Université Paris Est – Créteil Val de Marne

Study of language-theoretic computational paradigms inspired by biology

Habilitation Thesis

presented and publicly defended on 22 October 2010

by

Serghei Verlan

Composition of the examining board

Referees	Jürgen Dassow	Professor at the Otto-von-Guericke-University, Magdeburg, Germany
	Jean-Louis Giavitto	Senior researcher at CNRS, University of Evry
	Natasha Jonoska	Professor at the University of South Florida, USA
Examiners	Danièle Beauquier	Professor emeritus at the University of Paris Est
	Alessandra Carbone	Professor at the University of Paris 7
	Jacques Nicolas	Senior researcher at INRIA Rennes
	Anatol Slissenko	Professor emeritus at the University of Paris Est

Laboratoire d'Algorithmique, Complexité et Logique

Acknowledgments

This work would never be possible without the help of many people who contributed in a direct or indirect way to the presented results. First of all I would like to thank all my co-authors for original ideas that I would maybe never find without their help.

I would like to address particular acknowledgments to Erzsébet Csuhaj-Varjú, Rudolf Freund, Marian Gheorghe and Ion Petre for the time they shared with me during our numerous research visits. Every such visit gave key elements to the puzzle that finally assembled into this thesis. I would also like to thank Gheorghe Păun who inspired me with many interesting ideas and Arto Salomaa whose suggestions were very valuable.

I would like to address special thanks to Jean-Louis Giavitto, Natasha Jonoska and Jürgen Dassow who gratefully accepted to review this thesis. I also thank Danièle Beauquier, Alessandra Carbone, Jacques Nicolas and Anatol Slissenko who honored me by accepting to be members of the examining board.

I'm very much indebt to the supervisor of my PhD thesis, Maurice Margenstern, who knew to give me very helpful advices at the right time.

I would also like to thank my colleagues from Moldova – Yurii Rogozhin and Artiom Alhazov. Our collaboration has gone far beyond scientific topics and I consider them now as my second family.

I am very grateful to my colleagues from the Laboratoire d'Algorithmique, Complexité et Logique from the Paris Est University who created a very nice atmosphere at the department which I enjoyed a lot. I particularly thank Danièle Beauquier, Cătălin Dima, Flore Tsila and Anatol Slissenko whose help was extremely precious.

Finally, I would like to thank my wife and my parents who always believed in me and who encouraged me during these years.

Résumé

Nos travaux de recherche se situent dans le domaine de la théorie des langages formels. Cependant, l'objet de nos études sont les opérations sur les mots et les modèles de calcul qui ont une inspiration biologique. Dans notre étude nous utilisons par exemple l'opération de recombinaison (splicing) qui formalise la recombinaison des molécules d'ADN. Un autre exemple sont les systèmes à membranes qui sont un modèle de calcul distribué modélisant la structure et le fonctionnement d'une cellule vivante.

Le mémoire est fondé sur des résultats d'universalité, décidabilité, sémantique et complétude computationnelle. Cette étude permet de comprendre mieux les possibilités et surtout les limites des différentes opérations inspirées par la biologie. Ces informations donnent la possibilité d'avoir une meilleure modélisation biologique en utilisant ces opérations, mais également permettent d'avoir un retour intéressant pour la théorie des langages formels ce qui peut être utile pour la conception de nouvelles classes d'algorithmes et outils.

Le mémoire est composé de 5 chapitres. Les chapitres 2-4 présentent le travail effectué, tandis que le chapitre 5 montre les perspectives possibles.

Le premier chapitre recueille les éléments de la théorie des langages formels qui sont utilisés par la suite dans le mémoire. Ce chapitre précise les définitions et fixe les notations utilisées.

Résumé du chapitre 2

Le deuxième chapitre contient l'étude de l'opération d'insértion/effacement (I/E). Cette opération est bien connue dans la théorie des langages formels, surtout sa variante sans contexte. Il a été montré récemment que cette opération possède également une inspiration biologique et qu'elle formalise l'hybridation inadéquate (mismatched annealing) des brins d'ADN. D'une manière informelle, l'opération d'insertion rajoute une sous-chaîne dans une chaîne de caractères à l'intérieur d'un contexte pré-défini. L'opération d'effacement est une opération inverse: une sous-chaîne est effacée dans un contexte pré-défini. Les systèmes à insertion-effacement (SIE) possèdent un ensemble fini de règles d'insertion et d'effacement et le résultat de leur calcul est un sous-ensemble de mots obtenus à partir d'un ensemble fini initial, *les axiomes*, par l'itération des règles. La complexité d'un SIE est décrite par un vecteur $(n, m, m'; p, q, q')$, dit *taille*, ou les premiers trois nombres représentent la taille maximale de la sous-chaîne insérée et la taille maximale des contextes gauche et droit. Les trois derniers nombres représentent la même information, mais pour les opérations d'effacement. La section 2.1 donne les définitions précises de l'opération d'I/E et des SIE.

Ensuite, la section 2.2 détaille les méthodes de base utilisées pour obtenir nos résultats. Nous donnons d'abord une présentation des méthodes existantes, puis nous présentons la méthode de la simulation directe que nous avons élaborée. Cette méthode nous a permis d'établir la plupart des résultats présentés dans ce chapitre.

La section 2.3 est dédiée aux SIE sans contexte, c.-à-d. aux systèmes ayant les paramètres de complexité $(n, 0, 0; p, 0, 0)$. Ces modèles ont un fonctionnement particulier: le résultat du calcul est l'insertion ou l'effacement itératif d'un nombre fini de chaînes de caractères dans les axiomes. Cette variante permet de faire le lien avec les travaux précédents des années 80 dans le cadre de la théorie des langages formels: plus précisément ce type d'opérations est une généralisation des opérations de concaténation et quotient, deux opérations fondamentales de la théories des langages formels. Nos résultats présentent une description inattendue de la puissance d'expression de ce type de SIE, en montrant que les SIE de taille $(3, 0, 0; 2, 0, 0)$ et $(2, 0, 0; 3, 0, 0)$ ont la puissance des machines de Turing, tandis que les SIE de taille $(2, 0, 0; 2, 0, 0)$ ou inférieure forment une sous-classe des langages algébriques. Les résultats obtenus donnent une réponse définitive sur la puissance d'expression des opérations d'I/E sans contexte. La présentation de cette section est fondée sur les articles [51] et [74].

La section 2.4 porte sur les SIE ayant des contextes non-vides d'un seul côté uniquement, par exemple les SIE de taille $(n, m, 0; p, q, 0)$. Ce modèle n'a jamais été considéré avant nos travaux. Il présente des caractéristiques uniques, ayant certains traits communs aux SIE sans contexte et aux SIE contextuels. En appliquant notre méthode de simulation directe on a pu classifier ces SIE en deux classes par rapport à leur puissance d'expression. Dans la première classe nous avons placé les SIE qui possèdent une grande puissance d'expression, équivalente à celle des machines de Turing. Dans la deuxième classe nous avons placé les SIE qui ont une puissance d'expression strictement inférieure. L'existence de cette deuxième classe a été l'une des découvertes importantes que nous avons réalisée – auparavant il n'y avait aucun exemple de classe de SIE qui n'avait pas de puissance d'expression maximale. La présentation de cette section est fondée sur les articles [54, 47, 48, 46].

La section 2.5 présente des extensions possibles, afin d'augmenter la puissance de calcul des SIE qui ne sont pas suffisamment puissants. Nous adaptons le concept des grammaires ayant un contrôle dirigé par graphe (graph-controlled grammars) aux SIE en remplaçant l'opération de réécriture par les opérations d'I/E. Les résultats obtenus sont conséquents: comme dans le cas des grammaires traditionnelles, la puissance d'expression des modèles obtenus augmente jusqu'au niveau maximal. La présentation de cette section est fondée sur les articles [49, 4] et le chapitre de livre [5].

Résumé du chapitre 3

Le troisième chapitre est dédié à l'étude des systèmes à membranes (SM). Ces systèmes ont été introduits par Gh. Păun qui s'est inspiré de la structure et du fonctionnement de la cellule vivante. La cellule est modélisée comme un ensemble de compartiments imbriqués les uns dans les autres et délimités par des membranes; les molécules sont modélisées par des multi-ensembles, et les interactions chim-

iques – par des règles de réécriture de multi-ensembles et changement de structure. Même si les SM peuvent être vus comme des systèmes de réécriture parallèle de multi-ensembles, leur présentation naturelle donne un outil clair et puissant pour la modélisation des interactions biologiques, surtout au niveau cellulaire.

Nous nous sommes concentrés sur l'étude théorique des SM, surtout sur leur puissance d'expression et leur sémantique.

La section 3.1 donne une présentation générale des systèmes à membranes et leurs liens avec les autres modèles existants comme les réseaux de Petri et la réécriture des multi-ensembles.

La section 3.2 présente nos résultats sur l'une des variantes les plus importantes des SM: les systèmes à membranes à communication (pure). Ces modèles formalisent la communication trans-membranaire d'une cellule vivante et permettent de modéliser le flux des substances chimiques qui passent par la membrane cellulaire. Outre le domaine biologique, ces systèmes sont très intéressants du point de vue théorique, car ils ne permettent pas de renommer, supprimer ou créer des objets dans le système, ce qui permet d'avoir une notion stricte de ressource de calcul et également de simuler la propagation des signaux dans des réseaux informatiques. Nous montrons comment est-il possible de modéliser des mécanismes du transport trans-membranaire et comment généraliser le modèle obtenu dans le cadre de la théorie de langages formels. La présentation de cette section est fondée en partie sur les articles [77, 16, 7, 76, 19]

La section 3.3 s'intéresse à la sémantique des SM et à la sémantique de l'étape du calcul en particulier. Il est important de souligner que le concept des SM est tellement simple et intuitif, qu'il n'a pas eu longtemps une définition formelle. Cela s'est accentué les dernières années, lorsque différents modes d'exécution ont commencé à être étudiés. Dans cette section nous présentons une définition mathématique précise des systèmes à membranes ayant une structure statique et nous donnons des exemples de plusieurs modes d'exécution étudiés précédemment ou pas dans la littérature. Nous montrons également que certaines variantes des SM sont identiques aux autres variantes des SM, mais avec une sémantique d'exécution différente. La présentation de cette section est fondée en partie sur les articles [26, 27, 24, 75, 3].

La section 3.4 montre l'adaptation du problème de synchronisation aux SM. On s'intéresse au problème similaire au celui du peloton d'exécution pour les automates cellulaires. On part d'une configuration du système où toutes les membranes sont dans le même état sauf une et qui doivent arriver toutes en même temps dans un état spécial (de tir). La résolution de ce problème donne la possibilité de définir plusieurs horloges dans le système. Nous donnons une solution pour le problème de synchronisation pour plusieurs types de SM; dans certains cas la solution est déterministe et dans d'autres – non-déterministe. La présentation de cette section est fondée sur les articles [10, 6].

La section 3.5 s'intéresse au problème de la construction des SM universels de petite taille. Nous rappelons qu'un système universel est un système concret qui peut simuler l'exécution de n'importe quel dispositif de calcul, en supposant qu'un codage adéquat est utilisé. La découverte des machines de Turing universelles au début du 20^e siècle a permis de fonder les bases des ordinateurs actuels. La recherche des modèles de calcul universels de petite taille est un domaine important de la théorie des langages formels et de la calculabilité, car elle permet

de montrer que des actions simples peuvent engendrer des comportements très complexes. Pour la première fois nous avons considéré ce problème dans le domaine des SM et nous avons construit un système à membranes universel fondé sur l'opération de recombinaison ayant uniquement 6 règles. Un autre résultat important est un système à membranes universel fondé sur l'opération d'antiport ayant 23 règles. Ce dernier résultat est directement convertible dans le domaine de la réécriture parallèle de multi-ensembles et pour lequel il représente un résultat fondamental. La présentation de cette section est fondée en partie sur les articles [16, 8, 64].

Résumé du chapitre 4

Ce chapitre présente un modèle de calcul original inspiré de l'opération la plus courante utilisée dans les laboratoires bio-chimiques: la séparation par le *gel electrophoresis*. Ce procédé permet de classer les molécules d'une solution par leur longueur. Nous avons formalisé ce processus et proposé une classe de modèles où la séparation des mots (correspondant aux molécules) par longueur est l'ingrédient principal. L'avantage principal du modèle obtenu est sa faisabilité *in vitro*. De plus ce modèle a un intérêt particulier du point de vue théorique. La présentation de ce chapitre est fondée en partie sur l'article [18].

Contents

Introduction

Understanding the surrounding matter, the laws and the functioning of the universe, this is the main challenge of the humankind, appeared at the dawn of times. Significant progress was made during last centuries, especially in the field of physics, which ultimately permitted the engineering of new devices and the development of new technologies. The mathematics is in tight relation with all these discoveries, as it provided a formal basis for these achievements.

The challenges of the actual science more and more reside in the field of biology. This is quite natural as the fundamental questions about life: what distinguishes living and non-living matter and what is the organization of a living thing, are considered from the very old times. Recent developments in most areas of science permit to approach above questions in a methodological way giving a hope for substantial breakthrough.

Still the biology is an experimental science that accumulates a large collection of knowledge and that needs the help of other disciplines in order to explain and correlate the gathered information. The mathematics is the primary tool serving this goal, for example, with differential equations in systems biology or statistical analysis for population's evolution.

The computer science also plays an important role in biological investigations. Large-scale databases, data mining, sequence analysis and protein structure prediction – these are several tasks that would never be possible without the arrival of the computers, computational theory and corresponding algorithms. The molecular biology highlighted the key role of DNA and of related mechanisms for the information processing, which is crucial for the understanding of the functioning of living organisms. The computer science traditionally deals with sequences, so it was natural to use the provided tools for the sequence-related research, forming the field of bioinformatics.

We note that computer science is specialized on the discrete representation of the universe and on the representation of information flow processes. Many key concepts from the field where borrowed from the biology. This is why it fits well to the biological modeling providing discrete models in contrast to a modeling with differential equations, which are continuous. This property stimulated the investigation of (computational) models and operations issued from biology. In this case real biological phenomena are abstracted and a discrete system based on the involved operations and functioning is proposed.

There are two motivations for the investigation of such models. The first motivation is very clear – since the model represents an abstraction of a biological phenomena, it is possible to describe the last one in precise terms and provide simulations, estimations and predictions. A close relationship with target biolog-

ical system permits to express it in a short and clear manner giving the hope to extract additional knowledge about the model. This approach, very nice in theory, encounters important difficulties while faced to the practice. Usually, simple models are too abstract for obtaining non-trivial relations for the initial biological system, while more complete models encounter rapidly the combinatorial explosion of their size, which makes them very difficult to analyze. The success of the approach relies on a deep knowledge of the target phenomena that permits to find a good balance between abstraction and adequateness.

The second motivation comes from the fact that most successful applications of computer science are based on ideas borrowed from the biology. During the evolution of living organisms, the nature provided solutions for many complicated problems encountered and it is very fruitful to adapt these solutions to different problems. Ant colony optimization, cellular automata, neural networks, evolutionary algorithms are some examples of such an approach.

This thesis is centered around two main topics: insertion-deletion systems and P systems. The first part does a systematical study of the operations of insertion and deletion. While it is possible to consider them as biologically inspired operations, we perform in the thesis a pure theoretical investigation in terms of the theory of formal languages. Thus, our research provide new classes of formal grammars which extend the Chomsky hierarchy by introducing new levels. We introduce a new proof method and show a series of computational-completeness and non-completeness results and discuss possibilities for the extension of the computational power for the latter case.

The research on P systems follows a different motivation. Primary, they represent a general framework for distributed multiset rewriting, which captures important processes in cell biology. In contrast to traditional approaches in systems biology, P systems provide a discrete framework for the representation of molecular interactions[1] that focuses on the structure of the system and on the identification of its components. The field of application for P systems is not limited to systems biology, the topic incorporates concepts from cell biology and aims to propagate them to all fields of the computer science.

The first part of our work targets the core of the P systems framework – its formal definition, which, surprisingly, was not always clear; moreover the introduction of new concepts like the *minimal parallelism* lead to different interpretations by different authors. We provide a single formal definition, which covers most of the classes of P systems with static structure known up to now.

The second part of Chapter 3 deals with one of most important models in the area of P systems – P systems using only communication, *i.e.*, the objects present in the system cannot evolve, but can only be moved from one compartment to another. We generalize the idea of co-transporters from the cell biology and end up with an interesting computational model based on the synchronization of signals. It is worth to note that the obtained model generalize all previous communication-only models of P systems.

The last part of Chapter 3 deals with two concrete problems. The first problem, the problem of the synchronization on a tree, is the generalization of the firing squad synchronization problem for cellular automata, where a linear com-

[1]We remark that other discrete models like Petri nets or process algebra are also used, each of them having advantages and limitations.

munication structure is replaced by a hierarchical one. The second problem, the construction of universal P systems of small size, is closely related to the problem of the construction of small size universal Turing machines, which is a fundamental problem of the computer science.

The last part of the thesis presents an exploratory topic, where we performed all the stages for the construction of a new model – we start with a biological phenomena, give its formalization and start theoretical investigations. The closeness of the obtained model to the initial biological subject permit us to affirm that results that are obtained theoretically, can be verified *in vitro*.

Outline

Chapter 1

This chapter contains elements of the theory of formal languages used in the thesis. It mostly gives the used definitions and fixes the notations.

Chapter 2

This chapter contains a study of the insertion-deletion operation. After an introduction and a formal definition given in Section 2.1, a description of formal proof methods is shown and a new proof technique is introduced. Section 2.3 considers context-free insertion-deletion systems and their links to the previous results in the formal language theory. After that one-sided insertion-deletion systems are considered. Section 2.5 investigates possible extensions of insertion-deletion systems in order to extend the computational power for non-complete classes. The results from this chapter are based on [51, 74, 54, 47, 48, 46, 49, 4, 5].

Chapter 3

This chapter studies various variants of P systems. We start in Section 3.1 with a general presentation of the model of P systems, its formal definition and the semantics. Section 3.2 overviews our results related to models of P systems using only communication. Last two sections contain more practical problems: Section 3.4 investigates the problem of synchronization in the case of P systems, while Section 3.5 investigates the construction of small universal P systems. The results from this chapter are partially based on [77, 16, 7, 76, 19, 26, 27, 24, 75, 3, 10, 6, 16, 8, 64, 79, 78, 22].

Chapter 4

This is an exploratory chapter that presents an original model based on the operation of length separation of molecules, usually performed by the *gel electrophoresis* in a bio-chemical laboratory. This chapter presents the first results and gives the possible insights for the future extensions. The results from this chapter are based on [18].

Chapter 5

This last chapter gives an overview of the further research directions and open problems raised by our research.

Chapter 1

Elements of formal language theory

In this chapter we fix the notations that we shall use in the future. For more details concerning the theory of formal languages we refer to [37] and [65].

