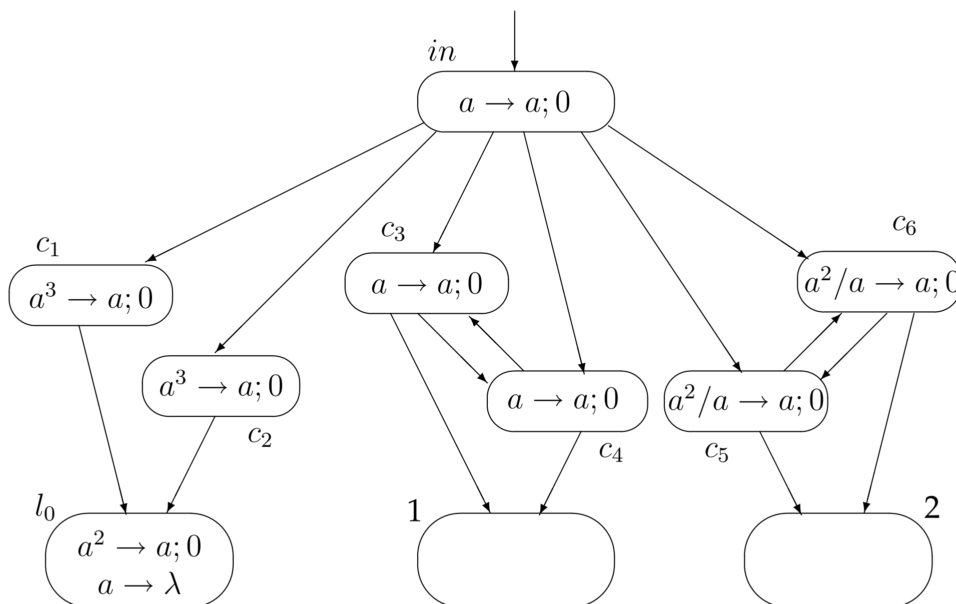


Figure 5.45: Module INPUT

After this direct construction, we get a system with 91 neurons (9 for the registers of the starting register machine – one further register is necessary for technical reasons, 25 for its labels, 24×2 for the ADD and SUB instructions, 7 in the input module, and 2 in the output module). However, some "code optimization" is possible, based on certain properties of the register machine from [50] (for instance, consecutive ADD instructions can be simulated by a specific module, smaller than two separate ADD modules); we skip the technical details and we only mention that the final SN P system will contain 84 neurons. □

This is a small number (a small "brain", compared to the human one; it would

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

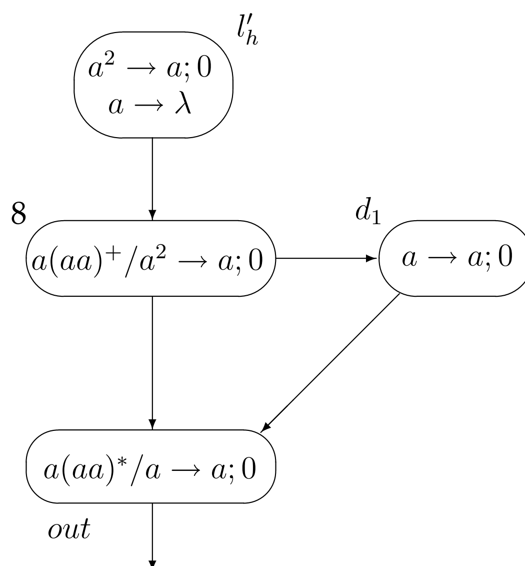


Figure 5.46: Module OUTPUT

be nice to know where in the evolution scale there are animals with about 84 neurons in their brain), but we do not know whether it is optimal or not. Anyway, we believe that in the previous setup, we cannot significantly decrease the number of neurons from a universal SN P system.

However, we can do better starting from the following observation. In many modules mentioned above we need pairs of intermediate neurons for duplicating the spike to be transmitted further (this is the case for neurons l'_i, l''_i in Figure 5.11), and this suggests to consider a slight extension of the rules of SN P systems: to allow spiking rules of the form $E/a^c \to a^p; d$, where all components are as usual, and $p \geq 1$. The meaning is that c spikes are consumed and p spikes are produced. To be "realistic", we impose the restriction $c \geq p$ (the number of produced spikes is not larger than the number of consumed spikes).

Theorem 5.6.2 *There is a universal SN P system with 49 neurons, using rules of the form $E/a^c \to a^p; 0$, with $p \geq 1$.*

(Note that the delay is zero in the rules of the extended form used in the theorem.) As above, we do not know whether this result is optimal, but we again believe that it cannot be significantly improved (without, maybe, changing the definition of SN P systems in an essential way).

5.7 Using the Rules in an Exhaustive Way

An essential difference between SN P systems and usual P systems is, besides the use of a unique type of objects, the sequential use of rules at the level of each

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

neuron (the system itself is synchronized, in each time unit the neurons work in parallel, each of them applying a rule). In what follows we introduce parallelism also at the local level, in the sense of the *exhaustive* use of rules: when a rule $E/a^c \rightarrow a^p; d$ can be applied (the contents of a neuron is described by the regular expression E), then we apply it as many times as possible in that neuron. For instance, if we have a rule $a(aa)^*/a^2 \rightarrow a; 0$ associated with a neuron which contains 7 spikes, then the rule is enabled ($a^7 \in L(a(aa)^*)$); using it exhaustively means using it 3 times, the maximal number of times a^2 is contained in a^7; thus, 6 spikes are consumed, one remains in the neuron, and 3 spikes are produced (one for each use of the rule) and sent to the neighboring neurons (3 spikes to each neuron to which a synapse exists, originating from the neuron where the rule was used).

More formally, if a rule $E/a^c \rightarrow a^p; d$ is associated with a neuron σ_i which contains k spikes, then the rule is enabled (we also say *fired*) if and only if $a^k \in L(E)$. Using the rule means the following. Assume that $k = sc + r$, for some $s \geq 1$ (this means that we must have $k \geq c$) and $0 \leq r < c$ (the remainder of dividing k by c). Then sc spikes are consumed, r spikes remain in the neuron σ_i, and sp spikes are produced and sent to the neurons σ_j such that $(i, j) \in syn$ (as usual, this means that the sp spikes are replicated and exactly sp spikes are sent to each of the neurons σ_j). In the case of the output neuron, sp spikes are also sent to the environment. Of course, if neuron σ_i has no synapse leaving from it, then the produced spikes are lost.

It is important to note that only one rule is chosen and applied, the remaining spikes cannot evolve by another rule. For instance, even if a rule $a(aa)^*/a \rightarrow a; 0$ exists, it cannot be used for the remaining unused spike after applying the rule $a(aa)^*/a^2 \rightarrow a; 0$. Of course, the rule $a(aa)^*/a \rightarrow a; 0$ can be chosen instead of $a(aa)^*/a^2 \rightarrow a; 0$, and then all spikes are consumed. This is the reason for which we use the term *exhaustive* and not the term *parallel* for describing the way the rules are used.

Another important detail is that the covering of the neuron is checked only for enabling the rule, not step by step during its application. For instance, the rule $a^7/a^2 \rightarrow a; 0$ has the same effect as $a(aa)^*/a^2 \rightarrow a; 0$ in the case of a neuron containing exactly 7 spikes: the rule is enabled, 6 spikes are consumed, 3 are produced; the 3 applications of the rule are concomitant, not one after the other, hence, all of them have the same enabling circumstances.

If several rules of a neuron are enabled at the same time, one of them is nondeterministically chosen and applied. The computations proceed as usual, and a spike train is associated with each computation by writing 0 for a step when no spike exits the system and 1 with a step when one or more spikes exit the system. Then, a number is associated – and said to be generated/computed by the respective computation – with a spike train containing at least two occurrences of the digit 1, in the form of the steps elapsed between the first two occurrences of 1 in the spike train.

For an SN P system Π, we denote by $N_2^{ex}(\Pi)$ the set of numbers computed by

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Π in this way, and by $Spik_2^{gen}P_m^{ex}(rule_k, cons_q, forg_r)$ we denote the family of all sets $N_2^{ex}(\Pi)$ generated by SN P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^r \rightarrow a^p; t$ having $r \leq q$, and all forgetting rules $E/a^c \rightarrow \lambda$ having $c \leq r$. When any of the parameters m, k, q, r is not bounded, then it is replaced with $*$. The corresponding families of sets of numbers *accepted* by SN P systems are denoted by $Spik_2^{acc}P_m^{ex}(rule_k, cons_q, forg_r)$. We stress the fact that the number to be accepted is introduced as the distance between two spikes which enter the input neuron of the system, i.e., (i) in each time unit one or no spike is introduced, not more, and (ii) exactly two spikes are introduced. When using only deterministic systems (with a unique continuation in each step of a computation), then we add the letter D in front of $Spik$.

5.7.1 Examples

In order to clarify the definitions, we start by discussing two examples.

Example 1. In the system Π_1 (Figure 5.47) we have two neurons, labeled 1 and 2 (with neuron σ_2 being the output one), which have 5 and 3 spikes, respectively, present in the initial configuration. Neuron σ_2 fires in the first step of the computation.

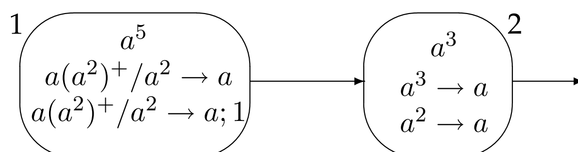


Figure 5.47: A simple example of an SN P system

One can notice that the regular expression for both firing rules in neuron σ_1 covers the number of spikes initially present in this neuron, hence we can non-deterministically choose the rule to be applied. If σ_1 uses the rule without delay ($a(a^2)^+/a^2 \rightarrow a$) in the first step it immediately sends two spikes to neuron σ_2 (one for each use of the rule) and neuron σ_2 spikes again, in the second step of the computation using the rule $a^2 \rightarrow a$. Thus, the result of the computation in this case is $2 - 1 = 1$.

If the rule with delay 1 is used in the first step by neuron σ_1, the two spikes generated reach neuron σ_2 with one step delay. In step 3 neuron σ_2 fires and the result of the computation is $3 - 1 = 2$.

Hence, Π_1 generates the finite set $\{1, 2\}$.

Example 2. The second example we consider is depicted in Figure 5.48. The system has four neurons and all of them, including the output one, fire in the first step of the computation. Note that neuron σ_3 can choose non-deterministically which rule to apply.

If neurons σ_1, σ_2, and σ_3 are using the same rule ($a(a^2)^+/a^2 \rightarrow a$), then they send to neuron σ_4, at every step, 6 spikes (2 from each neuron) which are deleted

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

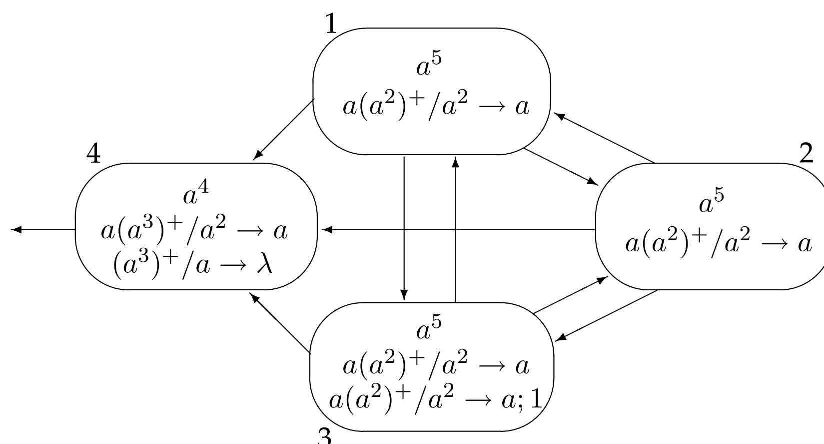


Figure 5.48: An SN P system generating all natural numbers

a step later applying the rule $(a^3)^+/a \to \lambda$. At the same time, the spikes that were consumed in each of the neurons σ_1, σ_2, and σ_3 are replaced by other spikes coming from the neurons they have incoming synapses with.

If at a moment during the computation, let us say t, neuron σ_3 chooses to use the rule with delay 1, a step later neuron σ_4 will receive only 4 spikes (2 from each of the neurons σ_1 and σ_2), and it will fire for the second time (in step $t+1$) using the rule $a(a^3)^+/a^2 \to a$. We thus have computed the number t as the result of the difference between $t+1$ and 1.

5.7.2 Computational Completeness

Let us start by making the following simple, but useful **observations**:

1. If an SN P system Π has only rules of the form $a^c \to a^p; d$ and forgetting rules $a^s \to \lambda$, then in each neuron each rule can be used exactly once, hence in this case the exhaustive mode coincides with the sequential mode.

2. Then, there are constructions in Section 5.2 where we used neurons with two spikes and two rules of the form $a^2/a \to a,\ \ a \to a; n$, such that this neuron spikes twice, at interval of n steps. In the exhaustive mode, when enabled, the first rule will consume both spikes, but the functioning of the neuron in the exhaustive mode can be the same as in the constructions from Section 5.2 if we start with three spikes and use the rules $a^3/a^2 \to a,\ \ a \to a; n$ (the first rule consumes two spikes and the second rule consumes the third spike; each rule is used only once).

Using these observations, several examples and results from Section 5.2 can be carried to the exhausting case. In particular, this is true for the characterizations of finite sets of numbers (they equal the sets generated by SN P systems with one or two neurons) and for semilinear sets of numbers (their family is equal to the

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

family of sets of numbers generated by SN P systems with bounded neurons: a bound there is on the number of spikes contained by the neurons of the system in any step of a computation).

We do not enter into details, but we just mention that the examples from Figures 5.3 (generating all even numbers) and 5.5 (generating all natural numbers) from Section 5.2, as well as all Lemmas 5.2.2 – 5.2.7 from Section 5.2 are valid also for the exhaustive way of using the rules, via the previous two observations, and this directly leads to the two characterization results mentioned above.

We pass now to proving the equivalence with Turing machines, considering first the generative case.

Theorem 5.7.1 $Spik_2^{gen}P_*^{ex}(rule_k, cons_p, forg_q) = NRE$ *for all* $k \geq 5, p \geq 5, q \geq 1$.

Proof We only have to prove the inclusion $NRE \subseteq Spik_2^{gen}P_*^{ex}(rule_5, cons_5, forg_1)$, and to this aim, we use the characterization of NRE by means of register machines.

Let $M = (m, I, l_0, l_h)$ be a register machine, having the properties specified in Section 2.2.3: the result of a computation is the number from register 1 and this register is never decremented during the computation.

We construct a spiking neural P system Π as follows.

For each register r of M we consider a neuron σ_r in Π whose contents correspond to the contents of the register. Specifically, if the register r holds the number $n \geq 0$, then the neuron σ_r will contain 3^{n+1} spikes; thus, number zero (the case when the register is empty), is represented by a register with 3 spikes inside, that is, at the beginning of the computation each neuron σ_r corresponding to a register r of M contains 3 spikes.

Increasing by one the contents of a register r which holds the number n means multiplying by 3 the number 3^{n+1}, of spikes from the neuron σ_r; checking whether the register is empty amounts at checking whether σ_r contains exactly 3 spikes, and decreasing by one the contents of a non-empty register means to divide by 3 the number of spikes.

With each label l of an instruction in M we also associate a neuron σ_l. Initially, all these neurons are empty, with the exception of the neuron σ_{l_0} associated with the start label of M, which contains 2 spikes. This means that this neuron is "activated". During the computation, the neuron σ_l which receives 2 spikes will become active. Thus, simulating an instruction $l_i : (\mathtt{OP}(r), l_j, l_k)$ of M means starting with neuron σ_{l_i} activated, operating the register r as requested by `OP`, then introducing 2 spikes in one of the neurons $\sigma_{l_j}, \sigma_{l_k}$, which becomes in this way active. When activating the neuron σ_{l_h}, associated with the halting label of M, the computation in M is completely simulated in Π, and we have to output the result in the form of a spike train with the distance between the first two spikes equal to the number stored in the first register of M.

Further neurons will be associated with the registers and the labels of M in a way which will be described immediately. All of them are initially empty, with the exception of the output neuron, which contains 4 spikes.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

The construction itself is not given in technical terms, but we present modules associated with the instructions of M (as well as the module for producing the output) in the graphical form introduced in the previous section. These modules are presented in Figures 5.49, 5.50, 5.51. Before describing these modules and their work, let us remember that the labels are injectively associated with the instructions of M, hence each label precisely identifies one instruction, either an ADD or a SUB one, with the halting label having a special situation – it will be dealt with by the FIN module.

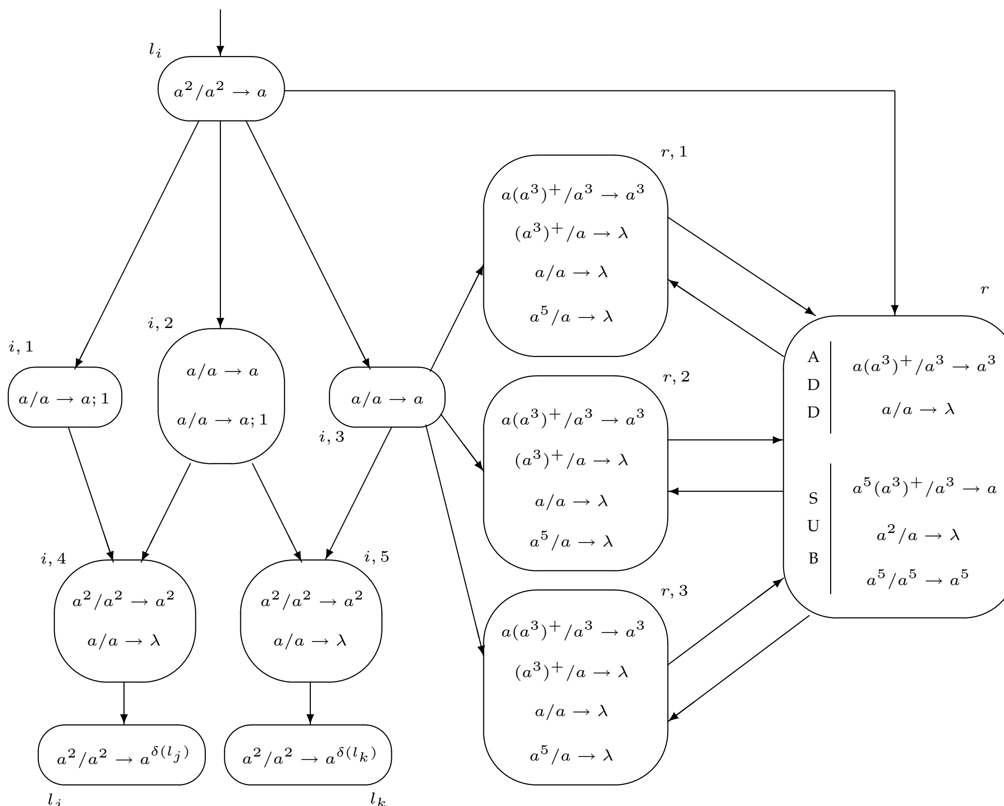


Figure 5.49: Module ADD (simulating $l_i : (\mathtt{ADD}(r), l_j, l_k)$)

Simulating an ADD instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ – module ADD (Figure 5.49).

The initial instruction, labeled l_0, is an ADD instruction. Assume that we are in a step when we have to simulate an instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$, with two spikes present in neuron σ_{l_i} (like σ_{l_0} in the initial configuration) and no spike in any other neuron, except those neurons associated with the registers (and the output neuron). Having two spikes inside, neuron σ_{l_i} gets fired. Its spike will simultaneously go to four neurons: $\sigma_{i,1}, \sigma_{i,2}, \sigma_{i,3}$, and σ_r.

Let us follow first the functioning of the first three neurons from this list. Their task is to pass non-deterministically to one of the instructions with labels l_j and

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

l_k, which in our system means activating one of the neurons σ_{l_j} or σ_{l_k}. To achieve this, we use the non-determinism of the rules in neuron $\sigma_{i,2}$ (the only one which behaves non-deterministically and the only one which contains a rule with the delay different from zero). If we use the rule $a/a \to a$, then both neurons $\sigma_{i,4}, \sigma_{i,5}$ receive immediately a spike from σ. For $\sigma_{i,4}$ this is the unique spike it receives now (neuron $\sigma_{i,1}$ fires now but its spike will leave one step later), hence in the next step the forgetting rule of neuron $\sigma_{i,4}$ should be used. Instead, neuron $\sigma_{i,5}$ receives two spikes, hence in the next step it is fired. Conversely, if instead of rule $a/a \to a$, in neuron $\sigma_{i,2}$ we use the rule $a/a \to a; 1$, then it is $\sigma_{i,5}$ which receives only one spike, and immediately "forgets" it, while in the next step neuron $\sigma_{i,4}$ receives two spikes, and fires. Thus, exactly one out of neurons $\sigma_{l_j}, \sigma_{l_k}$ receives two spikes, as requested.

Let us now examine the functioning of neurons $\sigma_r, \sigma_{r,1}, \sigma_{r,2}, \sigma_{r,3}$. As long as the number of spikes from σ_r is a multiple of 3 no rules can be applied. After receiving one spike from σ_{l_i}, neuron σ_r contains $3^n + 1$ spikes, hence it can use its rule $a(a^3)^+/a^3 \to a^3$. This means that all 3^n spikes are consumed and sent to each of the neurons $\sigma_{r,1}, \sigma_{r,2}, \sigma_{r,3}$. These spikes arrive in these neurons at the same time with one spike sent here by $\sigma_{i,3}$, hence also in these neurons we can use the rule $a(a^3)^+/a^3 \to a^3$. In this way, the contents of σ_r is returned to σ_r, tripled. This happens in a step when this neuron uses the rule $a/a \to \lambda$, thus forgetting the remaining spikes here. The number of spikes in σ_r is now 3^{n+1}, which corresponds to a value of register r increased with 1. In the next step, each of neurons $\sigma_{r,1}, \sigma_{r,2}, \sigma_{r,3}$ forgets the spike remaining in them, hence they return to the state without any spike inside.

The simulation of the ADD instruction is therefore correctly completed.

The other rules mentioned in Figure 5.49 in neurons $\sigma_r, \sigma_{r,1}, \sigma_{r,2}, \sigma_{r,3}$ are used in the case of simulating instructions SUB, and we will comment them below.

Simulating a SUB instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$ – module SUB (Figure 5.50).

Let us examine now Figure 5.50, starting from the situation of having two spikes in neuron σ_{l_i} and no spike in other neurons, except neuron σ_r, which holds a number of spikes of the form 3^n; if $n = 1$, then this means that register r of M is empty.

The spikes of neuron σ_{l_i} are sent both to σ_r and to $\sigma_{i,1}$. Because of the form of the number of spikes it contains, neuron σ_r fires.

Let us assume that the register r is not empty, hence we start with at least 3^2 spikes in neuron σ_r. This means that the rule $a^5(a^3)^+/a^3 \to a$ can be used (and this is the only rule applicable in this moment). Using this rule, the number of spikes from σ_r is divided by 3, and the 2 spikes received from σ_{l_i} remain unused. All the produced spikes – at least 3 – are sent both to neuron $\sigma_{r,4}$ and to neuron $\sigma_{r,5}$. The first neuron just returns them to neuron σ_r, simultaneously with using here the rule $a^2/a \to \lambda$. Thus, the neuron σ_r returns to a contents of 3^{n-1} spikes, which corresponds to subtracting 1 from the contents of register r.

The neuron $\sigma_{r,4}$ was able to work in this way because it has received before 2 spikes from $\sigma_{i,1}$; these spikes are immediately forgotten by using the rule $a^2/a \to$

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

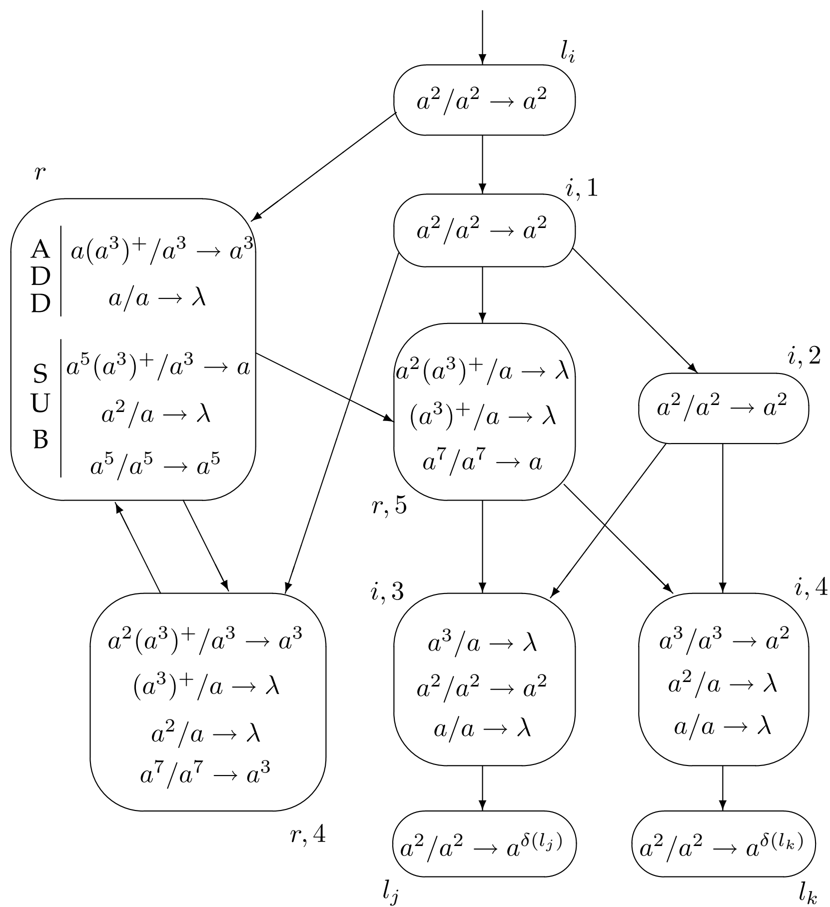


Figure 5.50: Module SUB (simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$)

λ.