1.1 Words, multisets and languages

We denote by $\mathbb{N}$ the set of natural numbers $\{0, 1, 2, \dots\}$ and by $\mathbb{N}^+$ the set of strictly positive integer numbers $\{1, 2, \dots\}$. We also denote by $\emptyset$ the empty set and by $\mathcal{P}(X)$ the set of all subsets of X. The number of elements of a set X is denoted by $Card(X)$.

An *alphabet* is a finite non-empty set of symbols which are also called *letters*. A *word* over the alphabet V is a concatenation of symbols of V. The empty concatenation is called the *empty word* and it is denoted by λ. The set of all words over V is denoted by V^*. The set of all words over an alphabet V, except the empty word, is denoted by V^+. Any subset of V^* is called a *language* over the alphabet V.

If $x = x_1x_2$ with $x_1, x_2 \in V^*$, then we say that the word x_1 is a *prefix* of x and that the word x_2 is a *suffix* of x. If $x = x_1x_2x_3$ with $x_1, x_2, x_3 \in V^*$, then x_2 is called factor of x. The set of prefixes, suffixes and factors of a word x is denoted by $Pref(x)$, $Suf(x)$ and $Fact(x)$, respectively.

The *length* of the word $x \in V^*$ is the number of symbols which appear in x and it is denoted by $|x|$. The number of occurrences of a symbol $a \in V$ in $x \in V^*$ is denoted by $|x|_a$. If $x \in V^*$ and $U \subseteq V$, the we denote by $|x|_U$ the number of occurrences of symbols from U in x. For a word $w \in V^*$ we define $Perm(w) = \{w' : |w'|_a = |w|_a \text{ for all } a \in V\}$. The length set of a language L is defined as $|L| = \{|x| : x \in L\}$. The length set of a family of languages $\mathcal{F}$ is defined analogously: $N\mathcal{F} = \{|L| : L \in \mathcal{F}\}$.

We denote boolean operations over languages in an ordinary way: the intersection by $\cap$, the union by $\cup$ or $+$ and the difference by $\setminus$.

The *concatenation* of two languages L_1 and L_2 is defined by:

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define the iteration of the concatenation as follows:

$$L^0 = \{\varepsilon\},$$
$$L_{i+1} = LL^i, \quad i \geq 0,$$
$$L^* = \bigcup_{i=0}^{\infty} L^i \quad \text{(Kleene closure)}.$$

A mapping $h : V \to U^*$, extended to $h : V^* \to U^*$ by $h(\lambda) = \{\lambda\}$ and $h(x_1x_2) = h(x_1)h(x_2)$, for $x_1, x_2 \in V^*$, is called a *morphism*. If $\lambda \notin h(a)$, for each $a \in V$, then h is a λ-free morphism.

A morphism $h : V^* \to U^*$ is called a *coding* if $h(a) \in U$ for each $a \in V$ and a *weak coding* if $h(a) \in U \cup \{\lambda\}$ for each $a \in V$. If $h : (V_1 \cup V_2)^* \to V_1^*$ is the morphism defined by $h(a) = a$ for $a \in V_1$, and $h(a) = \lambda$ otherwise, then we say that h is a *projection* (associated to V_1) and we denote it by pr_{V_1}. For a morphism $h : V^* \to U^*$, we define a mapping $h^{-1} : U^* \to \mathcal{P}(V^*)$ (and we call it an *inverse morphism*) by $h^{-1}(w) = \{x \in V^* \mid h(x) = w\}$.

For $x, y \in V^*$ we define their *shuffle* by

$$x \sqcup\!\sqcup\, y = \{x_1y_1 \dots x_ny_n \mid x = x_1 \dots x_n, y = y_1 \dots y_n,$$
$$x_i, y_i \in V^*, 1 \leq i \leq n, n \geq 1\}.$$

The *mirror image* of a string $x = a_1a_2 \dots a_n$, for $a_i \in V, 1 \leq i \leq n$, is the string $mi(x) = a_n \dots a_2a_1$.

In general, if we have an n-ary operation for strings, $g : V^* \times \dots \times V^* \to \mathcal{P}(U^*)$, we extend it to languages over V by

$$g(L_1, \dots, L_n) = \bigcup_{\substack{x_i \in L_i \\ 1 \leq i \leq n}} g(x_1, \dots, x_n).$$

For instance, $mi(L) = \{mi(x) \mid x \in L\}$.

We say that a family of languages $\mathcal{F}$ is closed with respect to an n-ary operation g if the result of the application of g on languages $L_1, \dots, L_n$ belonging to $\mathcal{F}$ also belongs to $\mathcal{F}$: $g(L_1, \dots, L_n) \in \mathcal{F}$.

The *Parikh image* of a language family $\mathcal{F}$ is a family of sets of vectors denoted by $Ps\mathcal{F}$ (we assume a fixed ordering on the alphabet $T = \{a_1, \dots, a_n\}$):

$$Ps(L) = \{(|w|_{a_1}, \dots, |w|_{a_n}) : w \in L\},$$
$$Ps\mathcal{F} = \{Ps(L) : L \in \mathcal{F}\}.$$

Let $O = \{a_1, \dots, a_k\}$ be an alphabet. A *finite multiset* M over O is a mapping $M : O \longrightarrow \mathbb{N}$, *i.e.*, for each $a \in O$, $M(a)$ specifies the number of occurrences of a in M. The size of the multiset M is $|M| = \sum_{a \in O} M(a)$. A multiset M over O can also be represented by any string x that contains exactly $M(a_i)$ symbols a_i for all $1 \leq i \leq k$, e.g., by $a_1^{M(a_1)} \dots a_k^{M(a_k)}$, or else by the set $\{a_i^{M(a_i)} : 1 \leq i \leq k\}$. For example, the multiset over $\{a, b, c\}$ defined by the mapping $a \to 3, b \to 1, c \to 0$ can be specified by a^3b or $\{a^3, b\}$. An empty multiset is represented by λ. The set of all finite multisets over the set V is denoted by $\langle V, \mathbb{N}\rangle$.

We may also consider mappings M of form $M : O \longrightarrow \mathbb{N} \cup \{\infty\}$, *i.e.*, elements of M may have an infinite multiplicity; we shall call them *infinite multisets*.

1.2 Formal grammars

A *formal grammar* is the following quadruplet $G = (N, T, S, P)$, where N and T are two disjoint alphabets, $S \in N$ is the axiom and P is a finite set of rewriting rules of the form $u \to v$, $u, v \in (N \cup T)^*$ with $|u|_N > 0$. The alphabet N, respectively T, is called the non-terminal, respectively terminal, alphabet of G. The rules of P are also called *productions*.

For $x, y \in (N \cup T)^*$ we write $x \Longrightarrow_G y$ if $x = x_1 u x_2$, $y = x_1 v x_2$ with $x_1, x_2 \in (N \cup T)^*$ and there is a production $u \to v \in P$.

In this case we say that x derive directly y. We can omit G if the context permits us to do so. We denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$. We write $x \Longrightarrow^k y$ if there are words $w_0, \ldots, w_k$ such that $w_i \Longrightarrow w_{i+1}$, $i \geq 0$ and $w_0 = x$ and $w_k = y$. We write $x \Longrightarrow^*_{P'} y$, where P' is a subset of P, if in order to obtain y starting from x we use productions only from P'.

Each word $w \in (N \cup T)^*$ derived from the axiom of the grammar G: $S \Longrightarrow^*_G w$ is called a *sentential form* of G.

The language generated by G, L(G), is defined by:

$$L(G) = \{w \in T^* : S \Longrightarrow^* w\}.$$

Depending on the form of its rules formal grammars may be of one of the following types.

- *Arbitrary grammars (of type 0):*
  Productions are of the form $u \to v$, where $u, v \in (N \cup T)^*$ and $|u|_N > 0$.

- *Context-sensitive grammars (of type 1):*
  Productions are of the form $u_1 A u_2 \to u_1 x u_2$, where $u_1, u_2 \in (N \cup T)^*$, $A \in N$ and $x \in (N \cup T)^+$. Also the production $S \to \lambda$ is allowed, where S does not appear at the right-hand side of any production from P.

- *Context-free grammars (of type 2):*
  Productions are of the form $A \to x$, where $A \in N$ and $x \in (N \cup T)^*$.

- *Regular grammars (of type 3):*
  Productions are either of form $A \to aB$ and $A \to a$, or of form $A \to Ba$ and $A \to a$, where $A, B \in N$ and $a \in T$.

We denote by *RE*, *CS*, *CF* and respectively *REG* the family of languages generated by arbitrary, context-dependent, context-free and regular grammars respectively. The families *RE*, *CS*, *CF* and *REG* respectively are called the family of recursively enumerable, context-sensitive, context-free and regular languages respectively. These families form the hierarchy of Chomsky. We also denote by *FIN* the family of finite languages. The following strict inclusions hold.

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

If Parikh images or length sets of corresponding families are considered, the following inclusions hold:

$$PsFIN \subset PsREG = PsCF \subset PsCS \subset PsRE.$$

$$NFIN \subset NREG = NCF \subset NCS \subset NRE.$$

The closure properties of the above families with respect to the operations on languages are shown in Tab. 1.1.

Table 1.1: Closure properties of families in the Chomsky hierarchy.

Operation	**RE**	**CS**	**CF**	**REG**
Union	Yes	Yes	Yes	Yes
Intersection	Yes	Yes	No	Yes
Concatenation	Yes	Yes	Yes	Yes
Kleene closure	Yes	Yes	Yes	Yes
Intersection with *REG*	Yes	Yes	Yes	Yes

The *Dyck language* D_n over $T_n = \{a_1, \overline{a}_1, \ldots, a_n, \overline{a}_n\}$, $n \geq 1$ is the context-free language generated by the grammar

$$G = (\{S\}, T_n, S, \{S \to \lambda, S \to SS\} \cup \{S \to a_i S \overline{a}_i \mid 1 \leq i \leq n\}).$$

Intuitively, the pairs $(a_i, \overline{a}_i)$, $1 \leq i \leq n$, can be viewed as parentheses, left and right, of different kinds. Then D_n consists of all strings of correctly nested parentheses. Sometimes it is convenient to define the Dyck language D over some alphabet V. In this case $n = card(V)$.

A *tag system* of degree $m > 0$, see [13, 56, 57], is the triplet $T = (m, V, P)$, where $V = \{a_1, \ldots, a_{n+1}\}$ is an alphabet and where P is a set of productions of form $a_i \to P_i, 1 \leq i \leq n$, $P_i \in V^*$. The symbol a_{n+1} is called a halting symbol. A configuration of the system T is a word w. We pass from the configuration $w = a_{i_1} \ldots a_{i_m} w'$ to the next configuration z by erasing the first m symbols of w and by adding P_{i_1} to the end of the word: $w \Longrightarrow z$, if $z = w' P_{i_1}$.

The computation of T over the word $x \in V^*$ is a sequence of configurations $x \Longrightarrow \ldots \Longrightarrow y$, where either $y = a_{n+1} a_{i_1} \ldots a_{i_{m-1}} y'$, or $y' = y$ and $|y'| < m$. In this case we say that T halts on x and that y' is the result of the computation of T over x. We say that T recognizes the language L if there exist a recursive coding φ such that for all $x \in L$, T halts on $\varphi(x)$, and T halts only on words from $\varphi(L)$.

We note that tag systems of degree 2 are able to recognize the family of recursively enumerable languages, see [13] and [56].

A context-free *matrix* grammar is a construct $G = (N, T, S, M)$, where N, T are disjoint alphabets (of nonterminals and terminals, respectively), $S \in N$ (axiom), and M is a finite set of matrices, that is, sequences of the form $(A_1 \to z_1, \ldots, A_n \to z_n)$, $n \geq 1$, of context-free rules over $N \cup T$. For a string x, an element $m = (r_1, \ldots, r_n)$ is executed by applying productions $r_1, \ldots, r_n$ one after the other, following the strict order they are listed in. The resulting string y is said to be directly derived from the original x and we write $x \Longrightarrow y$. Then, the generated language is defined in the usual way. The family of languages generated by context-free matrix grammars is denoted by MAT^λ (the superscript indicates that λ-rules are allowed); when using only λ-free rules, we denote the corresponding family by MAT.

A context-free *programmed* grammar is a construct $G = (N, T, S, P)$, where N, T, S are as above, the set of nonterminals, the set of terminals and the start symbol,

and P is a finite set of productions of the form $(b : A \to z, E, F)$, where b is a label, $A \to z$ is a context-free production over $N \cup T$, and E, F are two sets of labels of productions of G. (E is said to be the *success field*, and F is the *failure field* of the production.) A production of G is applied as follows: if the context-free part can be successfully executed, then it is applied and the next production to be executed is chosen from those with the label in E, otherwise, we choose a production labeled by some element of F, and try to apply it. More precisely, the configuration is the couple (x, i), where x is a word and i is a label of a production $(i : A \to z, E, F)$ from P. The transition $(x, i) \Longrightarrow (y, j)$ is performed if either $x \Longrightarrow_{A \to z} y$ and $j \in E$, or $x = y$, $|x|_A = 0$ and $j \in F$. In the latter case we say that the production is applied in *appearance checking* mode. This type of programmed grammars is said to be with appearance checking; if no failure field is given for any of the productions, then a programmed grammar without appearance checking is obtained.

A *graph-controlled* grammar is the tuple $G = (N, T, S, P, I, F)$, where N, T, S and P are defined as above and $I, F \subseteq Lab(R)$ are sets of labels of rules from R. The computation in a graph-controlled grammar follows the same principles as in the programmed grammar (N, T, S, P), however the starting label can be only one of labels from the set I, while the terminal string can be collected only when the current label is from F:

$$L(G) = \{y \in T^* \mid \exists i \in I, j \in F : (S, i) \Longrightarrow^* (y, j)\}.$$

1.3 Finite automata and Turing machines

A finite automaton is the quintuplet $M = (Q, V, q_0, F, \delta)$, where Q is a finite set of *states*, V is a finite set of symbols, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and where $\delta : Q \times V \to \mathcal{P}(Q)$ is the transition function of the automaton. We define the transition $\vdash$ in an ordinary way: $(s, ax) \vdash (s', x)$ if $s' \in \delta(s, a)$, where $s, s' \in Q$, $a \in V$ and $x \in V^*$. We denote by $\vdash^*$ the reflexive and transitive closure of $\vdash$.

We say that the word x is accepted by M if $(q_0, x) \vdash^* (q, \varepsilon)$ and $q \in F$. The language accepted by M is:

$$L(M) = \{x \in V^* : (q_0, x) \vdash^* (q, \varepsilon), q \in F\}.$$

We also use a graphic notation in order to define a finite automaton. In this case the finite automaton will be represented by a finite oriented graph whose nodes represent the states of the automaton and whose edges are defined by the transition function of the automaton. More exactly, there is an arc labeled by a between vertices q_i and q_j of the graph if $q_j \in \delta(q_i, a)$. The final states are enclosed by a double circle.

A Turing machine is the 6-tuple $M = (Q, T, a_0, q_0, F, \delta)$, where Q is a finite set of states, T is the tape alphabet, $a_0 \in T$ is the blanc symbol, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and where δ is a transition function. Every rule of δ, also called instruction, is of form $q_i a_k D a_l q_j$, where $q_i, q_j \in Q$, $a_k, a_l \in T$ and where $D \in \{L, S, R\}$. The semantic of the rule $q_i a_k D a_l q_j$ is the following: if being in the state q_i the head of the machine sees the symbol a_k on the tape, then it changes its state to q_j, replaces a_k by a_l and moves to the left if $D = L$, to the right if $D = R$ or remains in the same position if $D = S$. We note that for any Turing machine

which has stationary instructions, *i.e.* without a move, it is possible to construct an equivalent machine which will have no stationary instruction.

A *configuration* of the Turing machine M is the following word $w_1 q_i a_k w_2$, where $w_1 a_k w_2$ is the part of the tape which is not empty, q_i is the state of the machine and a_k is the cell which is examined by the head of the machine.

A *computation* of the Turing machine M on the word $x \in T^+$ is a sequence of configurations $q_0 x \Longrightarrow \ldots w_1 q_f w_2$, where $q_f \in F$. The initial configuration may be of form $u_1 q_0 u_2$, where $x = u_1 u_2$, but we may suppose without loosing generality that it is of form $q_0 x$. In this case we say that $w_1 w_2$ is the result of computation of M on the word x.

An important notion is the *universal* Turing machine. Such a machine is a fixed object $\mathfrak{M}_u$ capable to simulate the work of any Turing machine M on an input w providing that a suitable encoding of M and w are given as an input to $\mathfrak{M}_u$.

Let us stress here an important distinction between *computational completeness* and *universality*. Given a class C of computability models, we say that C is computationally complete if the devices in C can characterize the power of Turing machines (or of any other type of equivalent devices). This means that given a Turing machine M one can find an element C in C such that C is equivalent with M. Thus, completeness refers to the capacity of covering the level of computability (in grammatical terms, this means to generate all recursively enumerable languages). Universality is an internal property of C and it means the existence of a fixed element $\mathfrak{U}$ of C which is able to simulate any given element E (of C), providing that an appropriate encoding of E and of the input is given. If C does not have a universal element, then there is a family $C' \supset C$ that contains an element $\mathfrak{U}$ universal for C. We note that C' is not necessarily universal for RE.

1.4 Register machines

Register machines were introduced in [56], see also [23].

A *register machine* with k registers is a 5-tuple $M = (Q, R, q_0, q_f, P)$, where

- Q is a finite non-empty set, called the set of *states*,
- $R = \{A_1, \ldots, A_k\}$, $k \geq 1$, is a set of *registers*,
- $q_0 \in Q$ is the *initial state*, and
- $q_f \in Q$ is the *final state*,
- P is a set of *instructions* of the following forms:
  - $(p, A+, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called an *increment instruction* or
  - $(p, A-, q, s)$, where $p, q, s \in Q, p \neq q_f, A \in R$, called a *decrement instruction*.

Furthermore, for every $p \in Q$, $(p \neq q_f)$, there is exactly one instruction of the form either $(p, A+, q, s)$ or $(p, A-, q, s)$.

According to [45], we will also use the notations $(p, [RiP], q, s)$ and $(p, \langle RiZM \rangle, q, s)$ instead of (p, A_i+, q, s) and (p, A_i-, q, s), respectively.

For generating languages over T, we use the model of a *register machine with output tape* (introduced in [56]), which also uses a tape operation:

$(p, WRITE(A), q)$, with $p, q \in Q$, $A \in T$.

A *configuration* of a register machine M, defined above, is given by a $k+1$-tuple $(q, m_1, \ldots, m_k)$, where $q \in Q$ and $m_1, \ldots, m_k$ are non-negative integers, q corresponds to the currents state of M and $m_1, \ldots, m_k$ are the current numbers stored in the registers (in other words, the current contents of the registers or the value of the registers) $A_1, \ldots A_k$, respectively.