In the second step both neurons $\sigma_{r,5}, \sigma_{i,2}$ have received 2 spikes from $\sigma_{i,1}$. For $\sigma_{r,5}$ this happens simultaneously with receiving 3^{n-1} spikes from σ_r, hence in the next step all the spikes from $\sigma_{r,5}$ are forgotten. Simultaneously, $\sigma_{i,2}$ sends two spikes to each neuron $\sigma_{i,3}$ and $\sigma_{i,4}$. The first of them fires, hence σ_{l_j} is activated, the other one forgets the two spikes. This is a correct simulation of the SUB instruction for the case when the register was non-empty.

If the neuron σ_r contained initially 3 spikes (hence register r was empty), then the rule to be used in the second step is $a^5/a^5 \to a^5$. This means that both $\sigma_{r,4}$ and $\sigma_{r,5}$ will end this step with 7 spikes insides (2 of them received from $\sigma_{i,1}$). While $\sigma_{r,4}$ returns 3 spikes to σ_r (in this way, σ_r ends this step with 3 spikes inside, as necessary – in the meantime, it has forgotten the 2 spikes remained here), neuron $\sigma_{r,5}$ spikes and sends one spike to each of $\sigma_{i,3}$ and $\sigma_{i,4}$. Both of them also receive at the same time 2 spikes from $\sigma_{i,2}$, hence now $\sigma_{i,4}$ will spike and $\sigma_{i,3}$ forgets its spikes. This means that neuron σ_{l_k} is activated, and again the simulation of the SUB instruction was correct.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

In both the ADD and the SUB module we have written the rules from the neurons $\sigma_{l_j}, \sigma_{l_k}$ in the form $a^2/a^2 \rightarrow a^{\delta(l_s)}$, because we do not know whether l_j and l_k are labels of ADD, SUB, or halting instructions. That is why we use the function $\delta : H \longrightarrow \{1, 2\}$, defined as follows:

$$\delta(l) = \begin{cases} 1, & \text{if } l \text{ is the label of an ADD instruction,} \\ 2, & \text{otherwise.} \end{cases}$$

Let us examine now the interferences between the simulation of different instructions. The same neuron σ_r sends spikes to all neurons $\sigma_{r,s}, s = 1, 2, 3, 4, 5$, irrespective of which is the instruction to simulate. However, neurons $\sigma_{r,s}, s = 1, 2, 3$, receive one additional spike only when simulating an ADD instruction, while neurons $\sigma_{r,s}, s = 4, 5$, receive two additional spikes only when simulating a SUB instruction. This means that in the case when any of neurons $\sigma_{r,s}, s = 1, 2, 3, 4, 5$, receive spikes without being involved in the correct simulation of an instruction as described above, those spikes are immediately forgotten by means of the rule $(a^3)^+/a \rightarrow \lambda$, present in each of them (note that the number of spikes is indeed a multiple of 3). This happens also with the 5 spikes received by the neurons $\sigma_{r,s}, s = 1, 2, 3$, in the case of simulating a SUB instruction on register r.

It is important in this discussion to remember that each element of H labels only one instruction of M, and to observe in Figures 5.49, 5.50 that the neurons are associated either with registers or with labels.

Neurons $\sigma_{r,s}, s = 1, 2, 3, 4$, send spikes only back to the associated neuron σ_r, but this is not the case with neuron $\sigma_{r,5}$, which sends spikes to all neurons $\sigma_{i,3}$ and $\sigma_{i,4}$ associated with all SUB instructions of the form $l_i : (\mathtt{SUB}(r), l_j, l_k)$ (i.e., operating on the register r). However, only one spike is sent; if it reaches a neuron which is not involved in the current simulation of a SUB instruction, then this spike is immediately forgotten by the rule $a/a \rightarrow \lambda$ present in all neurons of the form $\sigma_{i,3}, \sigma_{i,4}$ (the "correct" destinations of the spike emitted by $\sigma_{r,5}$ also receive 2 spikes from the "correct" $\sigma_{i,2}$).

Consequently, no incorrect step is possible in Π because of the interference of neurons appearing in ADD and SUB modules.

Ending a computation – module FIN (Figure 5.51).

Assume now that the computation in M halts, which means that the halting instruction is reached. For Π this means that the neuron σ_{l_h} gets two spikes and fires. At that moment, neuron σ_1 contains 3^{n+1} spikes, for $n \geq 0$ being the contents of register 1 of M. The spikes of neuron σ_{l_h} reaches immediately neurons $\sigma_{h,s}, s = 1, 2$, which fire and send spikes to σ_1, σ_{out}, and $\sigma_{h,4}$, respectively. Neuron σ_1 contains now $3^{n+1} + 2$ spikes and it can fire like in the case of a SUB instruction. Note however that this neuron was never involved in a SUB instruction, hence it does not contain any rule as those from Figure 5.50 marked with SUB in front of them.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

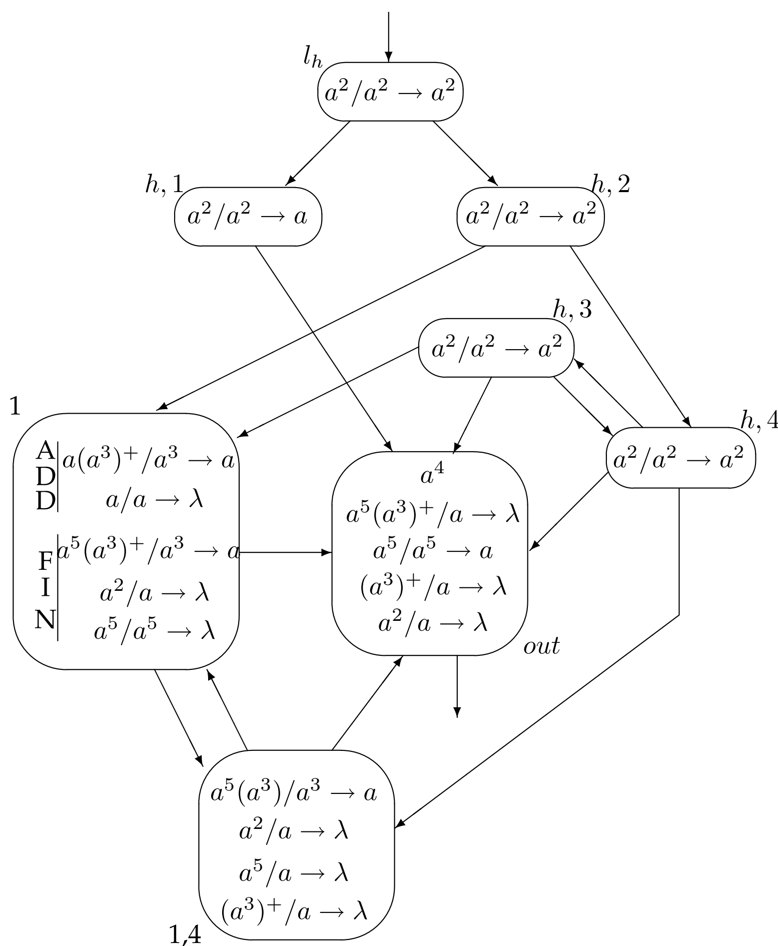


Figure 5.51: Module FIN (ending the computation)

Neurons $\sigma_1, \sigma_{h,3}$ repeatedly divide by 3 the number of spikes they contain, by making use of the rules $a^5(a^3)^+/a^3 \to a$. These rules can be used, because alternately in time, 2 spikes are sent to these neurons from $\sigma_{h,3}, \sigma_{h,4}$, respectively (first time, in the second step after activating σ_{l_h}, such spikes are sent to σ_1 from $\sigma_{h,2}$). In each step, the neurons $\sigma_1, \sigma_{h,3}$ also send spikes to the output neuron.

The output neuron spikes in the third step after activating σ_{l_h}, using the 4 spikes initially present in it and the unique spike received from $\sigma_{h,1}$. Next step when σ_{out} can spike is when receiving exactly 3 spikes from one of σ_1 and $\sigma_{1,4}$. This means that in that moment we had $3^2 + 2$ spikes in that neuron, hence this is the last step when a division with 3 is possible: in the next step, we have 5 spikes in one of σ_1 and $\sigma_{1,4}$ and 2 spikes in the other one, and all these spikes are forgotten. At the same time, the output neuron spikes again, which means that the distance between the two spikes sent out of the system is n.

Of course, if σ_1 contains initially only 3 spikes, then only one spiking of σ_{out} is possible, the one from step 3; because we do not have two spikes which exit the system, the computation provides no output (the number zero is ignored).

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Consequently, $N_2^{ex}(\Pi) = N(M)$. The observation that the maximum number of rules in a neuron is 5, the maximum number of consumed spikes is also 5, and we always use forgetting rules of the form $E/a \to \lambda$ completes the proof. □

The delay is used only in modules ADD and if it is non-zero, then the delay is 1. Actually, this feature can be avoided, at the price of slightly increasing the number of neurons in modules ADD, following the procedure introduced in [35]. It remains as an open problem to improve the previous result also from other points of view, such as the number of rules in each neuron, the number of consumed spikes in the rules, the form of the regular expressions, the indegree and the outdegree of the synapse graph, etc.

The SN P systems with the exhaustive use of rules are computationally complete also in the accepting case, even when using only deterministic systems.

We do not give here a formal statement and a proof of this assertion, but an inclusion of the form $NRE \subseteq DSpik_2^{acc}P_*^{ex}(rule_k, cons_q, forg_r)$ (with small values of k, q, r) can be obtained in a way similar to that used in the proof of Theorem 5.7.1: we start from a register machine $M = (m, H, l_0, l_h, I)$ working in the accepting mode (see again Section 2.2.3). Therefore, we may assume that M is deterministic, it starts with the number to be analyzed being introduced in register 1, and this number is accepted if and only if the machine reaches the halt instruction.

We construct a spiking neural P system Π working in the accepting mode in the same way as in the proof of the previous theorem, with the following differences:

1. The ADD modules are simpler than in Figure 5.49, because of the deterministic behavior of the ADD instructions; the changes in the construction from Figure 5.49 are simple and and the corresponding module of our SN P system is deterministic.

2. The SUB modules are the same as before, but the FIN module is missing: no action should be done when we activate the neuron σ_{l_h}, the computation in Π halts when the computation in M halts.

3. An input module is necessary, "reading" a spike train containing exactly two spikes which enter the input neuron at times t and $t + n$, and loading the neuron σ_1 of Π (hence the neuron corresponding to the first register of M) with 3^{n+1} spikes; when the second spike comes (hence the neuron σ_1 is loaded), two spikes should also be introduced in neuron σ_{l_0}, corresponding to the initial instruction of M, thus starting the computation of Π. We leave the details of the construction of this module to the reader.

5.7.3 Remarks and Further Research

We have introduced and investigated from the point of view of computing power a way to use the rules in a spiking neural P systems which we find rather natural:

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

in the exhaustive way. Specifically, when a rule is enabled, it is used as many times as possible in the respective neuron. Turing completeness was proved for this case for SN P systems used in both generative and accepting mode.

Many issues remain to be investigated for this class of SN P systems. Practically, all questions considered for sequential SN P systems are relevant also for the exhaustive case. We just list some of them: associating strings to computations (not only the binary spike train as considered here, but strings over arbitrary alphabets; specifically, if $i \geq 0$ spikes exit the output neuron, then the symbol b_i is generated, with two cases for $i = 0$: b_0 considered as a symbol, or as the empty string); finding universal SN P systems, if possible, with a small number of neurons; handling strings or infinite sequences over binary or arbitrary alphabets (both input and output neurons are considered, with the same convention: if i spikes enter/exit at a time, then the symbols b_i is considered in the input/output string); restricted classes of systems (e.g., with a bounded number of spikes present at a time in any neuron) or versions of output (taking k neurons as output neurons and thus producing vectors of dimension k of natural numbers). Proof techniques from the previous sections might be useful also in this case, while the results proved in these papers should be checked to see whether their counterparts hold true also for exhaustive SN P systems.

Another interesting issue is that of using the parallelism present in our systems in order to solve computationally hard problems in a polynomial time. Usual SN P systems are probably not able to do this (unless an arbitrarily large workspace is freely available/precomputed, initiated in polynomial time, and self-activated during the computation, as proposed in the following section). Is the parallelism of our version of SN P systems useful in this respect? (We expect a negative answer.)