A transition of the register machine consists in updating the number stored in a register and by changing the current state to another one, according to an instruction.

An increment instruction $(p, A+, q, s) \in P$ is performed if M is in state p, the number stored in register A is increased by 1, and after that M enters either state q or state s, chosen non-deterministically.

A decrement instruction $(p, A-, q, s) \in P$ is performed if M is in state p, and if the number stored in register A is positive, then it is decreased by 1 and then M enters state q, and if the number stored in A is 0, then the contents of A remains unchanged and M enter state s.

We say that a register machine $M = (Q, R, q_0, q_f, P)$, with k registers, given as above, *generates* a non-negative integer n if starting from the *initial configuration* $(q_0, 0, 0, \ldots, 0)$ it enters the *final configuration* $(q_f, n, 0, \ldots, 0)$.

The set of non-negative integers generated by M is denoted by $N(M)$.

It is known that register machines generate all recursively enumerable sets of non-negative integers [56]. If the WRITE instruction is used, then *RE* can be generated.

In the case when a register machine cannot check whether a register is empty we say that it is partially blind; the second type of instructions is then written as (p, A_k-, q) and the transition is undefined if register k is zero.

The word "partially" stands for an implicit test for zero at the end of a (successful) computation: counters $m+1, \cdots, d$ should be empty. It is known, that partially blind register machines generate exactly *PsMAT*.

1.5 Splicing operation

Now we briefly recall the basic notions concerning the splicing operation and related constructs [61].

A *splicing rule* (over an alphabet V) is a 4-tuple (u_1, u_2, u_3, u_4) where $u_1, u_2, u_3, u_4 \in V^*$. It is frequently written as $u_1\#u_2\$u_3\#u_4$, $\{\$, \#\} \notin V$, or in two dimensions as $\begin{array}{c|c} u_1 & u_2 \\ \hline u_3 & u_4 \end{array}$. Strings u_1u_2 and u_3u_4 are called splicing *sites*.

We say that a word x *matches* rule r if x contains an occurrence of one of the two sites of r. We also say that x and y are *complementary* with respect to a rule r if x contains one site of r and y contains the other one. In this case we also say that x or y may *enter* rule r. When x and y can enter a rule $r = u_1\#u_2\$u_3\#u_4$, *i.e.*, we have $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, it is possible to define the application of r to couple x, y. The result of this application is w and z where $w = x_1u_1u_4y_2$ and $z = y_1u_3u_2x_2$. We also say that x and y are spliced and w and z are the result of this splicing. We write this as follows: $(x, y) \vdash_r (w, z)$ or

$(x_1u_1|u_2x_2, y_1u_3|u_4y_2) \vdash_r (x_1u_1u_4y_2, y_1u_3u_2x_2)$.

The pair $h = (V, R)$ where V is an alphabet and R is a finite set of splicing rules is called a *splicing scheme* or an *H scheme*.

For a splicing scheme $h = (V, R)$ and for a language $L \subseteq V^*$ we define:
$\sigma_h(L) \stackrel{\text{def}}{=} \{w, z \in V^* \mid \exists x, y \in L, \exists r \in R : (x, y) \vdash_r (w, z)\}$.

Now we can introduce the iteration of the splicing operation.
$\sigma_h^0(L) = L$,
$\sigma_h^{i+1}(L) = \sigma_h^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0$,
$\sigma_h^*(L) = \cup_{i\geq 0}\sigma_h^i(L)$.

It is known that the iterated splicing preserves the regularity of a language:

Theorem 1.5.1. [61] *Let $L \subseteq V^*$ be a regular language and let $h = (V, R)$ be a splicing scheme. Then language $\sigma_h^*(L)$ is regular.*

A *Head-splicing-system* [35, 36], or *H system*, is a construct:

$$\mathcal{H} = (h, A) = ((V, R), A),$$

which consists of an alphabet V, a finite set $A \subseteq V^*$ of initial words over V, the *axioms*, and a finite set $R \subseteq V^* \times V^* \times V^* \times V^*$ of splicing rules. System $\mathcal{H}$ is called finite if A and R are finite sets.

The language generated by an H system $\mathcal{H}$ is defined as $L(\mathcal{H}) \stackrel{\text{def}}{=} \sigma_h^*(A)$.

Thus, the language generated by H system $\mathcal{H}$ is the set of all words that can be generated in $\mathcal{H}$ starting with A as initial words by iteratively applying splicing rules to copies of the words already generated.

Theorem 1.5.2. *The following relations hold:*

(1) For any H system $\mathcal{H}$, $L(\mathcal{H}) \in REG$.

(2) For any $L \in REG$ there exists an H system $\mathcal{H}$ and an alphabet T such that $L = L(\mathcal{H}) \cap T^*$.

Chapter 2

Study of insertion-deletion systems

This chapter focuses on the study of the operations of *insertion* and *deletion*. In general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string being in a specified (left and right) context. An insertion or deletion rule is defined by a triple (u, x, v) meaning that x can be inserted between u and v or deleted if it is between u and v. Thus, an insertion corresponds to the rewriting rule $uv \to uxv$ and a deletion corresponds to the rewriting rule $uxv \to uv$. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion or deletion operations as defined by the given rules one gets a language. The size of the alphabet, the number of axioms, the size of contexts and of the inserted or deleted string are natural descriptional complexity measures for insertion-deletion systems.

The idea of insertion of one string into another was firstly considered with a linguistic motivation in [50] and latter developed in [30, 59]. Marcus contextual grammars investigated in above references consider couples $(x, (u, v))$, meaning that words u and v can be adjoined to the word x. This corresponds in some sense to grammars having rules of type $x \to uxv$, *i.e.*, u and v are inserted around the position marked by x. Such grammars are alternative concepts to Chomsky grammars and present the evolution of the descriptive linguistics. Many interesting linguistic properties like ambiguity and duplication can be captured in this framework. The insertion of a string in a specified context was firstly considered in [30].

In [33, 34] the insertion operation and its iterated variant is introduced with a different motivation. The author considers this operation as generalization of Kleene's operations of concatenation and closure [43]. The operation of concatenation would produce a string x_1x_2y from two strings x_1x_2 and y. By allowing the concatenation to happen anywhere in the string and not only at its right extremity a string x_1yx_2 can be produced, *i.e.*, y is inserted into x_1x_2. In [38] the deletion is defined as a right quotient operation which happens not necessarily at the rightmost end of the string. In the same thesis the duality between the insertion and deletion is also highlighted: any insertion system generating a language $\mathcal{L}$ is at the same time a deletion system recognizing $\mathcal{L}$. The operations considered in above works correspond to context-free variants of insertion and deletion operations, be-

cause no contexts are used. In the same place several other variants of insertion and deletion are introduced and their closure properties are investigated.

The third inspiration for insertion and deletion operations comes, surprisingly, from the field of molecular biology. In fact they correspond to a mismatched annealing of DNA sequences. Insertion and deletion can be performed, at leat theoretically, as follows (for biological terminology see [61, 2]). Let us imagine that there is a test tube with a single stranded DNA sequence $5' - x_1uvx_2z - 3'$. If one adds to the test tube a single stranded DNA sequence $3' - \overline{u}\overline{y}\overline{v} - 5'$, where $\overline{u}$,$\overline{v}$ are the Watson-Crick complements of strings u,v then the two strings will anneal (u will stick to $\overline{u}$ and v will stick to $\overline{v}$, folding y, see Fig. 2.1(b). Now one can cut the sequence uv obtaining the structure depicted on Fig. 2.1(c). Adding a primer $\overline{z}$ and the polymerase the complete double-stranded sequence is obtained, see Fig. 2.1(d). Finally, melting the solution the strands are separated leading to situation depicted on Fig. 2.1(e), meaning that y was inserted between u and v.

By a similar mismatched annealing one can, theoretically, perform a deletion operation, taking uyv in the starting string and adding $\overline{u}\overline{v}$. The process is illustrated on Fig. 2.2 (in order to go from step (b) to step (c) a polymerization and removing of the loop by a restriction enzyme is done).

Therefore, insertion and deletion can be performed in the DNA framework. Such operations are also present in the evolution processes under the form of point mutations as well as in RNA editing, see the discussions in [9], [67] and [61]. This biological motivation of insertion-deletion operations lead to their study in the framework of molecular computing, see, for example, [20], [39], [61], [69].

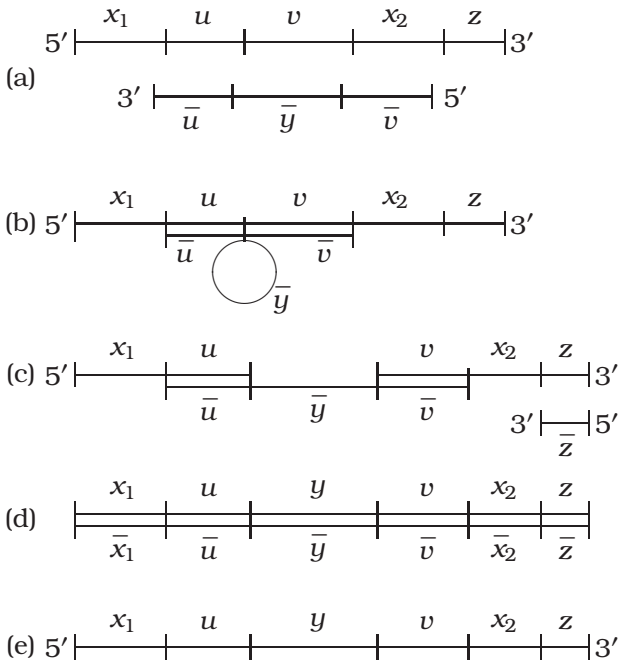


Figure 2.1: Inserting by mismatched annealing

Insertion-deletion systems are quite powerful, leading to characterizations of recursively enumerable languages. The proof of such characterizations was usually

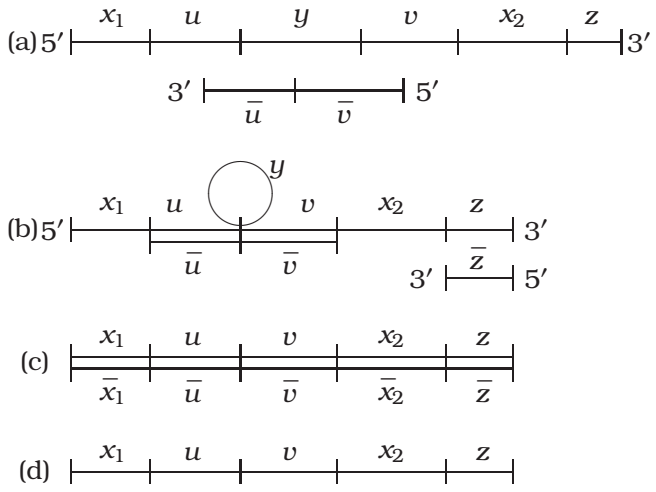


Figure 2.2: Deleting by mismatched annealing

done by showing that the corresponding class of insertion-deletion systems can simulate an arbitrary Chomsky grammar. However, the obtained constructions are quite complicated and specific to the used class of systems. We introduced a new method of such computational completeness proofs, which relies on a direct simulation of one class of insertion-deletion systems by another. The obtained method is quite generic and it was used to prove the computational completeness of 8 classes of insertion-deletion systems, the proof being similar for all cases.

Another important point bridging our work with investigations done in [33] and [38] is the proof that context-free insertion-deletion systems are computationally complete. This provides a new characterization of recursively enumerable languages, where every such language can be represented as a reflexive and transitive closure of the insertion-deletion of two finite languages.

We started a systematical investigation of classes of insertion-deletion systems with respect to the size of contexts and inserted/deleted strings and we showed for the first time that there are classes that are not computationally complete, moreover some of them are decidable. In these cases it is possible to consider a graph-controlled variant of insertion-deletion systems, known under the name of insertion-deletion P systems, which permits in most of the cases to increase the computational power of corresponding systems.

2.1 Formal definition

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V, and I, D are finite sets of triples of the form (u, a, v), $a \neq \lambda$, where u and v are strings over V. The elements of T are *terminal* symbols (in contrast, those of $V - T$ are called nonterminals), those of A are *axioms*, the triples in I are *insertion rules*, and those from D are *deletion rules*. An insertion rule $(u, a, v) \in I$ indicates that the string a can be inserted between u and v, while a deletion rule $(u, a, v) \in D$ indicates that a can be removed from the context (u, v).

As stated otherwise, $(u, a, v) \in I$ corresponds to the rewriting rule $uv \to uav$, and $(u, a, v) \in D$ corresponds to the rewriting rule $uav \to uv$. We denote by $\Longrightarrow_{ins}$ the relation defined by an insertion rule (formally, $x \Longrightarrow_{ins} y$ iff $x = x_1uvx_2, y = x_1uavx_2$, for some $(u, a, v) \in I$ and $x_1, x_2 \in V^*$) and by $\Longrightarrow_{del}$ the relation defined by a deletion rule (formally, $x \Longrightarrow_{del} y$ iff $x = x_1uavx_2, y = x_1uvx_2$, for some $(u, a, v) \in D$ and $x_1, x_2 \in V^*$). We refer by $\Longrightarrow$ to any of the relations $\Longrightarrow_{ins}, \Longrightarrow_{del}$, and denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$ (as usual, $\Longrightarrow^+$ is its transitive closure).

The language generated by *ID* is defined by

$$L(ID) = \{w \in T^* \mid x \Longrightarrow^* w, x \in A\}.$$

The complexity of an insertion-deletion system $ID = (V, T, A, I, D)$ is described by the vector $(n, m, m'; p, q, q')$ called *size*, where

$$\begin{aligned} n &= \max\{|a| \mid (u, a, v) \in I\}, & p &= \max\{|a| \mid (u, a, v) \in D\}, \\ m &= \max\{|u| \mid (u, a, v) \in I\}, & q &= \max\{|u| \mid (u, a, v) \in D\}, \\ m' &= \max\{|v| \mid (u, a, v) \in I\}, & q' &= \max\{|v| \mid (u, a, v) \in D\}. \end{aligned}$$

We also denote by $INS_n^{m,m'}DEL_p^{q,q'}$ corresponding families of insertion-deletion systems. Moreover, we define the total size of the system as the sum of all numbers above: $\psi = n + m + m' + p + q + q'$.

If some of the parameters n, m, m', p, q, q' is not specified, then we write instead the symbol $*$. In particular, $INS_*^{0,0}DEL_*^{0,0}$ denotes the family of languages generated by *context-free insertion-deletion systems*. If one of numbers from the couples m, m' and/or q, q' is equal to zero (while the other is not), then we say that corresponding families have a one-sided context.

We remark that, historically, another complexity measure called *weight* was used for insertion-deletion systems. It corresponds to 4-tuples $(n, \overline{m}; p, \overline{q})$, where $\overline{m} = \max\{m, m'\}$ and $\overline{q} = \max\{q, q'\}$.

2.2 Basic simulation principles

In this section we show some important properties of insertion-deletion systems, present some normal forms and indicate basic methods for equivalence proofs used in the rest of the chapter.

We start with the presentation of the normal form for insertion-deletion systems.

Definition 2.2.1. An insertion-deletion system $ID = (V \cup \{\$\}, T, A, I, D \cup D_2)$ of size $(n, m, m'; p, q, q')$ is said to be in the *normal form* if

- for any $(u, x, v) \in I$ it holds $|u| = m$, $|v| = m'$, $|x| = n$,
- for any $(u, x, v) \in D$ it holds $|u| = q$, $|v| = q'$, $|x| = p$,
- for any $(u, x, v) \in D$ it holds that x does not contain letters from T,
- the set D_2 is defined as $D_2 = \{(\lambda, \$, \lambda)\}$.

Theorem 2.2.2. *For any insertion-deletion system ID it is possible to construct a system ID′ in normal form and having same size such that $L(ID) = L(ID')$.*

This affirmation is quite obvious. For the first two conditions it is enough to replace any rule having left or right contexts of a smaller size by a group of rules, where the left (resp. right) context is a string over $V \cup \{\$\}$ of the required size. The same holds for the inserted or deleted symbol and axioms. More precisely, the new symbol \$ permits to fill the context of rules and sizes of axioms up to the desired size.

The third condition can be satisfied as follows. For any terminal symbol $t \in T$ a special non-terminal N_t is considered. All rules and axioms involving t are duplicated and t replaced by N_t. This construction ensures that symbol N_t acts like an alias for the symbol t, *i.e.* for any derivation producing $w_1 t w_2$ there is another derivation producing $w_1 N_t w_2$. Hence there is no difference between erasing t or N_t, therefore all deletion rules involving t can be omitted. A formal proof of the theorem can be found in [5].

Example 2.2.3.

Consider the insertion-deletion system $ID = (\{a, b, C\}, \{a, b\}, \{ab\}, I, D)$ of size $(2, 1, 1; 2, 1, 1)$, where $I = \{(a, aC, b), (a, b, C)\}$ and $D = \{(b, C, b)\}$. Then ID' can be defined as follows: $ID' = (\{a, b, C, \$\}, \{a, b\}, \{ab \sqcup\!\sqcup \$\$\}, I', D' \cup \{(\lambda, \$, \lambda)\})$, where $I' = \{(a, aC, b), (a, \$b, C), (a, b\$, C)\}$ and $D' = \{(b, C\$, b), (b, \$C, b)\}$.

Insertion-deletion systems represent a powerful model of computation. If the size of the system is not bounded, then an arbitrary grammar can be simulated.

Theorem 2.2.4. *For any type-0 grammar $G = (N, T, S, P)$ there is an insertion-deletion system $ID = (V, T, A, I, D)$ such that $L(G) = L(ID)$.*

Proof. Let $V = N \cup \{\#_i : 1 \leq i \leq |P|\} \cup \{\$\}$. Let $k_1 = \max\{|u|, u \to v \in P\}$ and $k_2 = \max\{|v|, u \to v \in P\}$. Consider $k = \max(k_1, k_2)$. The set A is defined as $A = \{\$^k S \$^k\}$.

For any rule $i : u \to v \in P$ we add insertion rules $(xu, \#_i v, y)$, $x, y \in (N \cup \{\$\})^*$, $|xu| = k$, $|y| = k$, to I and a deletion rule $(x, u\#_i, v)$, $x \in N \cup \{\$\}$ to D. Finally, a rule $(\lambda, \$, \lambda)$ is added to D.

It is not difficult to see that such system simulates G. Indeed, for any derivation $w_1 u w_2 \Longrightarrow w_1 v w_2$ in G there is a following two-step derivation $\$^k w_1 u w_2 \$^k \Longrightarrow \$^k w_1 u \#_i v w_2 \$^k \Longrightarrow \$^k w_1 v w_2 \$^k$ in ID that simulates the corresponding production of G. If $w \in L(G)$ then the string $\$^k w \$^k$ will be obtained in ID. Additional symbols \$ can be deleted at this moment. So $w \in L(ID)$.

For the converse inclusion it is enough to observe that if an insertion rule $(xu, \#_i v, y)$ is used, then no more insertions inside the corresponding site xu can be done. So, the only way to eliminate the symbol $\#_i$ is to perform the corresponding deletion. Hence the computation in ID can be rearranged in such a way that an insertion is followed by the corresponding deletion. This corresponds to a derivation step in G, which completes the proof. □

As one can see from the previous theorem, the basic idea of grammar simulation by insertion-deletion systems is a construction of a set of related insertion and deletion rules that shall be used in some specified sequence thus performing a grammar rule simulation. Usually, insertion rules introduce new non-terminal symbols in the string which can be deleted only by corresponding deletion rules (like symbols $\#_i$ in theorem above). If the correct sequence is not performed, then

some non-terminal symbols that cannot be deleted will remain in the string. In the subsequent sections different variants of this method are shown permitting to decrease the size of used insertion and deletion rules.

2.2.1 The method of direct simulation

A simulation of type-0 grammars by insertion-deletion systems is the main method permitting to prove the computational completeness of insertion-deletion systems. However, when several such results are established, it is much easier to prove the computational completeness by simulating another insertion-deletion systems. For example:

Theorem 2.2.5. $INS_1^{1,1} DEL_2^{0,0} = RE$.

Sketch of Proof. The proof may be done by simulating insertion-deletion systems of size $(1,1,1;1,1,1)$ which are known to be computationally complete, see [69, 70]. In this case it is enough to show how a deletion rule (a,b,c), $a,b,c \in V$ can be simulated using insertion and deletion rules of size $(1,1,1;2,0,0)$. Let $a \neq b \neq c$. Then a deletion rule (a,b,c) with label i may be simulated by a sequence of the following rules: $\{(a,\}_i,b),(b,]_i,c),(a,[_i,\}_i),([_i,\{_i,\}_i),([_i,K_i,\{_i,)\} \subseteq I$ and $(\lambda,\{_i\}_i,\lambda),(\lambda,K_ib,\lambda),(\lambda,[_i]_i,\lambda) \subseteq D$. The simulation is performed as follows (we underline the inserted symbols):

$$w_1abcw_2 \Longrightarrow_{ins} w_1a\underline{\}_i}bcw_2 \Longrightarrow_{ins} w_1a\underline{[_i}\}_ibcw_2 \Longrightarrow_{ins} w_1a[_i\}_i\underline{b]_i}cw_2 \Longrightarrow_{ins}$$
$$\Longrightarrow_{ins} w_1a[_i\underline{\{_i}\}_ib]_icw_2 \Longrightarrow_{ins} w_1a[_i\underline{K_i}\{_i\}_ib]_icw_2 \Longrightarrow_{del}$$
$$\Longrightarrow_{del} w_1a[_iK_ib]_icw_2 \Longrightarrow_{del} w_1a[_i]_icw_2 \Longrightarrow_{del} w_1acw_2.$$

The idea behind the simulation is the following. Symbols $[_i$ and $]_i$ delimit the deletion site. Symbol K_i performs the deletion of b, while symbols $\}_i$ and $\{_i$ ensure that K_i is inserted only once after $[_i$ (hence only one b can be deleted). If all the above steps are not performed, then some of additional symbols will remain in the string, hence it will never become terminal. This is a common method of simulation: the working (insertion or deletion) site is delimited by special symbols in order to avoid interactions between several such sites and inside the site the sequence of insertions and deletions permits to simulate exactly one application of the corresponding rule. All additional symbols are related in such a way that the whole sequence of insertions and deletions shall be performed in order to eliminate all of them.

We remark that it would be wrong to simulate a deletion rule (a,b,c) by only rules $\{(a,[_i,b),(b,]_i,c),([_i,K_i,b)\} \subseteq I$ and $(\lambda,K_ib,\lambda),(\lambda,[_i]_i,\lambda) \subseteq D$, because it is possible to erase several symbols b, which leads to a wrong computation:

$$w_1abbcw_2 \Longrightarrow_{ins} w_1a\underline{[_i}bbcw_2 \Longrightarrow_{ins} w_1a[_ibb\underline{]_i}cw_2 \Longrightarrow_{ins} w_1a[_i\underline{K_i}bb]_icw_2 \Longrightarrow_{del}$$
$$\Longrightarrow_{del} w_1a[_ib]_icw_2 \Longrightarrow_{ins} w_1a[_i\underline{K_i}b]_icw_2 \Longrightarrow_{del} w_1a[_i]_icw_2 \Longrightarrow_{del} w_1acw_2.$$

□

The above approach is very powerful and it permits to establish the computational completeness of the corresponding class of insertion-deletion systems in a

much easier way. For example, the proof of Theorem 2.2.5 in [61] (Theorem 6.3) takes more than two pages. The method is quite generic, in order to use it one should find a computational complete class of insertion-deletion systems having same insertion or deletion parameters. Then, in order to prove the computational completeness, it is sufficient to simulate corresponding deletion or insertion operation. This is significantly easier than the simulation of a Chomsky grammar because only left-hand or only right-hand side of a production $u \to v$ shall be simulated.

Most of our results about the universality of insertion-deletion systems are obtained using this technique.

2.3 Context-free insertion-deletion systems

In this section we study an important class of insertion-deletion systems: systems with context-free rules. This bridges our work with early investigations from [33] and [38] and answers old questions from this area.

2.3.1 Computational completeness results

We start with the proof of computational completeness of context-free insertion-deletion systems. We give here the sketch of the proof. More details can be found in [51].

Theorem 2.3.1. $INS_*^{0,0}DEL_*^{0,0} = RE$.

Sketch of proof. Let $G = (N, T, S, P)$ be type-0 Chomsky grammar where N, T are disjoint alphabets, $S \in N$, and P is a finite subset of rules of the form $u \to v$ with $u, v \in (N \cup T)^*$ and u contains at least one letter from N. We assume all rules from P labeled in a one-to-one manner with elements of a set M, disjoint of $N \cup T$.

We construct the following context-free insertion-deletion system: $\gamma = (N \cup T \cup M, T, \{S\}, I, D)$, where

$$I = \{(\lambda, vR, \lambda) \mid R : u \to v \in P,\ R \in M,\ u, v \in (N \cup T)^*\},$$
$$D = \{(\lambda, Ru, \lambda) \mid R : u \to v \in P,\ R \in M,\ u, v \in (N \cup T)^*\}.$$

Two rules $(\lambda, vR, \lambda) \in I, (\lambda, Ru, \lambda) \in D$ as above are said to be *M-related*.

We have the equality $L(G) = L(\gamma)$.

The inclusion $L(G) \subseteq L(\gamma)$ is obvious: each derivation step $x_1 ux_2 \Longrightarrow x_1 vx_2$, performed in G by means of a rule $R : u \to v$, can be simulated in γ by an insertion operation step $x_1 ux_2 \Longrightarrow_{ins} x_1 vRux_2$ which uses the rule $(\lambda, vR, \lambda) \in I$, followed by the deletion operation $x_1 vRux_2 \Longrightarrow_{del} x_1 vx_2$ which uses the rule $(\lambda, Ru, \lambda) \in D$.

Consider now the inclusion $L(\gamma) \subseteq L(G)$. The idea of the proof is to transform any terminal derivation in γ into one in which any two consecutive (odd, even) derivations steps simulate one production in G. Because the labels of rules from P precisely identify a pair of M-related insertion-deletion rules, and the elements of M are nonterminal symbols for γ, every terminal derivation with respect to γ must involve the same number of insertion steps and deletion steps; moreover, these steps are performed by using pairs of M-related rules from I and D.

For every terminal derivation in γ it is possible to construct an equivalent derivation, using the same rules in a different order, and having only matching pairs of consecutive rules, *i.e.* odd steps $w_i \Longrightarrow_{ins} w_{i+1}$ are performed by a rule $(\lambda, vR, \lambda) \in I$, while even steps $w_{i+1} \Longrightarrow w_{i+2}$ are performed by using the M-related rule $(\lambda, Ru, \lambda) \in D$. Clearly, two consecutive steps of a derivation in γ which use M-related rules $(\lambda, vR, \lambda) \in I, (\lambda, Ru, \lambda) \in D$, correspond to a derivation step in G which uses the rule $R : u \to v$. This implies the inclusion $L(\gamma) \subseteq L(G)$. □

The context control of a type 0 grammar does not really disappear in the corresponding insertion-deletion system (as constructed in Theorem 2.3.1 above). It rather changes its form, becoming a rigid synchronization of insertions and deletions. In other terms, if a word u represents the context of a word v in a "context-sensitive production" $R : u \to v$, then in the corresponding insertion-deletion system the word v will also be conditioned by the later occurrence of u in a successful derivation (hence u is yet again the context of v). This condition is enforced by the newly introduced symbol R which acts as a "remote context binder". The fact that the context u "seems" to appear after the context-controlled v is of no importance, reflecting the reversal of generative process of the grammar.

We illustrate the construction from the proof with a simple example.

Example 2.3.2.

Consider the context-sensitive grammar $G = (\{S, X, Y\}, \{a, b, c\}, S, P)$ with the set of productions

$$\begin{aligned} P \;=\; & \{R_1 : S \to aSX,\ R_2 : S \to aY,\ R_3 : YX \to bYc, \\ & R_4 : cX \to Xc,\ R_5 : Y \to bc\}. \end{aligned}$$

It is easy to see that this grammar generates the non-context-free language $L(G) = \{a^i b^i c^i \mid i \geq 1\}$.

The obtained system is

$$\begin{aligned} \gamma &= (V, \{a, b, c\}, \{S\}, I, D), \text{ where} \\ V &= \{S, X, Y, a, b, c, R_1, R_2, R_3, R_4, R_5\}, \\ I &= \{(\lambda, aSXR_1, \lambda),\ (\lambda, aYR_2, \lambda),\ (\lambda, bYcR_3, \lambda), \\ &\quad (\lambda, XcR_4, \lambda),\ (\lambda, bcR_5)\}, \\ D &= \{(\lambda, R_1S, \lambda),\ (\lambda, R_2S, \lambda),\ (\lambda, R_3YX, \lambda), \\ &\quad (\lambda, R_4cX, \lambda),\ (\lambda, R_5Y, \lambda)\}. \end{aligned}$$

Consider a derivation for the word $a^3b^3c^3$ in grammar G:

$$\begin{aligned} S \Longrightarrow aSX \Longrightarrow aaSXX \Longrightarrow aaaYXX \Longrightarrow aaabYcX \Longrightarrow \\ \Longrightarrow aaabYXc \Longrightarrow aaabbYcc \Longrightarrow aaabbbccc. \end{aligned}$$

One of the corresponding derivations in γ is as follows:

$$\begin{aligned} S \Longrightarrow_{ins} \underline{aSXR_1}S \Longrightarrow_{ins} a\underline{aSXR_1}SXR_1S \Longrightarrow_{ins} aa\underline{aYR_2}SXR_1SXR_1S \Longrightarrow_{del} aaaYR_2SXXR_1S \Longrightarrow_{del} \\ \Longrightarrow_{del} aaaYXXR_1S \Longrightarrow_{ins} aaa\underline{bYcR_3}YXXR_1S \Longrightarrow_{del} \Longrightarrow_{del} aaabYcXR_1S \Longrightarrow_{ins} aaabY\underline{XcR_4}cXR_1S \\ \Longrightarrow_{ins} aaabb\underline{YcR_3}YXcR_4cXR_1S \Longrightarrow_{del} aaabbYcR_3YXcR_1S \Longrightarrow_{del} aaabbYccR_1S \Longrightarrow_{ins} \\ \Longrightarrow_{ins} aaabbbc\underline{R_5}YccR_1S \Longrightarrow_{del} aaabbbcR_5Ycc \Longrightarrow_{del} aaabbbccc. \end{aligned}$$

Let us denote by $L \leftrightarrows^{L_1}_{L_2}$ the operation of insertion-deletion that inserts words from L_1 into L or deletes words belonging to L_2 from L and by $L \leftrightarrows^{*}{}^{L_1}_{L_2}$ its reflexive and transitive closure. Then the following representation of RE holds:

Theorem 2.3.3. *Any language $L \in RE$ can be represented in the following form $L = \left(\{S\} \leftrightarrows^* \begin{smallmatrix} L_1 \\ L_2 \end{smallmatrix}\right) \cap T^*$, where L_1 and L_2 are two finite languages and T is an alphabet.*

In the proof of Theorem 2.3.1, the length of inserted or deleted strings is not bounded, but a bound can be easily found by controlling the length of strings appearing in the rules of the starting type-0 grammar:

Theorem 2.3.4. $INS_3^{0,0}DEL_3^{0,0} = RE$.

Proof. Let $G = (N, T, S, P)$ be type-0 Chomsky grammar in Kuroda normal form. Then, the rules of the context-free insertion-deletion system constructed in the proof of Theorem 2.3.1 are of the form $(\lambda,\ a,\ \lambda)$ with $|a| \leq 3$, hence $RE \subseteq INS_3^0DEL_3^0$. □

The total size of the system provided by the proof of Theorem 2.3.1 is 6. We can improve by one this result, by decreasing by one either the length of the inserted strings or the length of the deleted strings. We give below the proof ideas for both cases. These proofs are done by a direct simulation of systems of size $(3, 0, 0; 3, 0, 0)$ using the method presented in 2.2.1. A different approach was used in [51] where a grammar in Kuroda normal form is simulated.

Theorem 2.3.5. $INS_3^{0,0}DEL_2^{0,0} = RE$.

Proof Idea. A deletion rule $i : (\lambda, abc, \lambda)$ can be simulated by an insertion rule $(\lambda, C_iB_iA_i, \lambda)$ and three deletion rules (λ, A_ia, λ), (λ, B_ib, λ) and (λ, C_ic, λ). The simulation works as follows.

$$w_1abcw_2 \Longrightarrow_{ins} w_1\underline{C_iB_iA_i}abcw_2 \Longrightarrow_{del} w_1C_iB_ibcw_2 \Longrightarrow_{del} w_1C_icw_2 \Longrightarrow_{del} w_1w_2.$$

The proof of the validity of this simulation may be obtained in a similar way to Theorem 2.3.1. □

A counterpart of this result is also true: we can trade-off the length of inserted and deleted strings.

Theorem 2.3.6. $INS_2^{0,0}DEL_3^{0,0} = RE$.

Proof Idea. An insertion rule $i : (\lambda, abc, \lambda)$ can be simulated by three insertion rule (λ, aA_i, λ), (λ, bB_i, λ), (λ, cC_i, λ) and a deletion rule $(\lambda, C_iB_iA_i, \lambda)$. The simulation works as follows.

$$w_1w_2 \Longrightarrow_{ins} w_1aA_iw_2 \Longrightarrow_{ins} w_1abB_iA_iw_2 \Longrightarrow_{ins} w_1abcC_iB_iA_iw_2 \Longrightarrow_{del} w_1abcw_2.$$

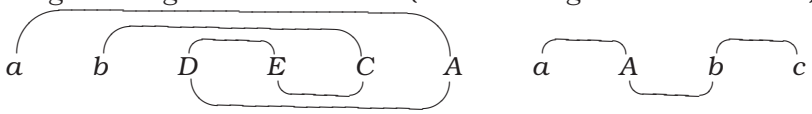
□

2.3.2 Non-completeness results

We show below that the obtained complexity parameters for context-free insertion-deletion systems are optimal. If one of the parameters is further decreased, then the language generated by such systems is included in the family of context-free languages.

The main idea used to obtain this result is that the non-terminal alphabet can be omitted, consequently, the deletion can also be omitted.

This can be argued as follows. Consider a derivation of $w \in T^*$ starting from an empty word. Let us mark the corresponding insertion pairs by an overline and the corresponding deletion pairs by an underline. For example, suppose that we insert aA, after that bC in position 1, DE in position 2, aA in position 6 and bc in position 8. After that suppose that we delete EC, DA and Ab. Then the corresponding marking will be as follows (the resulting word is $w = abac$):



We may interpret symbols as labeled graph nodes and lines as edges. In this case we obtain a graph. It is easy to observe that this graph consists of a set of disjoint linear paths and/or cycles. Indeed, for each node, at most two edges corresponding to an insertion and a deletion may be drawn. Let us also label edges corresponding to insertions by i and edges corresponding to deletions by d. If we take the example above, we obtain:

$$a \overset{i}{\text{—}} A \overset{d}{\text{—}} D \overset{i}{\text{—}} E \overset{d}{\text{—}} C \overset{i}{\text{—}} b$$

$$a \overset{i}{\text{—}} A \overset{d}{\text{—}} b \overset{i}{\text{—}} c$$

We may suppose that the first and the last edge of a path are marked with i. If this is not the case, we add an additional node labeled by λ and we connect this node with the last node by a path labeled by i. In particular, a path containing only one letter a (corresponding to an insertion of a) will be written as $\lambda \overset{i}{\text{—}} a$. Hence, each path consists of sequences of one insertion followed by one deletion.

We observe that for a derivation of a word $w \in T^*$ there can only be paths of the following 4 types:

1. Paths that start with a letter $a \in T$ and that end with a letter $b \in T$.
2. Paths that have at one end a terminal letter a and at the other end λ.
3. Paths that have λ at both ends.
4. Cycles.

We remark that in Case 1 the path leads to the word ab (*i.e.*, contributes to the production of the subword ab of w), in the second case the path produces the letter a and in the last two cases the path generates the empty word.

Without loss of generality, we may suppose that there are no paths of type 3 and 4, because by eliminating the corresponding insertions and deletions we obtain the same word.

Suppose that we have a path marked by over- and underlines as above. We shall understand by an interior of the path the set of all positions that are underlined. In the example above, all positions between D and the first A form the interior of the path. It is clear that no other path (of type 1 and 2) may be situated in the interior of some path, because in this case the corresponding deletion cannot be performed. Consequently, all paths are independent of each other, and we may group rules corresponding to each path and compute paths one after another. Moreover, each path contributes to at most two terminal symbols of the resulting word. Therefore,

the computation consists of insertion of terminal symbols corresponding to paths ends as well as of deletion of terminal symbols.

Moreover, we can show that it is possible to precompute all possible paths. This may be done by using the following observation. We may assume that each path p has the following property: if $A \overset{i}{-} B$ belongs to p, then p does not contain an insertion that has A in the left-hand side ($A \overset{i}{-} X$) or B in the right-hand side ($Y \overset{i}{-} B$). This assertion is obvious, because if p contains such a pair, for example $p = \cdots \overset{d}{-} A \overset{i}{-} X \overset{d}{-} \cdots \overset{d}{-} A \overset{i}{-} B \cdots$, then we may eliminate the subpath between two A's by obtaining an equivalent path (that leads to the same ends) $p' = \cdots \overset{d}{-} A \overset{i}{-} B \cdots$. Hence, the length of each path is bounded by $2 \cdot card(V)$, and we may precompute all possible paths.

In a similar manner it can be proved that the nonterminal alphabet is not relevant even in the general case. See [74] for details.