Of course, a major research topic in this area is to incorporate more details from neurology, thus making the model more "realistic", and to also bring ideas/topics from neural computing – but such issues are behind the scope of this paper, which has only considered the SN P systems in the sense of natural computing, as bio-inspired computing devices.

5.8 Spiking Neural P Systems with Self-Activation

In this section we address an important issue, somewhat complementary to the computing power issue considered up to now: the computational efficiency of SN P systems. Are these devices able to solve hard problems in a polynomial time? We do not have an answer to this question, but we propose a way to address it, by means of SN P systems with self-activation. We consider a neuron to be *inactive* if it contains no spike (hence no rule can be applied in it). As soon as a spike enters a neuron (as input in the initial configuration or from another neuron through a synapse), it makes it *active* altogether with the synapses that it establishes with other neurons.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

In a self-activating SN P system we have an arbitrarily large number of neurons which differ by the number of spikes and/or of rules they contain. Some of these neurons will be active, the others will be inactive.

In what follows, we construct an SN P system for solving SAT problem in constant time. Let us consider n variables $x_1, x_2, \ldots, x_n, n \geq 1$, and a propositional formula with m clauses, $\gamma = C_1 \wedge \cdots \wedge C_m$, such that each clause $C_i, 1 \leq i \leq m$, is of the form $C_i = y_{i,1} \vee \cdots \vee y_{i,k_i}, k_i \geq 1,$ where $y_{i,j} \in \{x_k, \neg x_k \mid 1 \leq k \leq n\}$.

The set of all instances of SAT with n variables and m clauses is denoted by $SAT(\langle n, m\rangle)$.

The instance γ is encoded as a set over

$$X = \{x_{i,j}, x'_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\},$$

where $x_{i,j}$ represents variable x_j appearing in clause C_i without negation, while $x'_{i,j}$ represents variable x_j appearing in clause C_i with negation.

Now, we give an informal description of the precomputed resource structure devoted to $SAT(\langle n, m\rangle)$. We look at Figure 5.52 which depicts an SN P system working in self-activating manner using precomputed resources, where the nodes and the arrows represent the neurons and the synapses, respectively. One can notice that the nodes have (four) different shapes (◯, ⊚, □, ▹), but this is just a way to make the construction easier to understand (the shape does not imply any differences in the behavior of the nodes). Also, we see that the structure has a sort of symmetry. Namely, for each clause we have a block of ◯-neurons and ⊚-neurons.

The Device Structure The precomputed device (initially inactive) able to deal with any $\gamma \in SAT(\langle n, m\rangle)$ is formed by $2^n(m+1) + 2nm + 1$ neurons and $2^n(3m + 1)$ synapses. Further on we are giving some details on the components of this structure.

Neurons of type $\bigcirc_{c_i x_j 1/0}$: For each variable x_j of a clause C_i, we associate 2 neurons $\bigcirc_{c_i x_j 1}$ and $\bigcirc_{c_i x_j 0}$, $1 \leq i \leq m, 1 \leq j \leq n$. Obviously, the subscript of the neurons indicates the clause (c_i for the clause C_i), and the variable (x_j for the jth variable in the clause). 1 and 0 are used to mark differently the two neurons needed to encode the same variable; their use will be detailed further on in the paper where the encoding part will be explained. However, each clause is described by $2n$ neurons, and there are exactly $2nm$ neurons of type $\bigcirc_{c_i x_j 1/0}$ associated with m clauses.

Neurons of type $\circledcirc_{c_i bin}$: There are 2^n $\circledcirc_{c_i bin}$ neurons, associated to each clause C_i, injectively labeled with elements of $\{c_i bin \mid bin \in \{1, 0\}^n\}$. They correspond to the 2^n truth-assignments for variables variables $x_1, \ldots, x_n$. In total, the device has $2^n m$ $\circledcirc_{c_i bin}$ neurons (2^n for each of the m clauses). Later on, we will see that these neurons will handle, during the computation, boolean operation $\vee$ (OR) present in the clauses of the formula.

Synapses $\bigcirc_{c_i x_j 1/0} \longrightarrow \circledcirc_{c_i bin}$: The connections are in one direction from $\bigcirc_{c_i x_j 1/0}$ to $\circledcirc_{c_i bin}$. The synapses are designed in such a way that the two neurons linked

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

by a connection have the same prefix of the labels (c_i), and the last symbol of label $c_i x_j 0/1$ is the same with the jth symbol (0 or 1) of the string bin. Each $\circledcirc_{c_i bin}$ neuron is connected to n $\bigcirc_{c_i x_j 1/0}$ neurons.

Neurons of type $\Box_{bin}$: There are exactly 2^n $\Box_{bin}$ neurons in the device labeled injectively with strings from $bin \in \{0,1\}^n$. These neurons are designed to handle boolean operation $\wedge$ (AND) between the clauses of the formula.

Synapses $\circledcirc_{c_i bin} \longrightarrow \Box_{bin}$: Each $\Box_{bin}$ neuron is connected to one $\circledcirc_{c_i bin}$ neuron from each clause block, hence, m double circled neurons may send spikes to each square neuron. Strings bin from the labels of the connected $\circledcirc_{c_i bin}$ and $\Box_{bin}$ neurons are the same.

Neuron of type $\triangleright_4$: Finally, there is a unique output neuron $\triangleright$ with label 4. By choosing label 4 for this neuron we emphasize that it spikes in the 4th step of the computation if the problem has at least a solution and does not spike in this step, otherwise. All $\Box_{bin}$ neurons are connected to the output neuron, hence, there are 2^n connections of type $\Box_{bin} \longrightarrow \triangleright_4$.

Rules: Here are the rules which apply to each type of neurons:
$R_\circ = \{a \to \lambda,\ a^2 \to a; 0,\ a^3 \to \lambda,\ a^4 \to a; 0\}$,
$R_\circledcirc = \{a^+/a \to a; 0\}$,
$R_\Box = \{a^m \to a; 0\}$,
$R_\triangleright = \{a^+/a \to a; 0\}$.

We have described the precomputed device structure for solving SAT problem as depicted in Figure 5.52, with the neurons in the inactive mode. Note that this structure is independent of the instance of *SAT* we want to solve, and it depends only on n and m. Let us explain now how we encode the particular instance of the problem into the device.

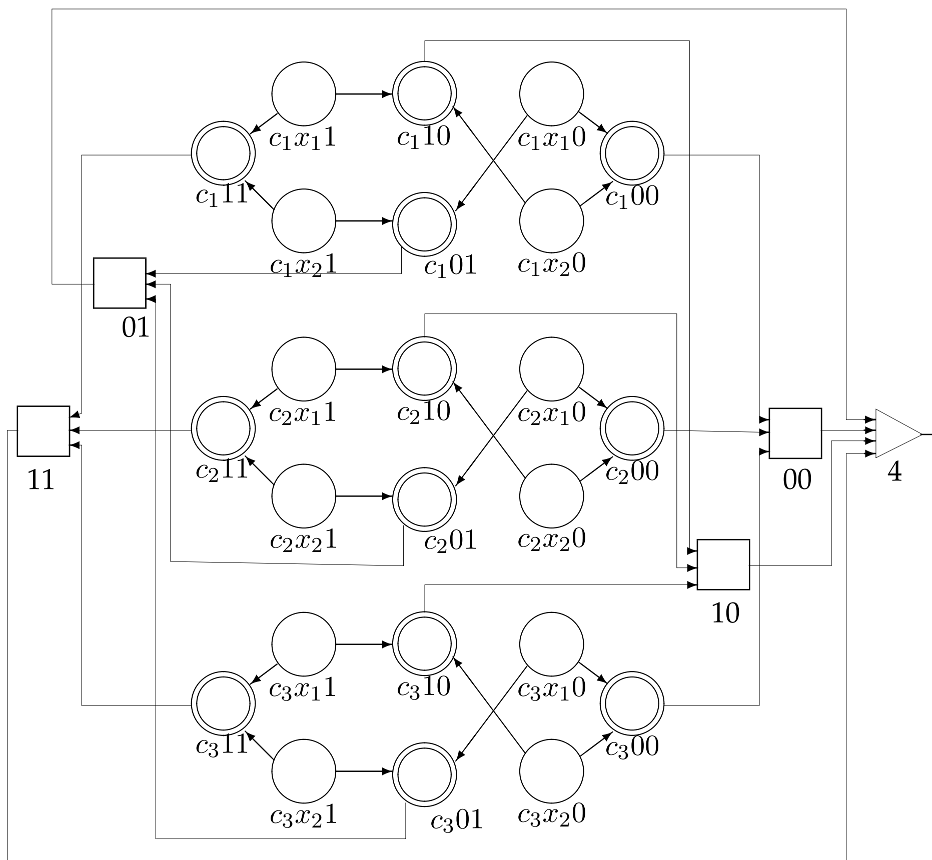
The Problem Encoding The variables are encoded by spikes as follows: one assigns values 1 and 0 to each variable x_j and $\neg x_j$. Further, a variable x_j is encoded by two spikes (a^2), and one spike (a) if we assign to x_j values 1 and 0, respectively. Similarly, we use a^3 and a^4 to encode variable $\neg x_j$, which has assigned values 1 and 0, respectively.

	1	0
$x_{i,j}$	$(a^2, \bigcirc_{c_i x_j 1})$	$(a, \bigcirc_{c_i x_j 0})$
$\neg x_{i,j}$	$(a^3, \bigcirc_{c_i x_j 1})$	$(a^4, \bigcirc_{c_i x_j 0})$

Table 5.2: The variable encoding.

In Table 5.2 one can notice how we introduce the encoded variables (the spikes) into the precomputed device. The (encoded) variables x_j or $\neg x_j$, from a clause C_i, assigned with value 1 are introduced in the neuron with label $c_i x_j 1$ ($\bigcirc_{c_i x_j 1}$), while the other "half" of the encoding, the one where variable is assigned with value 0 is introduced in the neuron labeled $c_i x_j 0$, (hence $\bigcirc_{c_i x_j 0}$). Some of the $\bigcirc_{c_i x_j 1/0}$ neurons will not be activated in the case the corresponding variables are missing form the given instance of the problem. Anyway, we stress the fact that

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009



Rules used in $\Pi_{SAT(\langle 2,3\rangle)}$: $R_\circ = \{a \to \lambda, a^2 \to a; 0, a^3 \to \lambda, a^4 \to a; 0\}$;
$R_\circledcirc = \{a^+/a \to a; 0\}$; $R_\square = \{a^3 \to a; 0\}$; $R_\triangleright = \{a^+/a \to a; 0\}$.

Figure 5.52: Precomputed spiking neural net for SAT($\langle 2, 3\rangle$)

at most $2nm$ $\bigcirc_{c_ix_j1/0}$ neurons are activated in when the device is initialized, the other neurons in the system remaining inactive.

The Computation Starting this moment (when the device is initialized and the corresponding neurons activated), the computation can be performed and the problems will be solved in 4 steps. Once the system starts to evolve the spikes follow the one-directional path:

$$\bigcirc\text{-neurons} \longrightarrow \circledcirc\text{-neurons} \longrightarrow \square\text{-neurons} \longrightarrow \triangleright\text{-neuron},$$

as shown in Figure 5.52.

5.8.1 An Example

In order to illustrate the procedure discussed above, let us examine a simple example.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

We consider the following instance of SAT, with two variables and three clauses, $\gamma \in SAT(\langle 2,3\rangle)$,

$$\gamma = (x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2).$$

The device structure The system we construct is given in a pictorial way in Figure 5.52, and has 29 inactive neurons ($4 * 2^2 + 2 * 2 * 3 + 1$) and 40 synapses. There are 3 blocks of neurons, each dealing with a clause of the problem. Each block contains 4 solid circled ($\bigcirc_{c_i x_1 1}$, $\bigcirc_{c_i x_1 0}$, $\bigcirc_{c_i x_2 1}$, $\bigcirc_{c_i x_2 0}$) neurons – 2 for each variable – to which we assign 1 and 0 (see the labels). In each block, there are also 4 double circled neurons ($\circledcirc_{c_i 11}, \circledcirc_{c_i 10}, \circledcirc_{c_i 01}, \circledcirc_{c_i 00}$) connected to the $\bigcirc_{c_i x_j 1/0}$ neurons, corresponding to the 2^2 truth-assignments (see the labels). Moreover, we have 4 $\square_{bin}$ neurons connected to the corresponding $\circledcirc_{c_i bin}$ neurons.

Encoding According to the formula γ, in the first clause C_1, there is only one variable x_1. We assign values 1 and 0 to x_1 and encode it by two spikes (a^2) and one spike (a^1) that are placed in $\bigcirc_{c_1 x_1 1}$ and $\bigcirc_{c_1 x_1 0}$, respectively. The other two neurons $\bigcirc_{c_1 x_2 1}$ and $\bigcirc_{c_1 x_2 0}$ corresponding to variable x_2 remain empty, since there is no variable x_2 or $\neg x_2$ in the clause.