Lemma 2.3.7. *Let $ID = (V, T, A, I, D)$ be a context-free insertion-deletion system of size $(2, 0, 0; 2, 0, 0)$. Then it is possible to construct a system $ID_2 = (T, T, A_2, I_2, D_2)$ of size $(2, 0, 0; 2, 0, 0)$ such that $L(ID) = L(ID_2)$.*

Moreover, if we consider that the initial system is in the normal form, then there are no deletions of terminal symbols. Hence we obtain that it is sufficient to consider insertion-only systems as $INS_2^{0,0}DEL_2^{0,0} \subseteq INS_2^{0,0}DEL_0^{0,0}$.

We can describe insertion-deletion systems of size $(2, 0, 0; 0, 0, 0)$ by the following context-free grammar, which is a particular case of a more general result for systems of size $(*, 1, 1; 0, 0, 0)$ given in [61].

Let $ID = (T, T, A, I, \emptyset)$ be an insertion-deletion system of size $(2, 0, 0; 0, 0, 0)$. We construct the context-free grammar $G = (\{Z, S\}, T, Z, P)$ as follows. Define $P = P_A \cup P_I \cup \{S \to \lambda\}$, where

$$\begin{aligned} P_A &= \{Z \to Sa_1Sa_2S \ldots Sa_nS \mid a_1a_2 \ldots a_n \in A\}, \\ P_I &= \{S \to SaSbS \mid (\lambda, ab, \lambda) \in I\} \cup \{S \to SaS \mid (\lambda, a, \lambda) \in I\}. \end{aligned}$$

It is clear that $L(G) = L(ID)$. Indeed, symbol S marks all possible insertion positions and permits the simulation of insertion rules as well.

Consequently, we obtain:

Theorem 2.3.8. $INS_2^{0,0}DEL_2^{0,0} = INS_2^{0,0}DEL_0^{0,0} \subset CF$.

Proof. The strictness of the inclusion follows from the fact that insertion-deletion systems of size $(2, 0, 0; 0, 0, 0)$ cannot generate the language $L = \{a^*b^*\}$. Indeed, consider an arbitrary system $ID = (T, T, A, I, \emptyset)$. It is easy to observe that for each word w that belong to $L(ID)$ words $\{x^*wx^* \mid (\lambda, x, \lambda) \in I\}$ belong to $L(ID)$. Therefore, if we suppose that $L(ID)$ is not finite, then $I \neq \emptyset$, and then for any word $w \in L(ID)$, there are words $\{x^*wx^* \mid (\lambda, x, \lambda) \in I\}$ in $L(ID)$. It is easy to see that L does not have such a property. □

Theorem 2.3.9. $INS_2^{0,0}DEL_2^{0,0}$ *is incomparable with REG.*

Proof. From the previous theorem we obtain that $REG \setminus INS_2^{0,0}DEL_2^{0,0} \neq \emptyset$. It is also clear that the Dyck language D_n may be generated by a context-free insertion system having insertion rules $(\lambda, a_i\overline{a}_i, \lambda)$, $1 \leq i \leq n$. Hence, the assertion is proved. □

From the description above it is clear that languages generated by insertion-deletion systems of size $(2,0,0;2,0,0)$ have a particular structure (below, we denote by $\prod$ the concatenation operation).

Theorem 2.3.10. *A language L belongs to $INS_2^{0,0}DEL_2^{0,0}$ if and only if it can be represented in the form*

$$L = h\left(T'^* \sqcup \bigcup_{w=a_1\ldots a_n \in A} \prod_{i=1}^{|w|} Da_iD\right),$$

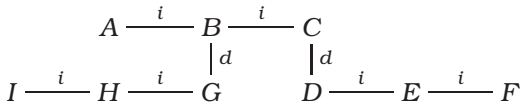
where $A \subseteq T^*$ is a finite set of words, T is an alphabet, D is the Dyck language over an alphabet $T'' \subseteq T$, h is a coding and $T' \subseteq T$.

In a similar way next two results can be obtained. See [74] for more details.

Theorem 2.3.11. *$INS_m^{0,0}DEL_1^{0,0} = INS_m^{0,0}DEL_0^{0,0} \subset CF$ for any $m > 0$.*

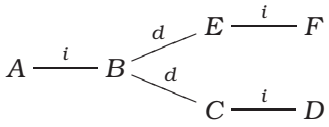
Theorem 2.3.12. *$INS_1^{0,0}DEL_p^{0,0} \subset REG$ for any $p > 0$.*

***Remark* 2.3.13.** Using the graphical representation of insertion and deletion rules it is possible to understand why the computational power increases in the case when the size of inserted or deleted string is increased to 3. In fact, if we construct a similar insertion/deletion graph – by connecting inserted, respectively deleted, symbols pairwise by edges labeled with i, respectively d, and following the ordering of symbols in the string – then this graph may contain non-linear structures. For example, suppose that we insert ABC, after that DEF in position 3, GHI in position 2 and finally delete CD and BG. Then, the corresponding graph will be the following:



The obtained graph no longer has a linear structure, hence it is not possible to affirm anymore that it yields to only two symbols in the final string; moreover we may increase the length of a word.

A similar thing happens when the deletion of strings of length 3 is permitted. In the example below, we inserted AB, CD in position 2, FE in position 1 and deleted EBC.



2.4 One-sided insertion-deletion systems

In this section we present results about insertion-deletion systems with one-sided context, *i.e.*, of size $(n,m,m';p,q,q')$ where either $m + m' > 0$ and $m * m' = 0$, or $q + q' > 0$ and $q * q' = 0$, *i.e.*, one of numbers in some couple is equal to zero.

One-sided insertion-deletion systems present features common to both contextual and context-free insertion-deletion systems. More precisely, an insertion rule having an empty left (or right) context can be applied any number of times like in

the case of context-free rules. However, while a context-free insertion can happen anywhere in the string, in the case of a one-sided insertion the context indicates the place where the insertion can happen. Similar properties are exposed by deletion rules.

Example 2.4.1.

Consider a system $ID = (T, T, \{a\}, I, \emptyset)$, where $T = \{a, b, c, d\}$ and I is defined as follows: $I = \{(a, b, \lambda), (b, c, \lambda), (c, d, \lambda), (d, a, \lambda)\}$.

Let L be the language generated by ID ($L = L(ID)$). It is clear that L can defined by the following formulas:

$$L = L_1 \qquad L_1 = aL_2^* \qquad L_2 = bL_3^* \qquad L_3 = cL_4^* \qquad L_4 = dL_1^*$$

By substituting L_i, for $2 \leq i \leq 4$ into the description of L_{i-1} we obtain:

$$L_1 = a(b(c(dL_1^*)^*)^*)^*$$

Let $R = \{(abcd)^*(dcb)^*\}$. Consider the language $L'' = L \cap R$. Consider the word $w = abcddbc$ from R. This word is generated in L as follows (we underline the inserted symbol):

$$a \Longrightarrow a\underline{b} \Longrightarrow a\underline{b}b \Longrightarrow ab\underline{c}b \Longrightarrow ab\underline{c}cb \Longrightarrow abc\underline{d}cb \Longrightarrow abc\underline{d}dcb$$

We observe that the generation of the second part of w, the subword dcb, is related to the generation of its first part $abcd$, because every letter is inserted two times: first for the second part and after that for the first part. It is also clear that this is the only way to generate the subword dcb. Moreover, it can be easily seen that such a generation leads to a one-to-one correspondence between $abcd$ and dcb. Now, taking w it is possible to insert a after the first letter d and to continue in a similar manner as before and so on, which gives $w_n = (abcd)^n(dcb)^n$, $n \geq 1$. It is also possible to obtain more copies of $abcd$ by performing insertions of four corresponding letters after d, c, b or a in the first part of w_n. Hence, we finally obtain $L'' = \{(abcd)^i(dcb)^j, j \leq i\}$, which is a non-regular context-free language (by the inverse morphism $\{abcd \to x, dcb \to y\}$ it becomes the well known language $\{x^iy^j, 1 \leq j \leq i\}$). Since the intersection of two regular languages would be regular, we obtain that L is a non-regular context-free language.

2.4.1 Computational completeness results

Generally, computational completeness proofs for one-sided insertion-deletion systems take into account the above behavior and ensure that additional symbols that potentially can be inserted more than one time are inserted exactly once. This property is usually satisfied by introducing groups of insertion and deletion rules of a special form that can act only if a specified pattern is present in the string. If the pattern is compromised by inserting or deleting more than one additional symbol, then the whole group of rules will fail and non-terminal symbols will remain in the string; moreover, it can be guaranteed that these symbols cannot be eliminated anymore.

The proofs are based on simulation of insertion-deletion systems from Sections 2.2 and 2.3 which are known to generate all RE languages. The proof technique is very similar to the one from Theorem 2.2.5.

We remark that by symmetry, all results for classes $INS_n^{m,m'}DEL_p^{q,q'}$ are also true for classes $INS_n^{m',m}DEL_p^{q',q}$.

We give the sketch of proof for the following theorem.

Theorem 2.4.2. $INS_1^{1,2}DEL_1^{1,0} = RE$.

Sketch of Proof. The proof is based on the simulation of insertion-deletion systems of size $(1, 1, 1; 1, 1, 1)$ in normal form. Hence, it is sufficient to show how a deletion rule (a, x, b), with $a, b, x \in V$, may be simulated by using rules of the target system, *i.e.*, insertion rules of type $(a', x', b'c')$ and deletion rules of type (a'', y, λ).

Since the system is in normal form, we may assume that $ab \neq \lambda$. Moreover, we may assume that the system has no insertion rules of the form $(a, b, b), a, b \in V$. If this is the case then we replace every such rule by two insertion rules (a, X, b), (a, b, X), and one deletion rule (b, X, b), where X is a new nonterminal.

A deletion rule $i : (a, x, b)$, where i is the label of the rule, is simulated by two insertion rules (x, X_i, b), (a, D_i, xX_i) and three deletion rules (D_i, x, λ), (D_i, X_i, λ), (a, D_i, λ).

Symbols D_i and X_i act like left and right parentheses that surround x before deleting it. The simulation is performed as follows. First, two insertions are performed:

$$w_1axbw_2 \Longrightarrow_{ins} w_1ax\underline{X_i}bw_2 \Longrightarrow_{ins} w_1a\underline{D_i}xX_ibw_2,$$

and then x is deleted:

$$w_1aD_ixX_ibw_2 \Longrightarrow_{del} w_1aD_iX_ibw_2.$$

At this moment symbols X_i and D_i are deleted:

$$w_1aD_iX_ibw_2 \Longrightarrow_{del} w_1aD_iw_2 \Longrightarrow_{del} w_1abw_2.$$

Hence, every derivation in an insertion-deletion system of size $(1, 1, 1; 1, 1, 1)$ can be carried out in a system of size $(1, 1, 2; 1, 1, 0)$. On the other hand, we observe that once being inserted, the nonterminals X_i, D_i can be erased only by the rules shown above. Moreover, if they are not deleted, then no symbol can be inserted at the right of a or at the left of b. The rule (D_i, x, λ) can delete at most one x as the pair D_ix is followed by X_ib and $b \neq x$. Thus, there is a one-to-one correspondence between the original and the new systems, which implies that the theorem statement holds. □

In a similar way the results from Table 2.1 are obtained. We remark that last three results are counterparts of the first three results, where the sizes for insertion and deletion are interchanged. However, in general, systems where insertion parameters are $1, 1, 0$ are simpler than systems having deletion parameters $1, 1, 0$. This is due to the fact that it is easier to control a repeated insertion of symbols by using deletion than a repeated deletion of symbols by using insertion. In the latter case, special "barrier" symbols shall be inserted in order to delimit exactly one symbol to be deleted.

2.4.2 Non-completeness results

In what follows we show that there are classes of one-sided insertion-deletion systems that are not computationally complete.

We start with the following result.

Theorem 2.4.3. $REG \setminus INS_1^{1,0}DEL_1^{1,1} \neq \emptyset$.

Table 2.1: Computationally complete one-sided insertion-deletion systems

Size	Reference
(1,1,2;1,1,0)	[47]
(2,0,2;1,1,0)	[47]
(2,0,1;2,0,0)	[47]
(1,2,0;1,0,2)	[48]
(1,1,0;1,1,2)	[54]
(1,1,0;2,0,2)	[54]
(2,0,0;2,0,1)	[54]

Sketch of Proof. Consider the regular language $L = \{(ba)^+\}$. We claim that there is no insertion-deletion system ID of size (1,1,0;1,1,1) such that $L(ID) = L$. We can suppose that ID is in normal form.

Let $w_f \in (ba)^+$ be a word generated by ID. Now consider an arbitrary ba block of w_f ($w_f = \beta ba\gamma$, $\beta, \gamma \in (ba)^*$) and take its letter a. Since there are no rules deleting terminal symbols in ID this letter is either inserted by an insertion rule or it was a part of an axiom. We may omit the latter case by taking a derivation that produces a string that is long enough. Now suppose that this letter was inserted using a rule $(z, a, \lambda) \in I$, $z \in V$:

$$w \Longrightarrow^* w_1 z w_2 \Longrightarrow w_1 z a w_2 \Longrightarrow^* \beta ba\gamma = w_f. \tag{2.1}$$

This means that:

$$\begin{aligned} w_1 z &\Longrightarrow^* \beta b \\ a w_2 &\Longrightarrow^* a\gamma \end{aligned} \tag{2.2}$$

Now we remark that symbol a might be inserted twice:

$$w \Longrightarrow^* w_1 z w_2 \Longrightarrow w_1 z a w_2 \Longrightarrow w_1 z a a w_2. \tag{2.3}$$

From (2.3) and (2.2) we obtain:

$$w \Longrightarrow^* w_1 z a a w_2 \Longrightarrow^* \beta baa\gamma$$

which is a contradiction. □

In way similar to Theorem 2.4.3 it is possible to show several non-completeness results for one-sided insertion-deletion systems. Table 2.2 summarizes these results. We remark that systems having smaller parameters, like systems of size $(1, 1, 0; 1, 1, 0)$ are also not complete.

Now we shall concentrate on systems of size $(1, 1, 0; 1, 1, 0)$. We show that the language generated by such insertion-deletion systems is a particular subclass of the family of context-free languages.

We start our investigations by systems that do not contain deletion rules. In the book [61] it is already shown that the family $INS_n^{1,1}DEL_0^{0,0}$, $n \geq 1$, is a subset of the family of context-free languages. From the Example 2.4.1 it follows that even a smaller subclass, $INS_1^{1,0}DEL_0^{0,0}$, having one-sided insertion rules contains non-regular context-free languages.

Table 2.2: Computationally non-complete one-sided insertion-deletion systems

Size	Witness language	Reference
(1,1,0;1,1,1)	$(ba)^+$	[47]
(1,1,1;1,1,0)	$a^n b^n$, $n \geq 0$	[54]
(1,1,0;2,0,0)	$(ba)^+$	[46]
(2,0,0;1,1,0)	$(ba)^+$	[46]

Theorem 2.4.4. $INS_1^{1,0}DEL_0^{0,0} \cap (CF \setminus REG) \neq \emptyset$.

The family $INS_1^{1,0}DEL_0^{0,0}$ can be easily described by a context-free grammar. In order to describe the family $INS_1^{1,0}DEL_1^{1,0}$ it is important to show that all possible deletions may be precomputed. We start with the following definitions.

Definition 2.4.5. For a word $w \in L(ID)$ ($u \Longrightarrow^* w$, $u \in A$) we construct the derivation tree of w iteratively as follows:

- Initially the tree has a root labeled by λ with children $a_1, \cdots, a_n$, where $u = a_1 \cdots a_n$. If $n = 1$, we can consider that the tree is rooted by a_1.
- For a transition $w'aw'' \Longrightarrow_{ins} w'abw''$ we consider the node corresponding to the letter a above and add as a left child a node labeled by symbol b.
- For a transition $w'abw'' \Longrightarrow_{del} w'aw''$ we consider the node corresponding to the letter b above and strike it out. In the future, this node is not considered anymore – it is treated like it is replaced it by its children (the corresponding links from the parent of b to all children should be added).

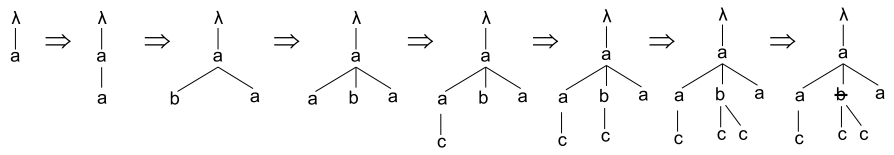
Having a derivation tree T for w, one can read w by concatenating labels of vertices from the preordering of T by a depth-first search. Hence, the root corresponds to the first letter of w and the rightmost label of the tree corresponds to the last letter of w.

Example 2.4.6.

Let $ID=(\{a,b,c\},\{a,b,c\},\{a\},I,D)$ with $I = \{(a,b,\lambda),(a,a,\lambda),(b,c,\lambda),(a,c,\lambda)\}$ and $D = \{(c,b,\lambda)\}$. We can derive $w = aaccca$ as follows:

$$a \Longrightarrow aa \Longrightarrow aba \Longrightarrow aaba \Longrightarrow aacba \Longrightarrow aacbca \Longrightarrow aacbcca \Longrightarrow aaccca$$

This corresponds to the following sequence of trees leading to the derivation tree of w.



It is possible to show that in order to compute the effect of deletion rules for a system of size $(1,1,0;1,1,0)$ it is enough to take only those derivation trees that do not contain repetitions of letters in width and height, *i.e.* for every node there are no child nodes labeled by same letter and any path from the root does not contain

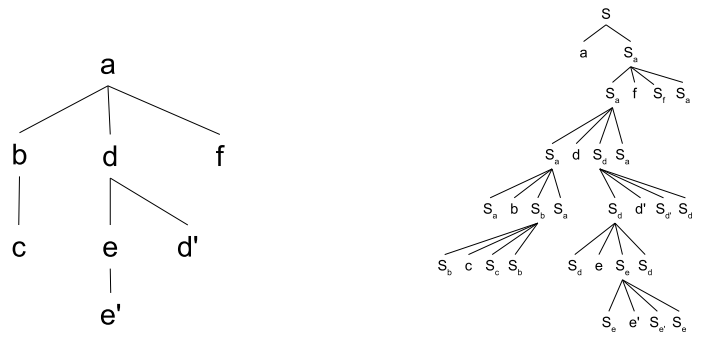
repetitions of letters. Any derivation involving deletions can be decomposed into subderivations where the deletions happen only inside trees of the above form. Since the number of corresponding subtrees is finite, corresponding deletions can be precomputed in advance. This gives the following result:

Theorem 2.4.7. $INS_1^{1,0}DEL_1^{1,0} \subset CF$.

The details of the construction can be consulted in [46].

Example 2.4.8.