The second clause (C_2) is encoded in the following clause block. To each variable $\neg x_1$ and x_2 are assigned values 1 and 0. Further, they are encoded by a^3 in $\bigcirc_{c_2 x_1 1}$, a^4 in $\bigcirc_{c_2 x_1 0}$, for $\neg x_1$, and a^2 in $\bigcirc_{c_2 x_2 1}$, a^1 in $\bigcirc_{c_2 x_2 0}$, for x_2.

The last clause C_3 has only one variable, $\neg x_2$, which is encoded in the neurons $\bigcirc_{c_3 x_2 1}$ and $\bigcirc_{c_3 x_2 0}$. The other two neurons corresponding to this clause remain empty. See Step 1 of Figure 5.53.

Computation Once the encoding is done, single circled neurons are activated and send signals, or erase spikes according to the rules they contain. In the first step, neurons having inside a^2 and a^4 fire, while spikes a^1 and a^3 from the other neurons are deleted. We see in Step 2 from Figure 5.53, that only 7 double circled neurons out of 12 contain spikes, hence only 7 will be activated in the second step. Then, in next step of computation, double circled neurons containing spikes fire because of the rules $a^+/a \to a$ inside. One can notice that neurons $\square_{11}$, $\square_{00}$, and $\square_{10}$ have received two spikes, and neuron $\square_{01}$ has received only one spike. In the third step of the computation, only neuron $\circledcirc_{c_2 01}$ fires since there was one spike remained, and nothing else happens in the system. The rule present inside the square neurons, $a^3/a \to a$, cannot be applied, because there is no such neuron containing three spikes. At the fourth step, the output neuron does not spike, since no spike has arrived here in the third step of the computation, hence the given problem has no solution.

We have mentioned before that what happens after the fourth step of the computation is out of our interest because it does not give further details with respect to the satisfiability of the given problem. We just want to mention that the system may not stop after giving the answer to our problem.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

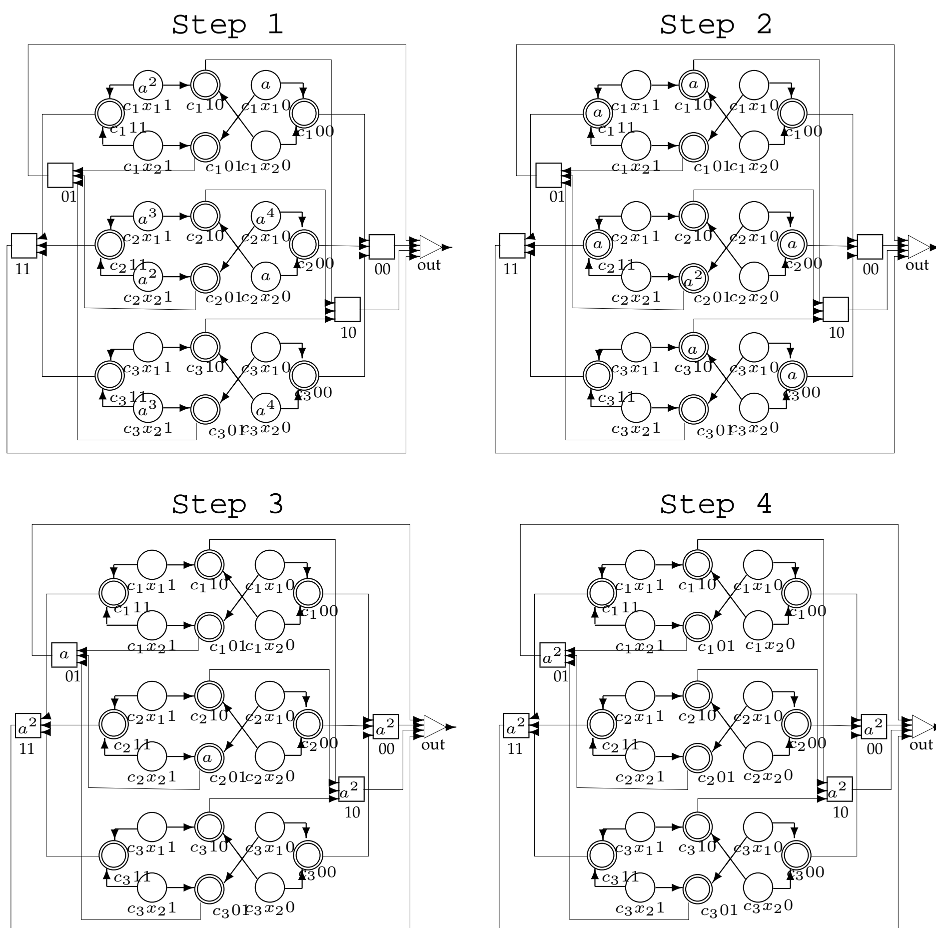


Figure 5.53: The steps of the computation for $\gamma \in SAT(\langle 2, 3\rangle)$.

5.8.2 A Formal Presentation

Formally, for given $(n, m) \in \mathbf{N}^2$, an SN P system using precomputed resources, working in a self-activating manner, devoted to solve SAT problem with n variables and m clauses, is a construct

$$\Pi^{n,m}_{\mathtt{SAT}} = (\Pi_{SAT(\langle n,m\rangle)}, \Sigma(\langle n, m\rangle))$$

with:

- $\Pi_{SAT(\langle n,m\rangle)} = (O, \mu, \triangleright_4)$, where:
    1. $O = \{a\}$ is the singleton alphabet;
    2. $\mu = (H, \Omega, R_H, syn)$ is the precomputed device structure, where:
        - $H = H_1 \cup H_2 \cup H_3 \cup H_4$ is a finite set of neuron labels, where
          $H_1 = \{c_i x_j 1, c_i x_j 0 \mid 1 \leq i \leq m, 1 \leq j \leq n\}$,
          $H_2 = \{c_i bin \mid 1 \leq i \leq m, bin \in \{0, 1\}^n\}$,

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$H_3 = \{bin \mid bin \in \{0,1\}^n\}$,
$H_4 = \{4\}$;

- $\Omega = \{(0, \bigcirc_{h_1}), (0, \circledcirc_{h_2}), (0, \Box_{h_3}), (0, \triangleright_{h_4}) \mid h_1 \in H_1, h_2 \in H_2, h_3 \in H_3, h_4 \in H_4\}$ are the empty (inactive) neurons present in the precomputed structure (with $|\Omega| = 2^n(m+1) + 2nm + 1$);
- $R_H = R_{H_1} \cup R_{H_2} \cup R_{H_3} \cup R_{H_4}$ is a finite set of rules associated to the neurons, where
  R_{H_1}=$\{a^1 \to \lambda$, $a^2 \to a; 0$, $a^3 \to \lambda$, $a^4 \to a; 0\}$,
  $R_{H_2} = R_{H_4} = \{a^+/a \to a; 0\}$,
  $R_{H_3} = \{a^m \to a; 0\}$;
- $syn = \bigcup_{i=1}^m \{(\bigcirc_{c_i x_j 1}, \circledcirc_{c_i bin}) \mid bin|_j = 1, 1 \leq j \leq n, bin \in \{0,1\}^n\}$
  $\cup \bigcup_{i=1}^m \{(\bigcirc_{c_i x_j 0}, \circledcirc_{c_i bin}) \mid bin|_j = 1, 1 \leq j \leq n, bin \in \{0,1\}^n\}$
  $\cup \{(\circledcirc_{c_i bin}, \Box_{bin}) \mid bin \in \{0,1\}^n, 1 \leq i \leq m\} \cup \{\Box_{bin}, \triangleright_4 \mid bin \in \{0,1\}^n\}$;

3. $\triangleright_4$ is the output neuron;

- $\Sigma(\langle n, m\rangle)$ is a polynomial encoding from an instance γ of SAT into $\Pi_{SAT(\langle n,m\rangle)}$, providing the initialization of the system such that

$$\begin{aligned}
\Sigma(\langle n, m\rangle)(\gamma) &= \{(1, \bigcirc_{c_i x_j 0}) \mid x_{i,j} \in \gamma, 1 \leq i \leq m, 1 \leq j \leq n\} \\
&\cup \{(2, \bigcirc_{c_i x_j 1}) \mid x_{i,j} \in \gamma, 1 \leq i \leq m, 1 \leq j \leq n\} \\
&\cup \{(3, \bigcirc_{c_i x_j 0}) \mid x'_{i,j} \in \gamma, 1 \leq i \leq m, 1 \leq j \leq n\} \\
&\cup \{(4, \bigcirc_{c_i x_j 1}) \mid x'_{i,j} \in \gamma, 1 \leq i \leq m, 1 \leq j \leq n\}.
\end{aligned}$$

In the precomputed structure of $\Pi(\langle n, m\rangle)$ (for any SAT problem with n variables and m clauses), neurons are described as a pair *(spikes inside, neuron)*. There are $2^n(m+1) + 2nm + 1$ neurons and $2^n(3m+1)$ synapses initially inactive. $\Sigma(\langle n, m\rangle)$ encodes the given instance of the problem in spikes, that is, the encoded problem is introduced into the precomputed structure (spikes are assigned to solid circled neurons) activating the corresponding neurons. It is important to note that at most $2nm$ neurons will be activated, hence the initialization takes a polynomial time.

The system evolves exactly in the same manner as the basic SN P systems. The result of the computation is obtained in its 4th step. If the system (output neuron) spikes, then the given problem has at least a solution. Otherwise, it does not have any solution. The evolution of the system after this step of the computation is ignored.

Thus, the system evolves as follows:

Step 1: After encoding the problem (the variables) in (the number of) spikes we introduce into the system, namely in at most $2nm$ neurons of type

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

$\bigcirc_{c_i 0 x_j 1/0}$, these neurons are activated and evolve according to the rules inside.

Step 2: Neurons of type $\circledcirc_{c_i bin}$ which provide information on the truth-assignments at the level of clauses (OR operations are simulated) – at most $2^n m$ initially inactive neurons – will be activated, and they will send spikes to the 2^n $\Box_{bin}$ neurons corresponding to the truth-assignments at the level of the system.

Step 3: Now, only those neurons of type $\Box_{bin}$ in which the threshold is reached (AND operations are simulated, if all m clauses are satisfied, hence there exist m spikes inside) will spike (because of the rule $a^m \to a; 0$) toward the output neuron.

Step 4: If in the output neuron will spike, it means that our problem has at least a solution. Otherwise, we do not have any solution for the given problem.

As already mentioned both in the definition of the system and in the example, it is not mandatory for the system to halt. We only observe its behavior in the fourth step of the computation.

Based on the previous explanations we can state that:

Theorem 5.8.1 $\Pi^{n,m}_{\texttt{SAT}}$ *can deterministically solve each instance of size* (n, m) *of* SAT *in constant time.*

5.8.3 Remarks and Further Research

We have shown that SN P systems are not only computationally universal, but also computationally efficient devices. We show that the idea of using an already existing, but inactive, workspace proves to be very efficient in solving **NP**-complete problems. We illustrate this possibility with the satisfiability problem. The initial system is fixed, depending only on the number of variables (n) and the number of clauses (m). Then any instance of SAT with size nxm is encoded in a polynomial time in spikes introduced in the system and then it is solved in exactly 4 steps by our device.

If we use rules of type $R_\circ = R_\triangleright = \{a^+ \to a; 0\}$ (and we interpret such a rule in the sense that if at least one spike is present in a moment inside a neuron, then it should spike immediately and all spikes are consumed) in the system $(\Pi_{SAT(\langle n,m\rangle)}, \Sigma(\langle n, m\rangle))$, then the computation *halts* in the fourth step with the system spiking if the problem has at least a solution. Otherwise the problem does not have any solution. After the fourth step, no spike will remain in the system.

Another way to stop the computation after sending the answer out is the following. Let us introduce two intermediate neurons in between each $\Box_{bin}$ and $\triangleright_4$ neuron, so that in step 4 (the intermediate neurons take one step for transmitting the spikes further) neuron $\triangle_4$ receives an even number of spikes. Then, instead

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

of the rule $a^+/a \to a;0$, in neuron $\triangleright_4$ we use the rule $(a^2)^+/a \to a;0$. Thus, if any spike reaches neuron $\triangleright_4$, then an even number of spikes arrive here; thee rule $(a^2)^+/a \to a;0$ can be used only once, in step 5, because after that the number of remaining spikes is odd.

In this way, we add $2 \cdot 2^n$ neurons and further $3 \cdot 2^n$ synapses, while the computation lasts five steps only.

Moreover, if we consider 3SAT problem, then each clause block will contain exactly $14 = 3 * 2 + 2^3$ neurons.

One possible line of research would be to try to investigate other computationally hard problems with the SN P systems using precomputed resources. Finally, investigating other computational complexity issues within this framework would be also very challenging.