Let $ID = (T, T, \{a\}, I, D)$ with $T = \{a, b, c, d, d', e, e', f\}$, $I = \{(a, b, \lambda), (a, d, \lambda), (a, f, \lambda), (b, c, \lambda), (d, e, \lambda), (d, d', \lambda), (e, e', \lambda)\}$ and $D = \{(c, d, \lambda), (c, e, \lambda)\}$. Denote by τ one of trees not having repetitions in width and height and corresponding to the word $abcdee'd'f$ (see Fig. 2.3(a)). Consider also the grammar $G_1 = (N, T, S, P)$, with $N = S \cup \{S_a \mid a \in T\}$, $P = \{S_a \to S_a b S_b S_a \mid (a, b, \lambda) \in I\}$. Consider also the derivation tree τ'' of the grammar G_1 corresponding to τ (see Fig. 2.3(b)).



(a) (b)

Figure 2.3: The trees for the derivation of $abcdee'd'f$ from Example 2.4.8. The derivation in ID (a) and in G_1 (b).

In order to take into account the action of deletion rules from D, following rules shall be added to G_1:

$$S_a \to S_a b S_b c S_d e S_e e' S_{e'} S_e S_d d' S_{d'} S_d S_a f S_f S_a \qquad \text{(the action of } (c, d, \lambda)\text{)}$$
$$S_a \to S_a b S_b c S_e e' S_{e'} S_e S_d d' S_{d'} S_d S_a f S_f S_a \qquad \text{(the action of } (c, e, \lambda)\text{)}$$

In order to finish the elimination of deletion rules, similar rewriting rules shall be added after examination of other derivation trees without repetition of letters.

2.5 Graph-controlled insertion-deletion systems

In previous sections it was shown that there are classes of insertion-deletion systems that cannot generate RE. Making an analogy to context-free grammars, a natural extension of insertion-deletion systems using the graph-controlled or programmed approach can be done. Such model introduces states (or labels of the program) associated to every insertion or deletion rule. The transition is performed by applying corresponding rule and choosing the new state (thus the rule to be applied) among a specific set of rules. Another definition of this model in the style

of [60] or [14] can be done. This definition supposes that there are disjoint groups of insertion and deletion rules (corresponding to *membranes* from [60] or *components* from [14]). The transition is performed by firstly choosing and applying one of applicable rules from the current group and switching to the next group indicated in the rule description.

2.5.1 Formal definition

A *graph-controlled insertion-deletion system* is a construct

$$\Pi = (V, T, A, H, I_0, I_f, R) \text{ where}$$

- V is a finite alphabet,
- $T \subseteq V$ is the *terminal alphabet*,
- $A \subseteq V^*$ is a finite set of *axioms*,
- H is a set of labels associated (in a one-to-one manner) to the rules in R,
- $I_0 \subseteq H$ is the set of *initial labels*,
- $I_f \subseteq H$ is the set of *final labels*, and
- R is a finite set of rules of the form $l : (r, E)$ where r is an insertion or deletion rule over V and $E \subseteq H$.

As it is common for graph controlled systems, a configuration of Π is represented by a pair (w, i), where i is the label of the rule to be applied and w is the current string. A transition $(w, i) \Rrightarrow (w', j)$ is performed if there is a rule $l : ((u, a, v)_t, E)$ in R such that $w \Longrightarrow_t w'$ by the insertion/deletion rule $(u, a, v)_t$, $t \in \{ins, del\}$, and $j \in E$. The result of the computation consists of all terminal strings reaching a final label from an axiom and the initial label, *i.e.*,

$$L(\Pi) = \{w \in T^* \mid (w', i_0) \Rrightarrow^* (w, i_f) \text{ for some } w' \in A,\ i_0 \in I_0,\ i_f \in I_f\}.$$

We will use another rather similar definition for a graph-controlled insertion-deletion system, thereby assigning groups of rules to *components* of the system:

A *graph-controlled insertion-deletion system with k components* is a construct

$$\Pi = (k, V, T, A, H, i_0, i_f, R) \text{ where}$$

- k is the number of components,
- V, T, A, H are defined as for graph-controlled insertion-deletion systems,
- $i_0 \in [1..k]$ is the initial component,
- $i_f \in [1..k]$ is the final component, and
- R is a finite set of rules of the form $l : (i, r, j)$ where r is an insertion or deletion rule over V and $i, j \in [1..k]$.

The set of rules R may be divided into sets R_i assigned to the *components* $i \in [1..k]$, *i.e.*, $R_i = \{l : (r,j) \mid l : (i,r,j) \in R\}$; in a rule $l : (i,r,j)$, the number j specifies the *target component* where the string is sent from component i after the application of the insertion or deletion rule r. A configuration of Π is represented by a pair (w,i), where i is the number of the *current* component (initially i_0) and w is the current string. We also say that w is *situated* in component i. A transition $(w,i) \Rrightarrow (w',j)$ is performed as follows: first, a rule $l : (r,j)$ from component i (from the set R_i) is chosen in a non-deterministic way, the rule r is applied, and the string is moved to component j; hence, the new set from which the next rule to be applied will be chosen is R_j. More formally, $(w,i) \Rrightarrow (w',j)$ if there is $l : ((u,a,v)_t,j) \in R_i$ such that $w \Longrightarrow_t w'$ by the rule $(u,a,v)_t$; we also write $(w,i) \Rrightarrow_l (w',j)$ in this case. The result of the computation consists of all terminal strings situated in component i_f reachable from the axiom and the initial component, *i.e.*,

$$L(\Pi) = \{w \in T^* \mid (w', i_0) \Rrightarrow^* (w, i_f) \text{ for some } w' \in A\}.$$

Is is not difficult to see that graph-controlled insertion-deletion systems with k components are a special case of graph-controlled insertion-deletion systems. Without going into technical details, we just give the main ideas how to obtain a graph-controlled insertion-deletion system from a graph-controlled insertion-deletion system with k components: for every $l : ((u,a,v)_t,j) \in R_i$ we take a rule $l : (i,(u,a,v)_t, Lab(R_j))$ into R where $Lab(R_j)$ denotes the set of labels for the rules in R_j; moreover, we take $I_0 = Lab(R_{i_0})$ and $I_f = Lab(R_{i_f})$. Finally, we remark that the labels in a graph-controlled insertion-deletion system with k components may even be omitted, but they are useful for specific proof constructions. On the other hand, by a standard powerset construction for the labels (as used for the determinization of non-deterministic finite automata) we can easily prove the converse inclusion, *i.e.*, that for any graph-controlled insertion-deletion system we can construct an equivalent graph-controlled insertion-deletion system with k components.

We define the *communication graph* of a graph-controlled insertion-deletion system with k components to be the graph with nodes $1, \ldots, k$ having an edge between node i and j if and only if there exists a rule $l : ((u,a,v)_t,j) \in R_i$. In [60], 5.5, special emphasis is laid on graph-controlled insertion-deletion systems with k components whose communication graph has a tree structure, as we observe that the presentation of graph-controlled insertion-deletion systems with k components given above in the case of a tree structure is rather similar to the definition of insertion-deletion P systems as given in [60]; the main differences are that in P systems the final component i_f contains no rules and corresponds with the root of the communication tree; on the other hand, in graph-controlled insertion-deletion system with k components, each of the axioms can only be situated in the initial component i_0, whereas in P systems we may situate each axiom in various different components.

Throughout the rest of this section we shall only use the notion of graph-controlled insertion-deletion systems with k components, as they are easier to handle and sufficient to establish computational completeness in the proofs of our main results presented in the succeeding section. By $GCID_k(ins_n^{m,m'}, del_p^{q,q'})$ we denote the family of languages $L(\Pi)$ generated by graph-controlled insertion-deletion systems with at most k components and insertion and deletion rules of size at most $(n,m,m';p,q,q')$. We replace k by $*$ if k is not fixed. The letter "G" is replaced by the letter "T" to denote classes whose communication graph has a

tree structure. Some results for the families $TCID_k(ins_n^{m,m'}, del_p^{q,q'})$ can directly be derived from the results presented in [46, 60], for the corresponding families of insertion-deletion P systems $ELSP_k(ins_n^{m,m'}, del_p^{q,q'})$, yet the results we present in the succeeding section either reduce the number of components for systems with an underlying tree structure or else take advantage of the arbitrary structure of the underlying communication graph thus obtaining computational completeness for new restricted variants of insertion and deletion rules.

Example 2.5.1.

Consider the following graph-controlled insertion-deletion system $\Pi = (3, T, T, \lambda, H, 1, 1, R)$, with $T = \{a, b, c\}$, $H = \{1, 2, 3\}$ and $R = R_1 \cup R_2 \cup R_3$, where $R_1 = \{1 : ((\lambda, a, \lambda)_{ins}, 2)\}$, $R_2 = \{2 : ((\lambda, b, \lambda)_{ins}, 3)\}$, $R_3 = \{3 : ((\lambda, c, \lambda)_{ins}, 1)\}$.

The system is inserting consecutively a, b and c. Therefore it is clear that $L(\Pi) = \{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$, which is not a context-free language.

We remark that using two nodes, it is possible to generate the non-regular language $L = \{w \in \{a, b\}^* : |w|_a = |w|_b\}$ in a similar manner. The communication graph has the form of a tree in this case.

2.5.2 Results

We start with the following result from [4].

Theorem 2.5.2. $PsTCID_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsGCID_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT$.

However, in terms of the generated language such systems are not very powerful. Like in the case of context-free insertion-deletion systems there is no control on the position of insertion. Hence, the language $L = \{a^*b^*\}$ cannot be generated, for insertion strings of any size. Hence we obtain:

Theorem 2.5.3. $REG \backslash GCID_*(ins_n^{0,0}, del_1^{0,0}) \neq \emptyset$, *for any* $n > 0$.

However, there are non-context-free languages that can be generated by such systems (even without deletion). From Example 2.5.1 we obtain:

Theorem 2.5.4. $GCID_*(ins_1^{0,0}, del_0^{0,0}) \setminus CF \neq \emptyset$.

We show a more general inclusion:

Theorem 2.5.5. $GCID_*(ins_n^{0,0}, del_1^{0,0}) \subset MAT$, *for any* $n > 0$.

Sketch of Proof. We can suppose that the system is in normal form, hence, in particular, there are no deletions of terminal symbols. Consider a graph-controlled insertion-deletion system $\Pi = (O, T, w, p_0, h, R)$ and $H = Lab(R)$. Such a system can be simulated by the following matrix grammar $G = (O \cup H, T, S, P)$.

For insertion instruction $((\lambda, a_1 \cdots a_n, \lambda)_{ins}, q)$ in component p, let P contain the matrix $\{p \to q, D \to Da_1D \cdots Da_nD\}$. For any deletion instruction $((\lambda, A, \lambda)_{del}, q)$ in component p, let P contain the matrix $\{p \to q, A \to \lambda\}$. We also add to P three additional matrices: $\{h \to \lambda\}$, $\{D \to \lambda\}$ and $\{S \to p_0Da_1D \cdots Da_mD\}$ $(w = a_1 \cdots a_m)$.

The above construction correctly simulates the system Π. Indeed, symbols D represent placeholders for all possible insertions. The first rule in the matrix simulates the navigation between cells. □

Next theorem shows that graph-controlled insertion-deletion systems are strictly more powerful than ordinary insertion-deletion systems of the same size.

Theorem 2.5.6. $TCID_5(ins_1^{1,0}, del_1^{1,0}) = RE$.

The proof is based on the following idea. Any rule $AB \to CD$ of a type-0 grammar in Kuroda normal form can be simulated in 4 stages: (1) erasing A, (2) erasing B, (3) inserting D and (4) inserting C. Every operation can be done by a dedicated component with the help of an additional symbol that marks the position before A and that is used in all operations. A typical computation may look as follows:

$$w_1ABw_2 \Rrightarrow w_1P_iABw_2 \Rrightarrow w_1P_iBw_2 \Rrightarrow w_1P_iw_2 \Rrightarrow$$
$$w_1P_iDw_2 \Rrightarrow w_1P_iCDw_2 \Rrightarrow w_1CDw_2$$

Other rules of the grammar can be simulated in a similar manner. We leave technical details that can be consulted in [48].

In a similar way it is possible to obtain a characterization of RE languages by the family $TCID_5(ins_1^{1,0}, del_1^{0,1})$, *i.e.* with contexts for insertion and deletion on different sides. Taking also into account the symmetrical cases we get:

Corollary 2.5.7. $TCID_5(ins_1^{1,0}, del_1^{0,1}) = TCID_5(ins_1^{0,1}, del_1^{1,0}) = TCID_5(ins_1^{0,1}, del_1^{0,1}) = RE$.

Using a similar technique it is possible to prove following theorems.

Theorem 2.5.8. $TCID_5(ins_1^{1,0}, del_2^{0,0}) = TCID_5(ins_1^{0,1}, del_2^{0,0}) = RE$.

Theorem 2.5.9. *[49]* $TCID_5(ins_2^{0,0}, del_1^{1,0}) = TCID_5(ins_2^{0,0}, del_1^{0,1}) = RE$.

However, in some cases graph-controlled insertion-deletion systems are still not complete.

Theorem 2.5.10. $REG \setminus GCID_*(ins_2^{0,0}, del_2^{0,0}) \neq \emptyset$.

Sketch of Proof. We show that $L_{ab} = \{a^*b\} \notin GCID_k(ins_2^{0,0}, del_2^{0,0})$, for any $k \geq 1$. Assume the converse, and let Π be a graph-controlled insertion-deletion system having context-free rules that may insert or delete at most two symbols and that $L(\Pi) = L_{ab}$. After a reasoning similar to the one from Lemma 2.3.7 we obtain that for every finite derivation in Π one can construct a partition of rules $P_1 \cup \cdots \cup P_r$, $r \geq 1$ such that the overall effect of rules from each P_i, $i = 1, \ldots, r$ is the context-free insertion of at most two terminals. By taking a word of sufficient length it is clear that some applications of rules from P_i which insert a or aa should be performed. Since the insertion is context-free, such an application can happen at the end of the word leading to a word having a preceded by b, which is a contradiction. □

2.5.3 Graph-controlled insertion-deletion systems with priorities

A further control can be added to graph-controlled insertion-deletion systems by introducing a priority of deletion over insertion, *i.e.*, if deletion and insertion rules are applicable, then one of deletion rules will be chosen. This condition can also be viewed as a particular case of the graph-controlled insertion-deletion systems if the latter have rules with appearance checking. We denote by $TCID_k(ins_n^{m,m'} < del_p^{q,q'})$ the families of languages generated by corresponding classes.

Using priorities it is possible to further decrease the length of contexts needed for computational completeness. It is quite astonishing that insertion-deletion systems that insert or delete one symbol in a context-free manner can generate *PsRE*. In case of general communication graph this is particularly easy to see: jumping to an instruction of a register machine corresponds to switching to the associated component, and the entire construction is a composition of graphs shown in Fig. 2.4. The decrement instruction works correctly because of priority of deletion over insertion. A configuration $(p, x_1, \cdots, x_n)$ of a register machine is encoded by strings $Perm(pA_1^{x_1} \cdots A_n^{x_n})$.

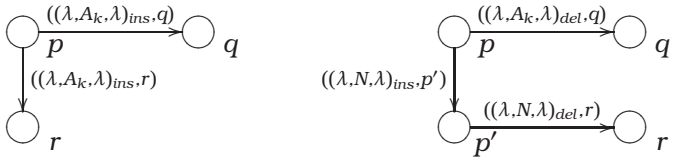


Figure 2.4: Simulating (p, A_k+, q, r)(left) and (p, A_k-, q, r) (right).

For the tree-like communication graph, the proof is more sophisticated and needs a communication graph depicted at Fig. 2.5. The main idea is to use a rule $((\lambda, p, \lambda)_{del}, p_1^+)$ if p is an increment instruction or $((\lambda, p, \lambda)_{del}, p_1^-)$ if p is a decrement instruction and redirect the computation to corresponding components that simulate only one instruction of the register machine. This gives:

Theorem 2.5.11. $PsTCID_*(ins_1^{0,0} < del_1^{0,0}) = PsRE$.

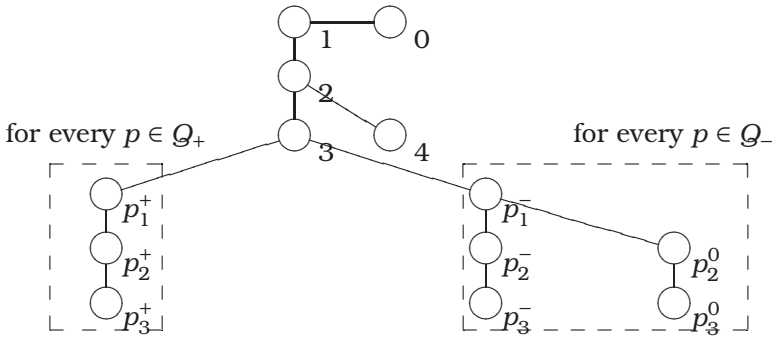


Figure 2.5: Communication graph for Theorem 2.5.11. The structures in the dashed rectangles are repeated for every instruction of the register machine.

Although the above theorem shows that corresponding systems are quite powerful, they cannot generate *RE* without control on the place where a symbol is inserted ($REG\backslash GCID_*(ins_n^{0,0} < del_1^{0,0}) \neq \emptyset$ for any $n > 0$, see Theorem 2.5.3). Once we allow a context in insertion or deletion rules, they can.

Theorem 2.5.12. $TCID_*(ins_1^{0,1} < del_1^{0,0}) = RE$.

For the proof it suffices to simulate register machines with WRITE instructions. It is possible to implement this instruction as an ADD instruction and use a special marker (deleted at the end) to place the "written" symbol at the left of the marker.

In a similar way following result can be obtained.

Theorem 2.5.13. $TCID_*(ins_1^{0,0} < del_1^{1,0}) = RE$.

However in this case the proof is more technical and needs additional components, see [4]. The output symbols are appended at the end of the string and the deletion is used to check for erroneous evolutions the symbol is not added at the end. A similar can be done with a context-free deletion of two symbols.

Theorem 2.5.14. $TCID_*(ins_1^{0,0} < del_2^{0,0}) = RE$.

We mention that the counterpart of Theorem 2.5.14 obtained by interchanging parameters insertion and deletion rules is not true, see Theorem 2.5.3.

2.6 Bibliographical remarks

Insertion systems, without using the deletion operation, were first considered in [30], however the idea of the context adjoining was exploited long time before by [50]. Context-free insertion systems as a generalization of concatenation were first considered in [33, 34]. A formal language study of both context-free insertion and deletion operations was done in [38], however the operations were considered separately. Both operations were first considered together in [40] and related formal language investigations can be found in several places; we mention only [53] and [59]. The biological motivation of insertion-deletion operations leaded to their study in the framework of molecular computing, see, for example, [20], [39], [61], [69]. An interesting study of the deletion operation can be found in [21].