5.9 Some Applications of SN P Systems

5.9.1 Simulating Logical Gates and Circuits

In this section we show how SN P systems can simulate logical gates by means of SN P systems with exhaustive use of rules. We consider that input is given in one neuron while the output will be collected from the output neuron of the system. Boolean value 1 is encoded in the spiking system by two spikes, hence a^2, while 0 is encoded as one spike.

We collect the result as follows. If the output neuron fires two spikes in the second step of the computation, then the Boolean value computed by the system is 1 (hence true). If it fires only one spike, then the result is 0 (false).

Simulating Logical Gates

Lemma 5.9.1 *Boolean AND gate can be simulated by SN P systems with exhaustive use of rules using two neurons and no delay on the rules, in two steps.*

Proof We construct the SNP system (consisting on only one neuron):

$$\Pi_{AND} = (\{a\}, \sigma_1 = (0, \{a^2 \to a;0, a^3 \to a;0, a^4/a^2 \to a;0\}), \emptyset, 1).$$

The functioning of the system is rather simple. Suppose in neuron 1 we introduce three spikes. This means we compute the logical AND between 1 and 0 (or 0 and 1). The only rule the system can use is $a^3 \to a;0$ and one spike (hence the correct result - 0 in this case) is sent to the environment.

If 4 spikes are introduced in neuron 1 (the case 11), the output neuron will fire using the rule $a^4/a^2 \to a;0$, and will send two spikes in the environment. The system with the input 00 behaves similarly to the 01 or 10 cases. The system we have constructed gives the right answer in one computational step and gets back to its initial configuration for a further use, if necessary. □

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

We want to emphasize here that no "extended" rule was used. Of course, a rule $a^4 \rightarrow a^2$ can substitute, with the same effect, the rule we have preferred above (namely $a^4/a^2 \rightarrow a; 0$) but, in simulating Boolean gates, we have tried to minimize the use of such rules. An extended rule is used only once in simulating Boolean gates, more precisely in the simulation of OR gate.

If in the system above, in the output neuron, we change only the rule $a^3 \rightarrow a; 0$ (with the rule $a^3 \rightarrow a^2; 0$) we obtain the OR gate.

Lemma 5.9.2 *Boolean OR gate can be simulated by SN P systems with exhaustive use of rules using two neurons and no delay on the rules, in two steps.*

We now pass to the simulation of logical gate NOT.

Lemma 5.9.3 *Boolean NOT gate can be simulated by SNP systems with exhaustive use of rules using two neurons, no delay on the rules, in two steps.*

Proof We first want to stress that in simulating this gate we use any extended rules. The case when such rules are used is left to the reader.

Let us construct the following SN P system:

$$\Pi_{NOT} = (\{a\}, \sigma_1, \sigma_2, \{(1, 2), (2, 1)\}, 1),$$

and:

- $\sigma_1 = (a, \{a^2/a \rightarrow a; 0, a^3 \rightarrow a; 0\})$,
- $\sigma_2 = (0, \{a/a \rightarrow a; 0, a^2/a^2 \rightarrow a; 0\})$.

Let us emphasize that in the initial configuration, neuron 1 contains 1 spike, which, once used to correctly simulate the gate, has to be present again in the neuron such that the system returns to its initial configuration. This is done with the help of neuron 2 which in step 2 of the computation refills neuron 1 with one spike.

The system is given in its initial configuration in Figure 5.54. If the input in the Boolean gate is 1, then two spikes are placed in neuron 1. Having three spikes inside (two from the input, and one initially present inside) neuron 1 can use only rule $a^3 \rightarrow a; 0$, thus sending one spike to the environment (hence Boolean 0 – the correct result – is obtained), and one spike to neuron 2. The latter one will send the spike back, in the second step of the computation by using rule $a/a \rightarrow a; 0$, and the system regains its initial configuration.

If the input in the Boolean gate is 0, hence one spike is introduced in neuron 1, it uses the rule $a^2/a \rightarrow a; 0$, two spikes are sent to the environment (and the result of the computation is 1), and to neuron 2 in the same time. In the second step of the computation neuron 2 uses the rule $a^2/a^2 \rightarrow a; 0$, consumes the two spikes present inside, and sends one back to neuron 1. The system recovers its initial configuration.

□

After showing how SN P systems can simulate logical gates, we pass to the simulation of circuits.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

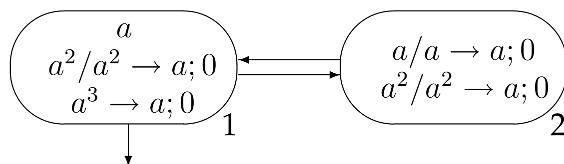


Figure 5.54: SN P systems simulating **NOT** gate

Simulating Circuits

Next, we are presenting an example of how to construct a SN P system to simulate a Boolean circuit designed to evaluate a Boolean function. Of course, in our goal we are using the systems Π_{AND}, Π_{OR}, and Π_{NOT} constructed before, to which we add extra neurons to synchronize the system for a correct output.

We start with the same **example** considered in Section 4.3.2, namely the function $f : \{0,1\}^4 \to \{0,1\}$ given by the formula

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee \neg(x_3 \wedge x_4).$$

The circuit corresponding to the above formula as well as the spiking system assign to it are depicted in Figure 5.55. In order for the system that simulates

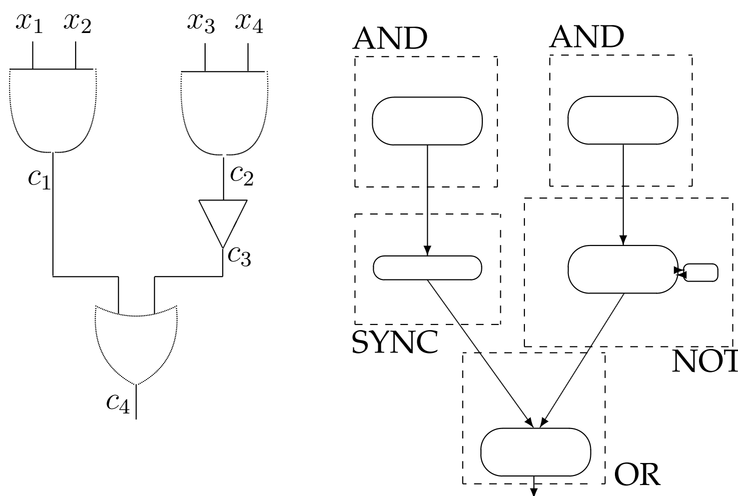


Figure 5.55: Boolean Circuit and the Spiking System

the circuit to output the correct result it is necessary for each sub-system (that simulates the gates AND, OR, and NOT) to receive the input from the above gate(s) at the same time. To this aim, we have to add synchronization neurons, initially empty with a single rule inside ($a \to a; 0$). Note that in Figure 5.55 we have added such a neuron in order for the output of the first AND gate to enter

ISBN:978-84-691-9877-3/DL:T-127-2009

gate OR at the same time with the output of NOT gate (at the end of the second step of the computation).

Having the overall image of the functioning of the system, let us give some more details on the simulation of the above formula. For that we construct the SN P system

$$\Pi_C = (\Pi^{(1)}_{AND}, \Pi^{(2)}_{AND}, \Pi^{(3)}_{NOT}, \Pi^{(4)}_{OR})$$

formed by the sub-SN P systems for each gate, and we obtain the unique result as follows:

1. For every gate of the circuit with inputs from the input gates we have an SN P system to simulate it. The input is given in neuron labeled 1 of each gate;

2. For each gate which has at least one input coming as an output of a previous gate we construct an SN P system to simulate it by considering a synapse between the output neuron of the gate from which the signal (spike) comes and the input neuron of the system that simulates the new gate.

   Note that if synchronization is needed the new synapse is constructed from the output neuron of the output gate to the synchronization neuron and from here another synapse is constructed to the input of the new gate in the circuit.

For the above formula and the circuit depicted in Figure 5.55 we will have:

– $\Pi^{(1)}_{AND}$ computes the first AND$_1$ gate $(x_1 \wedge x_2)$ with inputs x_1 and x_2.

– $\Pi^{(2)}_{AND}$ computes the second AND$_2$ gate $(x_3 \wedge x_4)$ with inputs x_3 and x_4; these two P systems, $\Pi^{(1)}_{AND}$ and $\Pi^{(2)}_{AND}$, act in parallel.

– $\Pi^{(3)}_{NOT}$ computes NOT gate $\neg(x_3 \wedge x_4)$ with input $(x_3 \wedge x_4)$. While $\Pi^{(3)}_{NOT}$ is working, the output value of the first AND$_1$ gate passes through the synchronization neuron.

– The input enters in the first neuron of OR gate, and SN P system $\Pi^{(4)}_{OR}$ completes its task. The result of the computation for OR gate (which is the result of the global P system), is sent into the environment of the whole system.

Generalizing the previous observations the following result holds:

Theorem 5.9.1 *Every Boolean circuit α, whose underlying graph structure is a rooted tree, can be simulated by an SN P system (with exhaustive use of rules), Π_α, in linear time. Π_α is constructed from SN P systems of type Π_{AND}, Π_{OR} and Π_{NOT}, by reproducing in the architecture of the neural structure, the structure of the tree associated to the circuit.*

ISBN:978-84-691-9877-3/DL:T-127-2009

5.9.2 A Sorting Algorithm

We pass now to a different problem SN P systems can solve, namely to sort n natural numbers, this time not using the rules in the exhaustive way, but as in the original definition of such systems.

We first exemplify our sorting procedure through an example. Let us presume we want to sort the natural numbers 1, 4, and 2, given in this order. For that we construct the system given only in its pictorial format below:

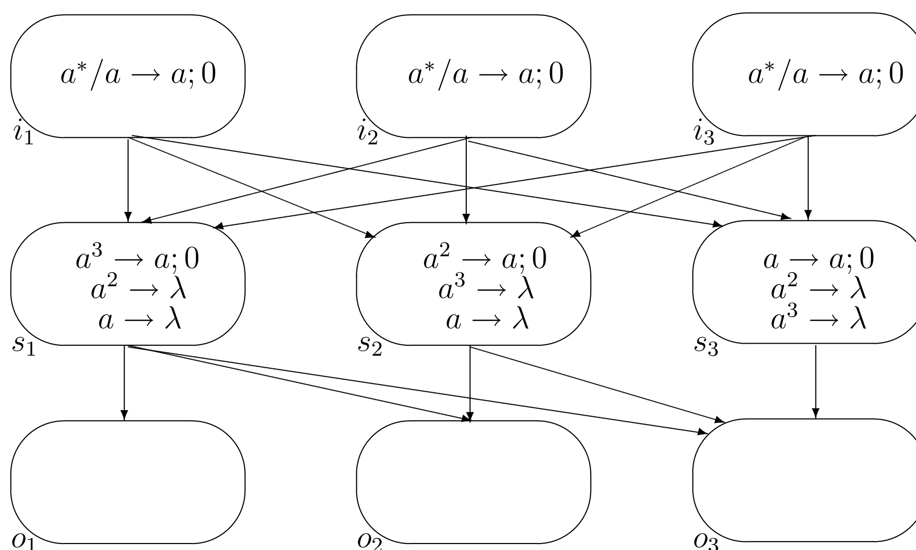


Figure 5.56: Sorting three natural numbers

We encode natural numbers in the number of spikes (1 – one spike, 4 – four spikes, 2 – two spikes) which we input in the first line of the system (hence in the neurons labeled i_1, i_2, an i_3). It can be noticed that the neurons in the first layer of the structure are having the same rule inside ($a^*/a \to a; 0$) and outgoing synapses to all the neurons in the second layer of the structure (the ones denoted s_1, s_2, and s_3). Neuron labeled s_1 has outgoing synapses with all neurons in the third layer of the system, only one spiking rule inside ($a^3 \to a; 0$, where 3 is the number of numbers that have to be sorted), and two deletion rules ($a^2 \to \lambda$, and $a \to \lambda$). For the other neurons in the second layer, the exponent of the firing rule decreases one by one as well as the synapses with the neurons from the third layer of the system.

In the initial configuration of the system we have one spike in neuron i_1, four spikes in neuron i_2 and two spikes in neuron i_3. In the *first step* of the computation, one spike from each neuron is consumed and sent to neurons from the second layer of the system. Each of them receives the same number of spikes, namely 3.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

In the *second step* of the computation, neuron labeled s_1 consumes all three spikes previously received and fires to neurons o_1, o_2 and o_3. Hence, each neuron from the output layer has one spike inside. The other neurons from the second layer delete the three spikes they have received. In the same time neurons i_2 and i_3 fire again sending 2 spikes (one each) to all neurons from the second layer.

In the *third step* of the computation, neuron s_2 fires only to neurons o_2 and o_3 (so, they will have one more spike inside, hence 2, while o_1 remains with only one spike), the other spikes from neurons s_1 and s_3 being deleted. In the same time neuron i_2 refills the neurons in the second layer of the system with one spike, which will be consumed in the *fourth step* of the computation by neuron s_3 and sent to the output neuron o_3. In the same step neuron i_2 spikes again sending the spike to the neurons from the second layer. Only neuron s_3 will spike in the *fifth step* of the computation sending its spike to neuron o_3.

So, in the last step of the computation there are: 1 spike in the neuron o_1, 2 spikes in the neuron o_2, and 4 spikes in the neuron o_3.

We pass now to the general case, constructing the system in the pictorial form:

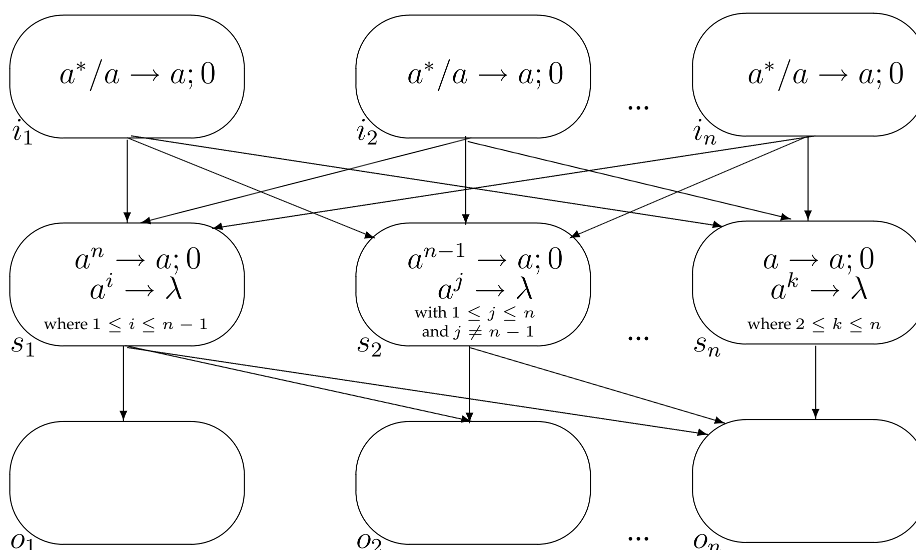


Figure 5.57: Sorting *n* natural numbers

The functioning of the system is similar to the one described in the example above. We introduce n natural numbers encoded as spikes, one in each neuron from the first layer of the structure (denoted by i_j, with $1 \leq j \leq n$). As long as they are not empty they consume at each step a spike, and send n spikes, one to each neuron from the second layer of the structure (denoted by s_i, with $1 \leq i \leq n$). The latter neurons have n different thresholds (decreasing one by one from n – neuron labeled s_1, to 1 – neuron labeled s_n), and have n different number of synapses with the neurons from the third layer of the structure. The latter ones

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

contain the result of the computation.

Theorem 5.9.2 *SN P systems can sort a vector of natural numbers where each number is given as number of spikes introduced in the neural structure.*

Based on the above construction, the time complexity (measured as usually as the number of configurations reached during the computation) is $O(T)$, where T is the magnitude of the numbers to be sorted. Although the time complexity is better than the ”classical”, sequential algorithm, in this case one can notice that the construction presented depends on the number of numbers to be sorted.

5.9.3 Remarks and Further Research

Spiking neural P systems are a versatile formal model of computation that can be used for designing efficient parallel algorithms for solving known computer science problems. Here we firstly studied the ability of SN P systems to simulate Boolean circuits. In addition, this simulation, enriched with some ”memory modules” (given in the form of some SN P sub-systems), may constitute an alternative proof of the computational completeness of the model.

Another issue studied here regards the sorting of a vector of natural numbers using SN P systems. In this case, due to its parallel features, the obtained time complexity for the proposed algorithm overcomes the classical sequential ones.

Several open problems arose during our research. For instance, in case of Boolean circuits the simulation is done for such circuits whose underlying graphs have rooted tree structures, therefore a constraint that need further investigations.

In what regards the sorting algorithm, the presented construction depends on the magnitude of the numbers to be sorted. We conjecture that this inconvenient might be eliminated. Also, we conjecture that further improvements concerning time complexity can be made.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Chapter 6

An SN P Systems Simulator

This chapter will present (various screen-shots from) a software which simulates the behavior of a SN P system. The software (written in JAVA programming language) allows the user to *construct* its own SN P system and then to *observe* the way it evolves during the computation.

The variant of the SN P system model chosen to be simulated is the initial one introduced in Section 5.2. More, the test example used below is the system depicted in Figure 5.5 (An SN P system generating all natural numbers).

6.1 Construction Area

Construction Area is the top part (Figure 6.1) of the user interface shown in Figure 6.2.

Figure 6.1: Construction Area

Here the user can construct the system whose behavior (or result) wants in a very easy way, just with the help of the buttons.

Hence, just by pushing "*Add Neuron*" button and then clicking in the *Simulation Area* a neuron will be added there. "*Add Synapse*" button constructs a synapse by clicking on the button and then by dragging the mouse from the pre-synaptic neuron to the post-synaptic one. "*Add Spike*" button adds spikes to the neurons. If one wants to add 2 spikes to neuron 1 then first clicks on the button and then goes over neuron 1 and clicks twice. "*Add Rule*" button adds rules to the neurons. The rule to be added is chosen from a (predefined) list of rules and then dragged to the neuron.

In Figure 6.3 one can see that we have constructed the first neuron of the system.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Figure 6.2: User Interface

"*Mark Output*" marks the output neuron of the system. A user should first click on this button and then to the neuron it designates as the output neuron. The color of the neuron will then change from *blue* to *yellow*.

The last button of this area is "*Erase*" button. As it names says it simply deletes the screen making it ready to construct another system.

6.2 Simulation Area

When the button "*Step*" is pushed for the first time, number 1 (indicating that the first step is simulated) appears on the same line with the buttons in the Simulation Area and, in the same time, the rules used in this step are colored in *red* (the rules without delay) and in *pink* (the rules with delay – to mark that the neuron is closed for one/some step/steps). As for the rules, instead of writing $a^2 \to \lambda$ we wrote $a2 \to a0$.

When the same button is clicked again, the number of the step (here 1) does not change and the rules perform their task with the neurons interchanging

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

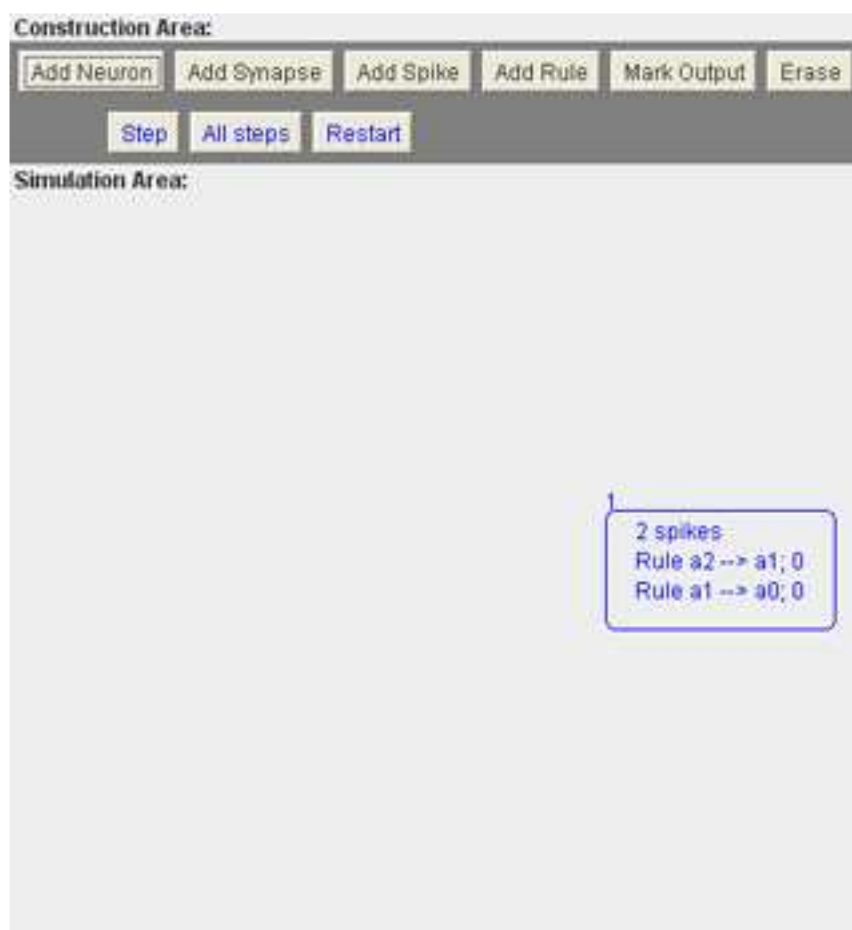


Figure 6.3: A neuron

spikes between them. For a better understanding of how these system work we first mark the rules which are going to be used (first click of the button *"Step"*) and only at the second click of the button the computational step is simulated.

Note that when the output neuron spikes, the word *"SPIKE"* written in *green* appears on the same level with the buttons and the computational step. At the

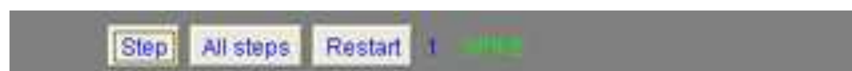


Figure 6.4: Message when the output neuron spikes

end of the computation the result is given after the text marked in *green*: *"The result is: "*.

If the user is not interested in the evolution of the system but only in the result a particular system can generate then it can click on the button *"All steps"* and the result is prompted immediately.

Button "Restart" once clicked returns the simulation in its initial phase, hence

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

Figure 6.5: Message at the end of the computation

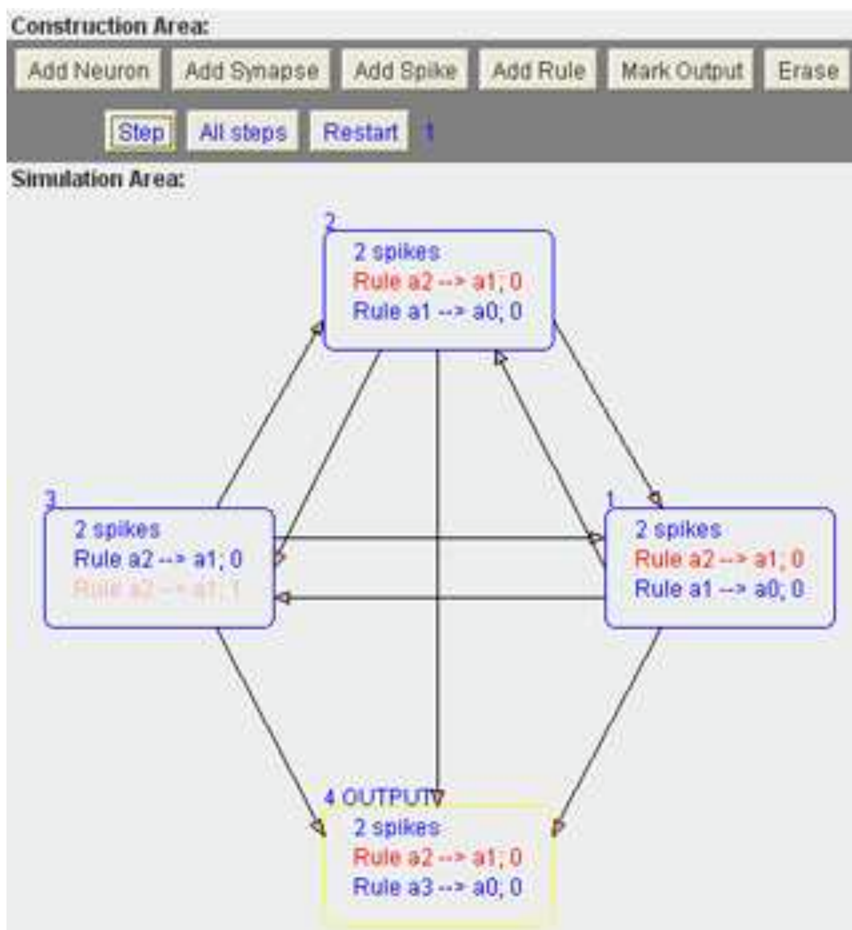


Figure 6.6: The SN P system during the first step of the computation

in the initial configuration of the SN P system.

ISBN:978-84-691-9877-3/DL:T-127-2009

Bibliography

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.

[2] A. Alhazov: P systems without multiplicities of symbol-objects, *Information Processing Letters*, 100(3), 2006, 124–129.

[3] A. Alhazov, M. Cavaliere: Proton pumping P systems, *Membrane Computing*, (C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), *LNCS* 2933, 2004, 70–88.

[4] A. Alhazov, T.-O. Ishdorj: Membrane operations in P systems with active membranes, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, 37–44.

[5] A. Alhazov, L. Pan, Gh. Păun: Trading polarizations for labels in P systems with active membranes, *Acta Informatica*, 41(2-3), 2004, 111–144.

[6] I.I. Ardelean: The relevance of cell membranes for P systems. General aspects, *Fundamenta Informaticae*, 49, 2002, 35–43.