The universality of context-free insertion-deletion systems of size $(2, 0, 0; 3, 0, 0)$ and $(3, 0, 0; 2, 0, 0)$ was shown in [51], while the optimality of this result was shown in [74]. The last article suggested to consider the sizes of each context as a complexity measure and not the maximum as it was done before. One-sided insertion-deletion systems were firstly considered in [54] and the graph-controlled variant in [48]. Graph-controlled insertion-deletion systems with priorities were introduced in [4].

Chapter 3

Study of P systems

The initial intuition for membrane computing, also called P systems, comes from the cell biology, more exactly it was given by the structure of a living cell, whose hierarchical organization is formalized as a set of nested compartments delimited by membranes. Each compartment can contain objects (chemicals) which can participate in evolution rules (reactions) and thus evolve in time. One of the most important features of P systems is that an object can leave the compartment where it was previously situated and move through the system.

The freedom of the choice of types of objects and evolution rules transforms the model of membrane computing into a powerful framework that can be adapted to many case studies. Traditionally, P systems are investigated from the computational completeness and complexity points of view, however there are a lot of successful applications in modeling of biological processes. We refer to the books [62, 12] and to the web page [82] for more details.

We can define, in an informal way, a P system Π as a collection of *cells* (also called *membranes*) $C = \{C_1, \ldots, C_n\}$ each of them containing, in most cases, multisets of objects. Another component of a P system is the *set of rules* $\mathcal{R}$. Every rule $r \in \mathcal{R}$ permits to rewrite a part of the multiset contained in C_i, $1 \leq i \leq n$. Rules of $\mathcal{R}$ induce a relation between cells which can be represented by a hypergraph whose nodes correspond to cells of C and the hyperedges are given by $\mathcal{R}$ (there is a hyperedge between nodes $i_1, \ldots, i_k$ iff there is $r \in \mathcal{R}$ involving at the same time aforementioned cells). We call this hypergraph the *communication (hyper)graph* of Π. In some cases $\mathcal{R}$ can contain rules that change the communication (hyper)graph by adding new nodes and links. This happens, for example, in the case of P systems with active membranes or P systems with membrane division/creation.

Historically, a particular class of P systems is widely investigated; these are the systems whose communication graph has the form of a tree. This corresponds to the initial intuition about P systems which considers them as a formalization of a living cell. In this case, the root of the tree corresponds to the skin membrane and inner nodes correspond to some inner membranes of a cell. The tree structure also permits to define the nesting of membranes in each other (child membranes are nested in the parent membrane). This has a direct correspondence in the biology. Theoretical investigations further extended the structure to a graph (tissue P systems) and hypergraph (network of cells).

The rules in the framework of P systems can be arbitrary, however in most of the cases simple rules are used corresponding to trivial local transformations.

These rules fall into two classes according to the fact if more than one membrane is used to determine the applicability of the rule.

For the applicability of rules of the first class one needs to investigate only the contents of one membrane. Hence, in some sense these rules are attached to the corresponding membrane. A typical example of such rules are multiset rewriting rules $u \to v$, where u is a multiset over V, the alphabet of Π, while v is a multiset over $V \times \{tar_k \mid 1 \leq k \leq n\}$. The multiset u gives the symbols that shall be removed from the membrane, while v gives the new distribution of objects: a symbol (a, tar_k) of v denotes that an object a will be added to the membrane k. The notation tar_k can be replaced by *here* if k is the current membrane, *out* if k is the parent of the current membrane or by *in* if k is one of the children of the current membrane. In the latter case, one of children membranes will be chosen for communication, non-deterministically.

In order to apply rules from the second class, one needs to check the contents of several membranes. In the simplest and most common case, the contents of two membranes is checked. In this case, it is possible to associate the corresponding rule to the edge linking the two membranes. An example of rules of this class are antiport rules, which exchange specific multisets of objects in two membranes. Using more complex rules that check several membranes, it is possible to obtain a direct correspondence with (colored) Petri Nets.

Another interesting class of P systems are communication P systems. In these systems objects cannot be rewritten and they can only be moved from one membrane to another. However, in order to have arbitrary computations an infinite supply of objects of some type shall be present in the environment.

It is worth to note that most of P systems dealing with symbol objects can be considered as ordinary multiset rewriting systems. This becomes obvious if every object is labeled by the number of the membrane in which it is situated. Hence the membrane structure and different types of rules can be viewed as a restriction of form of rules of a multiset rewriting system. However, considering a P system in such a way discards most of important information given by the structuring of the system.

The evolution of P systems is intuitively clear, rules are applied in a non-deterministic maximally parallel manner, which corresponds to a full consumption of ingredients used to perform the operations. However, further investigations of P systems, especially the definition of new transition modes, showed that such informal definitions cannot be applied in all cases and that a formal definition of the computational step should be done. The first attempt to formalize this was done in [26] and Section 3.1 presents this approach in details.

The obtained formal definition permitted to construct new classes of P systems with interesting properties as well as a comparison of existing classes of P systems. An overview of these results is given in Subsection 3.1.2.

3.1 Semantics study

In this section we shall give a precise mathematical definition of P systems. Although Section 3.5 of [60] gives a formal definition of some classes of P systems, there are a lot of classes defined in an informal way. Since the maximal parallelism is a quite simple concept, usually there are no ambiguities concerning the definitions. However, with the introduction of the minimal parallelism [11], the informal definition was not sufficient as several interpretations of this concept could be done. The first attempt to formalize it and to have a uniform definition for many kinds of P systems was done in [26]. In this section we shall follow the line of the aforementioned article.

We shall concentrate our effort on the description of P systems with static structure, *i.e.* where the number of cells/membranes does not evolve in time.

3.1.1 Formal definition of P systems

Since P systems represent computational devices, their formal definition is similar to many such definitions in the area. Hence, in order to define what is a P system we shall answer the following questions:

- What is a configuration of a P system?
- How a transition between two configurations is computed?
- When the computation halts?
- What is the result of the computation?

Below we give an answer to these questions which is valid for most types of P systems (with static structure) known up to now. Such an answer requires an abstraction of different types of P systems, more precisely of their rules and features.

We design a general class of multiset rewriting systems containing, in particular, P systems and tissue P systems. We recall that any P system may be seen at the most abstract level as a multiset rewriting system with only one compartment, encoding the membrane as part of the object representation. However, this approach completely ignores the inner structure of the system because all structural information is hidden (by an encoding) which makes it difficult do deduce any compartment-related information or to model (processes in) biological systems. At a lower level of abstraction, a P system may be seen as a network of cells (compartments) evolving with multi-cell multiset rewriting rules. At the lowest level, the graph/tree structure appears as well as a specialization of rules which are of a very particular form. This last level is usually used in the area of P systems because it permits to easily specify the system and to incorporate different new types of rules.

In order to be quite general, we place our reasoning at the abstract level of *networks of cells*, already considered in a slightly different way in [77] and [26].

Definition 3.1.1. A *network of cells of degree* $n \geq 1$ is a construct

$$\Pi = (n, V, w, Inf, R)$$

where

1. n is the number of cells;
2. V a *finite alphabet*;
3. $w = (w_1, \ldots, w_n)$ where $w_i \in \langle V, \mathbb{N} \rangle$, for all $1 \leq i \leq n$, is the *finite multiset initially associated to cell* i;
4. $Inf = (Inf_1, \ldots, Inf_n)$ where $Inf_i \subseteq V$, for all $1 \leq i \leq n$, is the *set of symbols occurring infinitely often in cell* i (in most of the cases, only one cell, called *the environment*, will contain symbols occurring with infinite multiplicity);
5. R is a finite set of *interaction rules* of the form

$$(X \to Y; P, Q)$$

where $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_n)$, $x_i, y_i \in \langle V, \mathbb{N} \rangle$, $1 \leq i \leq n$, are vectors of multisets over V and $P = (p_1, \ldots, p_n)$, $Q = (q_1, \ldots, q_n)$, p_i, q_i, $1 \leq i \leq n$ are finite sets of multisets over V. We will also use the following notation corresponding to the list representation of sparse vectors:

$$((x_1, 1) \ldots (x_n, n) \to (y_1, 1) \ldots (y_n, n); (p_1, 1) \ldots (p_n, n), (q_1, 1) \ldots (q_n, n))$$

for a rule $(X \to Y; P, Q)$; moreover, if some p_i or q_i is an empty set or some x_i or y_i is equal to the empty multiset, $1 \leq i \leq n$, then we may omit it from the specification of the rule.

A network of cells consists of n cells, numbered from 1 to n, that contain (possibly infinite) multisets of objects over V; initially cell i contains $w_i \cup Inf_i^\infty$. Cells can interact with each other by means of the rules in R. An interaction rule

$$((x_1, 1) \ldots (x_n, n) \to (y_1, 1) \ldots (y_n, n); (p_1, 1) \ldots (p_n, n), (q_1, 1) \ldots (q_n, n))$$

rewrites objects x_i from cells i into objects y_j in cells j, $1 \leq i, j \leq n$, if every cell k, $1 \leq k \leq n$, contains all multisets from p_k and does not contain any multiset from q_k. In other words, the first part of the rule specifies the rewriting of symbols, the second part of the rule specifies permitting conditions and the third part of the rule specifies the forbidding conditions. In the next section we give an even more detailed precise definition for the application of an interaction rule.

It is clear that the set of interaction rules induces a hypergraph structure over the set of cells (vertices). Indeed, for an interaction rule r of the form above, the set

$$\{i \mid x_i \neq \lambda \text{ or } y_i \neq \lambda \text{ or } p_i \neq \emptyset \text{ or } q_i \neq \emptyset\}$$

defines a hyperedge between the interacting cells. In most of the cases, this relation is binary, so corresponding systems are defined on a graph and even tree structure.

We note that most variants of P systems with static structure can be translated in terms of network of cells with specific restrictions of the above rule.

Now we can define the concept of the configuration.

Definition 3.1.2. Consider a network of cells $\Pi = (n, V, w, Inf, R)$. A *configuration* C of Π is an n-tuple of multisets over V $(u'_1, \ldots, u'_n)$ with $u'_i \in \langle V, \mathbb{N}_\infty \rangle$, $1 \leq i \leq n$; in the following, C will also be described by its finite part C^f only, *i.e.*, by $(u_1, \ldots, u_n)$ satisfying $u'_i = u_i \cup Inf_i^\infty$ and $u_i \cap Inf_i = \emptyset$, $1 \leq i \leq n$.

In the sense of the preceding definition, the *initial configuration* of Π, C_0, is described by w, *i.e.*, $C_0^f = w = (w_1, \ldots, w_n)$, whereas $w_i' = w_i \cup Inf_i^\infty$, $1 \leq i \leq n$, is the initial contents of cell i, *i.e.*, $C_0 = w \cup Inf^\infty$.

Definition 3.1.3. We say that an interaction rule $r = (X \to Y; P, Q)$ is *eligible* for the configuration C with $C = (u_1, \ldots, u_n)$ if and only if for all i, $1 \leq i \leq n$, we have

- for all $p \in p_i$, $p \subseteq u_i$ (every $p \in p_i$ is a submultiset of u_i),
- for all $q \in q_i$, $q \not\subseteq u_i$ (no $q \in q_i$ is a submultiset of u_i), and
- $x_i \subseteq u_i$ (x_i is a submultiset of u_i).

Moreover, we require that $x_j \cap (V - inf_j) \neq \emptyset$ for at least one j, $1 \leq j \leq n$. This last condition ensures that at least one symbol appearing only in a finite number of copies is involved in the rule. The set of all rules eligible for C is denoted by $Eligible(\Pi, C)$.

The marking algorithm. Let $C = (v_1, \ldots, v_n)$ be a configuration of a network of cells Π and C^f its finite description; moreover, let R' be a finite multiset of rules from R consisting of the (copies of) rules $r_1, \ldots, r_k$, where for each i, $1 \leq i \leq k$, we have $r_i = (X_i \to Y_i; P_i, Q_i) \in Eligible(\Pi, C)$, $X_i = (x_{i,1}, \ldots, x_{i,n})$, $Y_i = (y_{i,1}, \ldots, y_{i,n})$. Moreover, let X_i' and Y_i', $1 \leq i \leq k$, be the vectors of finite multisets from $\langle V, \mathbb{N} \rangle$ with $X_{i,j} = X_{i,j}' \cup Inf_j^\infty$ and $X_{i,j}' \cap Inf_j = \emptyset$, $1 \leq j \leq n$. Then:

1. consider a vector of multisets $Mark_0(\Pi, C, R') = (\lambda, \ldots, \lambda)$ of size n and let $i = 1$;
2. if $X_i' \leq C^f - Mark_{i-1}(\Pi, C, R')$, then set
$$Mark_i(\Pi, C, R') = C^f - Mark_{i-1}(\Pi, C, R') - X_i',$$
otherwise, end the algorithm and return **false**;
3. if $i = k$ then end the algorithm and return **true**, otherwise set i to $i + 1$ and return to step 2.

If the marking algorithm returns **true** for the pair (C, R') then we say that the configuration C may be *marked* by R', and define $Mark(\Pi, C, R') = Mark_k(\Pi, C, R')$.

Definition 3.1.4. Consider a configuration C and R' be a multiset of rules from $Eligible(\Pi, C)$ (*i.e.*, a multiset of eligible rules). We say that the multiset of rules R' is *applicable* to C if the marking algorithm as described above returns **true** and $Mark(\Pi, C, R')$. The set of all multisets of rules *applicable* to C is denoted by $Appl(\Pi, C)$.

We remark that the marking algorithm is not the only way to define the set $Appl(\Pi, C)$. There are other possibilities of its definition, here we just wanted to give at least one precise algorithm of its computation.

Definition 3.1.5. Consider a configuration C and a multiset of rules $R' \in Appl(\Pi, C)$. According to the marking algorithm described above, we define the configuration being the result of *applying* of R' to C as

$$Apply(\Pi, C, R') = (C - Mark(\Pi, C, R')) + \Sigma_{1 \leq i \leq k} Y_i'.$$

We remark that $Apply(\Pi, R', C)$ is again a configuration.

For the specific *derivation modes* to be defined in the following, the selection of multisets of rules applicable to a configuration C may only be a specific subset of $Appl(\Pi, C)$.

Definition 3.1.6. For the derivation mode δ, the selection of multisets of rules applicable to a configuration C is denoted by $Appl(\Pi, C, \delta)$.

It should be made clear that $Appl(\Pi, C, \delta) \subseteq Appl(\Pi, C)$.

For a derivation mode we now can define how to obtain a next configuration from a given one by applying an applicable multiset of rules according to the constraints of the underlying derivation mode:

Definition 3.1.7. Given a configuration C of Π and a derivation mode δ, we may choose a multiset of rules $R' \in Appl(\Pi, C, \delta)$ in a non-deterministic way and apply it to C. The result of this *transition step* from the configuration C with applying R' is the configuration $Apply(\Pi, C, R')$, and we also write $C \Longrightarrow_{(\Pi,\delta)} C'$. The reflexive and transitive closure of the transition relation $\Longrightarrow_{(\Pi,\delta)}$ is denoted by $\Longrightarrow^*_{(\Pi,\delta)}$.

Especially for computing and accepting devices, the notion of determinism is of major importance. For networks of cells, determinism can be defined as follows:

Definition 3.1.8. A configuration C is said to be *accessible* in Π with respect to the derivation mode δ if and only if $C_0 \Longrightarrow^*_{(\Pi,\delta)} C$ (C_0 is the initial configuration of Π). The set of all accessible configurations in Π is denoted by $Acc(\Pi)$.

Definition 3.1.9. A network of cells Π is said to be *deterministic* with respect to the derivation mode δ if and only if $|Appl(\Pi, C, \delta)| = 1$ for any accessible configuration C.

Halting conditions

A halting condition is a predicate applied to an accessible configuration. The system halts according to the halting condition if this predicate is true for the current configuration. In such a general way, the notion of halting with final state or signal halting can be defined as follows:

Definition 3.1.10. An accessible configuration C is said to fulfill the *signal halting* condition or *final state halting* condition (S) if and only if $C \in S(\Pi, \delta)$ where

$$S(\Pi, \delta) \quad = \quad \{C' \mid C' \in Acc(\Pi) \text{ and } State(\Pi, C', \delta) = true\}.$$

Here $State(\Pi, C', \delta)$ means a decidable feature of the underlying configuration C', e.g., the occurrence of a specific symbol (signal) in a specific cell.

The most important halting condition used from the beginning in the P systems area is the *total halting*, usually simply considered as *halting*:

Definition 3.1.11. An accessible configuration C is said to fulfill the *total halting* condition (H) if and only if no multiset of rules can be applied to C with respect to the derivation mode anymore, *i.e.*, if and only if $C \in H(\Pi, \delta)$ where

$$H(\Pi, \delta) \quad = \quad \{C' \mid C' \in Acc(\Pi) \text{ and } Appl(\Pi, C', \delta) = \emptyset\}.$$

The adult halting condition guarantees that we still can apply a multiset of rules to the underlying configuration, yet without changing it anymore:

Definition 3.1.12. An accessible configuration C is said to fulfill the *adult halting* condition (A) if and only if $C \in A(\Pi, \delta)$ where

$$\begin{aligned} A(\Pi, \delta) \quad = \quad & \{C' \mid C' \in Acc(\Pi),\ Appl(\Pi, C', \delta) \neq \emptyset \text{ and} \\ & Apply(\Pi, C', R') = C' \text{ for every } R' \in Appl(\Pi, C', \delta)\}. \end{aligned}$$

We should like to mention that we could also consider $A(\Pi, \delta) \cup H(\Pi, \delta)$ instead of $A(\Pi, \delta)$.

Computation, goal and result of a computation

A computation in a network of cells Π, $\Pi = (n, V, w, Inf, R)$, starts with the initial configuration C_0, $C_0 = w \cup Inf^{\infty}$, and continues with transition steps according to the chosen derivation mode until the halting condition is met.

The computations with a network of cells may have different goals, e.g., to generate (*gen*) a (vector of) non-negative integers in a specific output cell (membrane) or to accept (*acc*) a (vector of) non-negative integers placed in a specific input cell at the beginning of a computation. Moreover, the goal can also be to compute (*com*) an output from a given input or to output yes or no to decide (*dec*) a specific property of a given input.

The results not only can be taken as the number (N) of objects in a specified output cell (we also write N_i if corresponding cell has the label i), but, for example, also be taken modulo a terminal alphabet (T) or by subtracting a constant from the result ($-k$).

Such different tasks of a network of cells may require additional parameters when specifying its functioning, e.g., we may have to specify the output/input cell(s) and/or the terminal alphabet.

We shall not go into the details of such definitions here, we just mention that the goal of the computation $\gamma \in \{gen, acc, com, dec\}$ and the way to extract the results ρ (usually taken from halting computations) are two other parameters to be specified and clearly defined when defining the functioning of a network of cells or of a membrane system.