[7] I.I. Ardelean, M. Cavaliere, D. Sburlan: Computing using signals: From cells to P systems, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, 60–73, and *Soft Computing*, 9(9), 2005, 631–639.

[8] F. Arroyo, A.V. Baranda, J. Castellanos, Gh. Păun: Membrane computing: The power of (rule) creation, *Journal of Universal Computer Science*, 8(3), 2002, 369–381.

[9] C.H. Bennet: Logical reversibility of computation, *IBM Journal of Research and Development*, 17, 1973, 525–532.

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

[11] C.S. Calude, Gh. Păun, G. Rozenberg, A. Salomaa (eds.): *Multiset Processing*, *LNCS 2235*, Springer-Verlag, Berlin, 2001.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

[12] M. Cavaliere, O. Egecioglu, O.H. Ibarra, M. Ionescu, G. Păun, S. Woodworth: Asynchronous spiking neural P systems: Decidability and undecidability, *Proceedings of The 13th International Meeting on DNA Computing* (DNA13), Memphis, Tennessee, USA, June 2007.

[13] M. Cavaliere, M. Ionescu, T.-O. Ishdorj: Inhibiting/de-inhibiting rules in P systems, *Proceedings of Fifth Workshop on Membrane Computing* (WMC5), and *LNCS* 3365, Springer, Berlin, 2005, 224–238.

[14] M. Cavaliere, M. Ionescu, T.-O. Ishdorj: Inhibiting/de-inhibiting P systems with active membranes, *Cellular Computing (Complexity Aspects)*, ESP PESC Exploratory Workshop, Fénix Editora, Sevilla, 2005, 117–130.

[15] R. Ceterchi, D. Sburlan: Simulating boolean circuits with P systems, *Proceedings of Workshop on Membrane Computing WMC-Tarragona* (WMC2003) (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa eds.), and *LNCS*, 2933, Springer-Verlag, Berlin, 2004, 104–122.

[16] H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems, *Fundamenta Informaticae*, 75(1-4), 2007, 141–162.

[17] H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems, *Proceedings of the Eighth International Conference on Electronics, Information, and Communication*, Ulaanbaatar, Mongolia, June 2006, 49–52.

[18] H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: Universality and languages, submitted, 2006.

[19] H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by (small) spiking neural P systems, *Brainstorming Week on Membrane Computing 2006*, and *Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, 94–105.

[20] E. Csuhaj-Varju, G. Vaszil: P automata, in [74]

[21] W.S. McCulloch, W.H. Pitts: A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5, 1943, 115–133.

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

[23] J. Engelfriet, G. Rozenberg: Fixed point languages, equality languages, and representations of recursively enumerable languages. *Journal of the ACM*, 27, 1980, 499–518.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

[24] R. Freund: Bidirectional sticker systems and representations of RE languages by copy languages, *Computing with Bio-Molecules. Theory and Experiments* (Gh. Păun, ed.), Springer-Verlag, Singapore, 1998, 182–199.

[25] R. Freund, M. Ionescu, M. Oswald: Extended spiking neural P systems with decaying spikes and/or total spiking, *Proceedings of Automata for Cellular and Molecular Computing*, International Workshop, August 31, 2007, Budapest, Hungary.

[26] R. Freund, M. Oswald: A short note on analysing P systems, *Bulletin of the EATCS*, 78, 2002, 231–236.

[27] R. Freund, Gh. Păun: On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars. *Proc. Conf. Universal Machines and Computations*, Chişinău, 2001 (M. Margenstern and Y. Rogozhin, eds.), *LNCS* 2055, Springer-Verlag, Berlin, 2001, 214–225.

[28] R. Freund, M. Oswald: P systems with activated/prohibited membrane channels, in [74], 2002, 261–268.

[29] R. Freund, A. Păun: Membrane systems with symport/antiport: Universality results, in [74], 2002, 270–287.

[30] P. Frisco, H.J. Hoogeboom: Simulating counter automata by P systems with symport/antiport, in [74], 2002, 288–301.

[31] P. Frisco, H.J. Hoogeboom: P systems with symport/antiport simulating counter automata, *Acta Informaticae*, 41(2-3), 2004, 145–170.

[32] M.R. Garey, D.J. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[33] W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*, Cambridge Univ. Press, 2002.

[34] D. Hauschild, M. Jantzen: Petri nets algorithms in the theory of matrix grammars, *Acta Informatica*, 31, 1994, 719–728.

[35] O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems, *Fourth Brainstorming Week on Membrane Computing*, Fenix Editora, Sevilla, vol II, 2006, 105–136, and *Theoretical Computer Science*, 372(2-3), 2007, 196–217.

[36] O.H. Ibarra, S. Woodworth, H.-C. Yen, Z. Dang: On symport/antiport P systems and semilinear sets, *Membrane Computing, International Workshop, WMC6, Vienna, Austria, July 2005, Selected and Invited Papers* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), *LNCS* 3850, Springer, Berlin, 2006, 255–273.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

[37] M. Ionescu, T.-O. Ishdorj: Boolean circuits and a DNA algorithm in membrane computing, *Proceedings of the* 6th *Workshop on Membrane Computing*, Vienna, 2005, 410–438, and *LNCS* 3850, Springer, Berlin, 2006, 272–291.

[38] M. Ionescu, T.-O. Ishdorj: Replicative-distribution rules in P systems with active membranes, *Proceedings of First International Colloquium on Theoretical Aspects of Computing*, Guiyang, China, 20-24 September 2004, UNU/IIST Report No. 310, 263–278, and *LNCS* 4705, Springer-Verlag, Berlin, 2005, 69–84.

[39] M. Ionescu, C. Martín-Vide, A. Păun, Gh. Păun: Membrane systems with symport/antiport: (Unexpected) Universality results, *Proc. 8th Int. Meeting on DNA Based Computers* (M. Hagiya, A. Obuchi, eds.), Sapporo, Japan, 2002, 151–160, and *LNCS* 2568, Springer, Berlin, 2003, 281–290.

[40] M. Ionescu, C. Martín-Vide, A. Păun, Gh. Păun: Unexpected universality results for three classes of P systems with symport/antiport, *Natural Computing*, 2(4), 2003, 337–348.

[41] M. Ionescu, C. Martín-Vide, Gh. Păun: P systems with symport/antiport rules: The traces of objects, *Grammars*, 5, 2002, 65–79.

[42] M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems, *Proceedings of the 12th International Meeting on DNA Computing* (DNA12) (C. Mao, T. Yokomori, B.-T. Zhang, eds.), Seul, June 2006, 32–42.

[43] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems, *Fundamenta Informaticae*, 71(2-3), 2006, 279–308.

[44] M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with an exhaustive use of rules, *International Journal of Unconventional Computing*, 3, 2007, 135–153.

[45] M. Ionescu, D. Sburlan: On P systems with promoters/inhibitors, *Technical Report 01/2004*, University of Seville, Second Brainstorming Week on Membrane Computing, Sevilla, 2004, 264–280, and *Journal of Universal Computer Science*, 10(5), 2004, 581–599.

[46] M. Ionescu, D. Sburlan: Some applications of spiking neural P systems, *Proceedings of the Eighth Workshop of Membrane Computing*, WMC8, Thessaloniki, Greece, June 2007, 383–394.

[47] M. Ionescu, D. Sburlan: P systems with adjoining controlled communication rules, *Proceedings of 16th International Symposium on Fundamentals of Computation Theory*, August 27-30, 2007, Budapest, Hungary.

[48] Z. Kemp, A. Kowalczyk: Incorporating the temporal dimension in a GIS, *Innovations in GIS 1*, Taylor and Francis, London, 1994.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

[49] S.C. Kleene: Representation of events in nerve nets and finite automata, *Automata Studies*, Princeton University Press, Princeton, NJ, 1956, 3–42.

[50] I. Korec: Small universal register machines, *Theoretical Computer Science*, 168, 1996, 267–301.

[51] A. Lindenmayer: Mathematical models for cellular interaction in development I and II. *Journal of Theoretical Biology*, 18, 1968, 230–315.

[52] G. Liu, M. Ionescu: Further remarks on trace languages in P systems with symport/antiport: *Proceedings of The 7th Workshop on Membrane Computing* (WMC7), Leiden, The Nederlands, 2006, 417–428.

[53] V. Lum, P. Dadam, R. Erbe, J. Guenaver, P. Pistor, G. Walch, H. Werner, J. Woodfill: Designing dbms support for the temporal dimension, *Proceedings of SIGMOD'84 Conference*, 1984, 115–130.

[54] W. Maass: Computing with spikes, *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1), 2002, 32–36.

[55] W. Maass, C. Bishop (eds.): *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.

[56] C. Martín-Vide, A. Păun, Gh. Păun: On the power of P systems with symport rules, *Journal of Universal Computer Science*, 8, 2002, 317–331.

[57] C. Martín-Vide, A. Păun, Gh. Păun, G. Rozenberg: Membrane systems with coupled transport: Universality and normal forms, *Fundamenta Informaticae*, 49, 2002, 1–15.

[58] C. Martín-Vide, Gh. Păun: Elements of formal language theory for membrane computing, *Technical Report* 21/01 of the Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, 2001.

[59] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón: A new class of symbolic abstract neural nets: Tissue P systems, COCOON 2002, Singapore, *LNCS 2387*, Springer-Verlag, Berlin, 2002, 290–299.

[60] M.L. Minsky: *Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, 1967.

[61] L. Pan, A. Alhazov, T.-O. Ishdorj: Further remarks on P systems with active membranes, separation, merging and release rules, *Soft Computing*, 8, 2004, 1–5.

[62] L. Pan, T.-O. Ishdorj: P systems with active membranes and separation rules, *Journal of Universal Computer Science*, 10(5), 2004, 630–649.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

[63] Ch. P. Papadimitriou: *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[64] A. Păun: Membrane systems with symport/antiport: Universality results, in [74], 2002, 333–343.

[65] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport, *New Generation Computing*, 20, 2002, 295–306.

[66] A. Păun, Gh. Păun, A. Rodríguez-Patón: Further remarks on P systems with symport rules, *Ann. Univ. Al.I. Cuza, Iaşi*, 10, 2001, 3–18.

[67] A. Păun A., Gh. Păun, G. Rozenberg: Computing by communication in networks of membranes, *International Journal of Fundamentals of Computer Science*, 13, 2002, 779–798.

[68] Gh. Păun: Computing with membranes, *Journal of Computer and System Sciences*, 61, 2000, 108–143, and *Turku Center for Computer Science-TUCS Report* No 208, 1998 (www.tucs.fi).

[69] Gh. Păun: *Computing with Membranes: An Introduction*. Springer-Verlag, Berlin, 2002.

[70] Gh. Păun: P systems with active membranes: Attacking NP-complete problems, *Journal of Automata, Languages and Combinatorics*, 6(1), 2001, 75–90.

[71] Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems, *Intern. J. Found. Computer Sci.*, 17(4), 2006, 975–1002.

[72] Gh. Păun, G. Rozenberg: A guide to membrane computing, *Theoretical Computer Science*, 287(1), 2002, 73–100.

[73] Gh. Păun, G. Rozenberg, A. Salomaa: Membrane computing with external output, *Fundamenta Informaticae*, 41, 2000, 259–266.

[74] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds: *Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, 2002. Revised Papers. LNCS* 2597, Springer-Verlag, Berlin, 2003.

[75] Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen*, 60, 2002, 635–660.

[76] Z. Pawlak: *Rough Sets. Theoretical Aspects of Reasoning About Data*. Kluwer, Dordrecht, 1991.

[77] Z. Pawlak: A treatise on rough sets, *Transactions on Rough Sets* (J.F. Peters, A. Skowron, eds.), *LNCS* 3700, 2005, 1–17.

UNIVERSITAT ROVIRA I VIRGILI
MEMBRANE COMPUTING: TRACES, NEURAL INSPIRED MODELS, CONTROLS
Armand-Mihai Ionescu
ISBN:978-84-691-9877-3/DL:T-127-2009

[78] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: Complexity classes in models of cellular computation with membranes, *Natural Computing*, 2(3), 2003, 265–285.

[79] G. Rozenberg, A. Salomaa (eds.): *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.

[80] A. Salomaa: *Formal Languages*, Academic Press, New York, 1973.

[81] G.M. Shepherd: *Neurobiology*, Oxford University Press, NY Oxford, 1994.

[82] P. Sosik: P systems versus register machines: Two universality proofs, in [74], 2002, 371–382.

[83] C. Teuscher: *Turing's Connectionism. An Investigation of Neural Network Architectures*, Springer-Verlag, London, 2002.

[84] S. Yu: The time dimension of computation models, *Technical Report 549*, Computer Sci. Dept., Univ. of Western Ontario, London-Ontario, Canada, 2000.

[85] http://psystems.disco.unimib.it/

[86] http://diwww.epfl.ch/ gerstner/SPNM/SPNM.html