Taxonomy of networks of cells and (tissue) P systems

For a particular variant of networks of cells or especially P systems/tissue P systems we have to specify the derivation mode, the halting condition as well as the procedure how to get the result of a computation, but also the specific kind of rules that are used, especially some complexity parameters.

For networks of cells, we shall use the notation

$$O_m C_n(\delta, \phi, \gamma, \rho)[\text{parameters for rules}]$$

to denote the family of sets of vectors obtained by networks of cells defined by $\Pi = (n, V, w, Inf, R)$ with $m = |V|$, as well as δ, ϕ, ρ indicating the derivation mode, the halting condition, and the way how to get results, respectively; the *parameters for rules* describe the specific features of the rules in R. If any of the parameters m and n is unbounded, we replace it by $*$.

For P systems, with the interaction between the cells in the rules of the corresponding network of cells allowing for a tree structure as underlying interaction graph, we shall use the notation

$$O_m P_n(\delta, \phi, \gamma, \rho)[\text{parameters for rules}].$$

Observe that usually the environment is not counted when specifying the number of membranes in P systems, but this usually hides that in many cases the environment takes an important role in the functioning of the system.

For tissue P systems, with the interaction between the cells in the rules of the corresponding network of cells allowing for a graph structure as underlying interaction graph, we shall use the notation

$$O_m tP_n(\delta, \phi, \gamma, \rho)[\text{parameters for rules}].$$

In most of the cases the results obtained in the area of P systems are formulated for sets of numbers instead of sets of vectors. Although in most of the cases the corresponding formulations are valid for the vector case, we shall use the traditional notations and write symbol N in front, e.g., $NO_m tP_n(\delta, \phi, \gamma, \rho)$[parameters for rules].

3.1.2 Study of different derivation modes

In this section we give examples of different derivation modes that can be defined for network of cells and P systems. We start with the simplest possible derivation mode.

Definition 3.1.13. For the *asynchronous* derivation mode (*asyn*),

$$Appl(\Pi, C, asyn) = Appl(\Pi, C),$$

i.e., there are no particular restrictions on the multisets of rules applicable to C.

Definition 3.1.14. For the *sequential* derivation mode (*sequ*),

$$Appl(\Pi, C, sequ) = \{R' \mid R' \in Appl(\Pi, C) \text{ and } |R'| = 1\},$$

i.e., any multiset of rules $R' \in Appl(\Pi, C, sequ)$ has size 1.

The most important derivation mode considered in the area of P systems from the beginning is the *maximally parallel* derivation mode where we only select multisets of rules R' that are not extensible, *i.e.*, there is no other multiset of rules $R'' \supsetneq R'$ applicable to C.

Definition 3.1.15. For the *maximally parallel* derivation mode (*max*),

$$\begin{aligned} Appl(\Pi, C, max) \quad = \quad & \{R' \mid R' \in Appl(\Pi, C) \text{ and there is} \\ & \text{no } R'' \in Appl(\Pi, C) \text{ with } R'' \supsetneq R'\}. \end{aligned}$$

For the *minimally parallel* derivation mode, we need an additional feature for the set of rules R, *i.e.*, we consider a partitioning Θ of R into disjoint subsets R_1 to R_h. Usually, this partition of R' may coincide with the assignment of rules to the cells. For any set of rules $R' \subseteq R$, let $\|R'\|$ denote the number of sets of rules R_j, $1 \leq j \leq h$, with $R_j \cap R' \neq \emptyset$.

There are several possible interpretations of this minimally parallel derivation mode which in an informal way can be described as applying multisets such that from every set R_j, $1 \leq j \leq h$, at least one rule – if possible – has to be used (e.g., see [11]). We start with the basic variant where in each derivation step we only choose a multiset of rules R' from $Appl(\Pi, C, asyn)$ that cannot be extended to $R'' \in Appl(\Pi, C, asyn)$ with $R'' \supsetneqq R'$ as well as $(R'' - R') \cap R_j \neq \emptyset$ and $R' \cap R_j = \emptyset$ for some j, $1 \leq j \leq h$, *i.e.*, extended by a rule from a set of rules R_j from which no rule has been taken into R'.

Definition 3.1.16. For the *minimally parallel* derivation mode with partitioning Θ ($min(\Theta)$),

$$\begin{aligned} Appl(\Pi, C, min(\Theta)) = \{R' \mid\ & R' \in Appl(\Pi, C, asyn) \text{ and} \\ & \text{there is no } R'' \in Appl(\Pi, C, asyn) \\ & \text{with } R'' \supsetneqq R',\ (R'' - R') \cap R_j \neq \emptyset \\ & \text{and } R' \cap R_j = \emptyset \text{ for some } j,\ 1 \leq j \leq h\}. \end{aligned}$$

We can omit Θ if it can be deduced from the context.

In the following we also consider further restricting conditions on the four basic modes defined above, especially interesting for the minimally parallel derivation mode, thus obtaining some new combined derivation modes.

A derivation mode closely related to the maximally parallel one can be obtained if we not only demand that the chosen multiset R' is not extensible, but also contains the maximal number of rules among all applicable multisets:

Definition 3.1.17. For any derivation mode δ, we define the *maximal in rules* δ derivation mode ($max_{rule}\delta$) by setting

$$\begin{aligned} Appl(\Pi, C, max_{rule}\delta) = \{R' \mid\ & R' \in Appl(\Pi, C, \delta) \text{ and} \\ & \text{there is no } R'' \in Appl(\Pi, C, \delta) \\ & \text{with } |R''| > |R'|\}. \end{aligned}$$

In the case of the minimally parallel derivation mode, we can also maximize the sets of rules involved in a multiset to be applied ($max_{set}min$):

Definition 3.1.18. For any derivation mode δ and a partition of rules Θ we define the *maximal in sets of* Θ δ derivation mode ($max_{set}(\Theta)\delta$) by setting

$$\begin{aligned} Appl(\Pi, C, max_{set}(\Theta)\delta) = \{R' \mid\ & R' \in Appl(\Pi, C, \delta) \text{ and} \\ & \text{there is no } R'' \in Appl(\Pi, C, \delta) \\ & \text{with } \|R''\| > \|R'\|\}. \end{aligned}$$

We can restrict further the minimally parallel transition mode by allowing at most k rules to be taken from each partition R_j, $1 \leq j \leq h$.

Definition 3.1.19. The *k-restricted minimally parallel* transition mode with partitioning Θ ($min_k(\Theta)$) is defined as

$$\begin{aligned} Appl(\Pi, C, min_k(\Theta)) = \{R' \mid\ & R' \in Appl(\Pi, C, min(\Theta)) \text{ and} \\ & |R' \cap R_j| \leq k \text{ for all } j,\ 1 \leq j \leq h\}. \end{aligned}$$

The min_1 derivation mode is of special interest as it provides the simplest way of maximally parallel evolution: at the level of partition the execution is sequential, while partitions evolve in a maximally parallel manner. Such behavior is for example observed in catalytic or spiking P systems. Moreover, each multiset of rules from min_1 can be seen as a kind of basic maximally parallel vector and it can be used for the definition of the minimal parallelism as it is done by Gh. Păun in [62]. We give below the formalization of that definition:

Definition 3.1.20. For the *base vector minimally parallel* derivation mode with partitioning Θ ($min_G(\Theta)$),

$$\begin{aligned} Appl(\Pi, C, min_G(\Theta)) \quad = \quad & \{R' \mid R' \in Appl(\Pi, C, min(\Theta) \text{ and } R' \supseteq R'' \\ & \text{for some } R'' \in Appl(\Pi, C, min_1(\Theta))\}. \end{aligned}$$

Looking carefully into the definitions for all the derivation modes defined above, we immediately infer the following equalities which do not depend on the kind of rules at all (observe that the restricting conditions for the combined modes using the condition max_{rule} or max_{set} are defined with respect to the underlying basic mode, which, for example, immediately implies the equalities for the sequential mode):

Lemma 3.1.21. *The following equalities for derivation modes hold true in general for all kinds of networks of cells:*

$max_{rule}sequ = max_{set}sequ = sequ$,
$max_{rule}asyn = max_{rule}min = max_{rule}max$,
$max_{set}min = max_{set}asyn$,

It is also clear that different derivation modes may yield different application results. Moreover, the following simple example shows most of the incomparabilities between some derivation modes.

Example 3.1.22.

Consider the network of cells

$$\Pi = (4, \{a, b\}, (b^3, a^3, b, b), (\emptyset, \emptyset, \emptyset, \emptyset), R)$$

with the following rules in R:

1. $(b, 1)(a, 2) \to (a, 1)(b, 2)$
2. $(a, 2)(b, 3) \to (b, 2)(a, 3)$
3. $(aa, 2)(b, 4) \to (b, 2)(aa, 4)$

In fact, Π can be interpreted as a P system with antiport rules, 3 membranes with membrane i represented by cell $i + 1$, as well as cell 1 representing the environment, see Fig. 3.1. Due to the availability of objects in the four cells, only the following multisets of rules (represented as strings) are applicable to the initial configuration C_0, $C_0 = (b^3, a^3, b, b)$:

$$Appl(\Pi, C_0, asyn) = Appl(\Pi, C_0) = \{1, 1^2, 1^3, 1^2 2, 12, 13, 2, 23, 3\}$$

Assuming the partition for the minimally parallel derivation mode to be the partition into the three single rules (which corresponds to assigning the

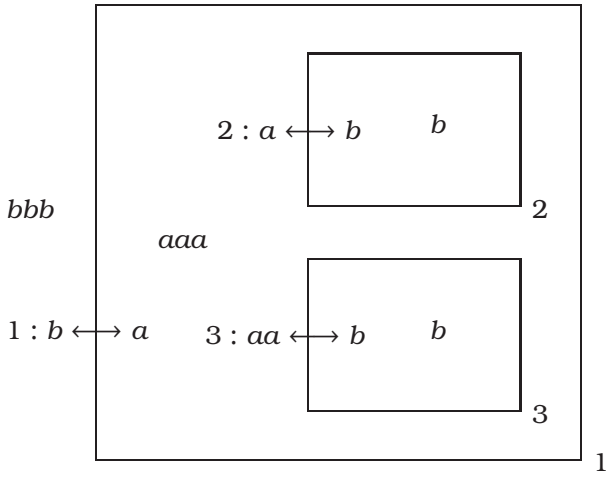


Figure 3.1: Network of cells depicted as P system with antiport rules.

rule i to cell $i+1$ – corresponding to membrane i in a membrane system – and no rule to cell 1 which represents the environment), we obtain the following sets of multisets of rules applicable to C_0 according to the different derivation modes:

$Appl(\Pi, C_0, \delta_1) = \{1, 2, 3\}$ for $\delta_1 \in \{sequ, max_{rule}\, sequ, max_{set}\, sequ\}$,

$Appl(\Pi, C_0, min) = \{1^3, 1^2 2, 12, 13, 23\}$,

$Appl(\Pi, C_0, max) = \{1^3, 1^2 2, 13, 23\}$,

$Appl(\Pi, C_0, max_{rule}\, \delta_2) = \{1^3, 1^2 2\}$, $\delta_2 \in \{asyn, min, max\}$,

$Appl(\Pi, C_0, max_{set}\, \delta_3) = \{1^2 2, 12, 13, 23\}$, $\delta_3 \in \{asyn, min\}$,

$Appl(\Pi, C_0, max_{set}\, max) = \{1^2 2, 13, 23\}$,

$Appl(\Pi, C_0, min_1) = \{12, 13, 23\}$,

$Appl(\Pi, C_0, min_G) = \{1^2 2, 12, 13, 23\}$.

All these sets of multisets of rules listed above are different which shows the incomparability of the corresponding derivation modes (observe that the inherent equalities for the modes δ_1, δ_2, and δ_3 follow from Lemma 3.1.21).

Example 3.1.23.

In this example we consider catalytic P systems. Such systems have a special set of symbols C, each of them called catalyst, and a set of rewriting rules R of type $ca \to cu$, where $c \in C$ and symbols of u can be distributed to other membranes. Such rules formalize the notion of a catalytic reaction from chemistry. Although the system evolves in a maximally parallel manner, it is clear that at most $|C|$ rules can be executed in parallel. This permits to consider a partition of rules Θ with respect to the used catalyst: $\Theta = \Theta_1, \ldots, \Theta_n$, $n = |C|$ and $\Theta_i = \{a \to u \mid c_i a \to c_i u \in R\}$. It is easy to see that $Appl(\Pi, C, max) = Appl(\Pi, C, min_1(\Theta))$. Hence, any catalytic P system evolving in maximally parallel derivation mode is a context-free rewriting P system working in $min_1(\Theta)$ mode, Θ being the partition of rules associated to each catalyst.

Other derivation modes were also considered, see for example [75, 27, 24].

3.2 Communication models

As it was mentioned before, the communication models present the particularity that objects of the system are not changed during the evolution, but only moved. This directly corresponds to the biological phenomena of membrane transport and the first communication models where the models using symport and antiport operations.

We shall give in more details the biological insight behind these operations. There are different membrane transport mechanisms, depending on the size and type of molecules. Fig. 3.2 gives an overview of the most important trans-membrane transport mechanisms.

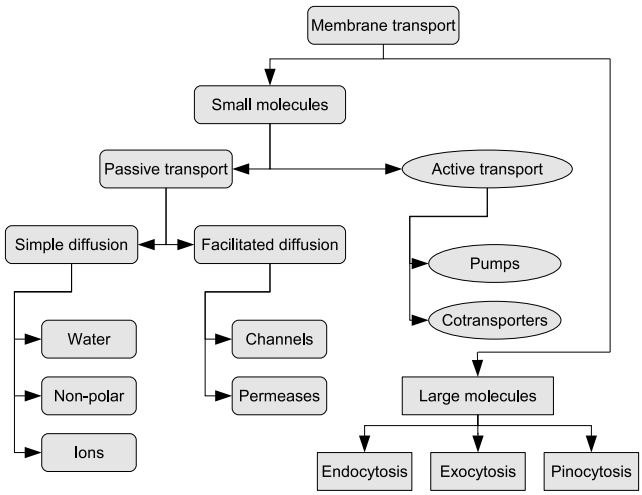


Figure 3.2: Membrane transport mechanisms

Large molecules need a deformation of the membrane they cross with a creation of a new membrane surrounding the molecule. Smaller molecules can directly penetrate inside a membrane or leave it. In the case of a passive transport, the molecules are very small (water, small ions) and can move through membrane in the direction of the chemical gradient. Such mechanism does not need any additional energy to be consumed.

In the case of an active transport, molecules are moved against the chemical gradient, so the consumption of energy is necessary. Symport and antiport (see Fig. 3.3) are particular example of co-transporters. In the symport case two molecules will travel together inside or outside the membrane, while in the antiport case two molecules will be exchanged.

This process can be formalized as follows. The biological membrane corresponds to a membrane in the P system, molecules to objects and the antiport process to the antiport operation. An antiport rule is denoted as $(u, in; v, out)$ and permits to exchange two multisets y and x, which are situated in region i and the outer region of i respectively. A symport rule (x, in) permits to move x into a region from the immediately outer region. In a similar way, a symport rule (x, out) permits to move the multiset x from a region to the outer region. In multiset rewriting terms, an antiport rule $(u, in; v, out)$ corresponds to the multiset rewriting rule $u_j v_i \rightarrow u_i v_j$, where i is the label of the inner region of the involved membrane and

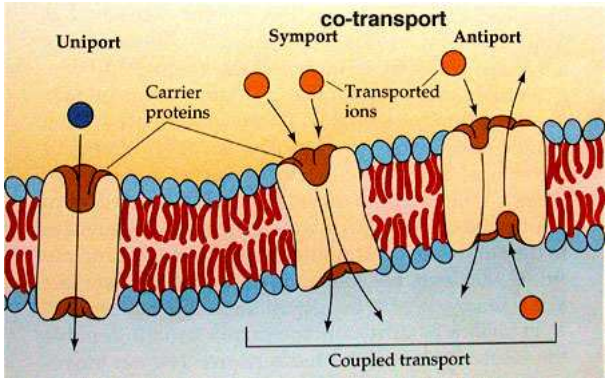


Figure 3.3: Uniport, symport and antiport

j is the label of the outer region of the membrane. A symport rule can be defined analogously.

We remark that during the evolution we consider only the action of symport or antiport rules. Hence, the objects (molecules) are not modified, but only moved between different cells or compartments of a cell.[1] From the formal language theory point of view, the evolution of such a system can lead only to a finite number of configurations, because in the initial configuration a finite number of objects is present and no new objects are created/removed. A natural addition to such a model would be the presence of a region where some objects are present in "infinite" (or more precisely in a large unbounded) number of copies. Taking the biological analogy, this addition corresponds to the environment of a cell or group of cells, which contain a huge number of substances interacting with the cell. After the addition of this "infinity" the obtained formal systems become more complicated and in a lot of cases their behavior is unpredictable.

3.2.1 Formal definition

We shall give the traditional definition of symport/antiport P systems and translate it afterwards using terms defined in Section 3.1.

Definition 3.2.1. A P system with symport/antiport of degree n is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_n, E, R_1, \ldots, R_n, i_0),$$

where:

1. O is a finite alphabet of symbols called objects,
2. μ is a membrane structure consisting of n membranes that are labeled in a one-to-one manner by $1, 2, \ldots, n$. Usually, μ is given as a set of nested parentheses.
3. $w_i \in O^*$, for each $1 \leq i \leq n$ is a finite multiset of objects associated with the region i (delimited by membrane i),

[1] Even if the objects are not modified, the couple object/position changes. These changes are best exposed by the multiset rewriting representation of the system

4. $E \subseteq O$ is the set of objects that appear in the environment in infinite numbers of copies,

5. R_i, for each $1 \leq i \leq n$, is a finite set of symport/antiport rules associated with the region i and which have the following form (x, in), (y, out), $(y, out; x, in)$, where $x, y \in O^*$,

6. i_0 is the label of a membrane of μ that identifies the corresponding output region.

This definition can be translated to the network of cells as follows. We consider $\Pi = (n + 1, O, w, Inf, R)$, where $Inf = (E, \emptyset, \ldots, \emptyset)$ (we note that the environment has the label 0), $w = (\lambda, w_1, \ldots, w_n)$ and the set of rules R is defined as follows:

- for any rule $(x, in; y, out) \in R_i$ we add to R the rule $(x,j)(y, i) \rightarrow (y,j)(x, i)$, where j is the parent membrane of i according to μ,
- for any rule $(x, in) \in R_i$ we add to R rules $(x, i) \rightarrow (x,j)$, for all children membranes j of i, according to μ,
- for any rule $(y, out) \in R_i$ we add to R the rule $(y, i) \rightarrow (y,j)$, where j is the parent membrane of i according to μ.

We remark that the root of μ has a parent node – the environment labeled by 0.

The obtained class of systems is denoted as $NO_mP_n(\delta, \phi, \gamma, \rho)[sym_r, anti_t]$, where r and t is the maximal size of symport and antiport rules respectively. The size of a symport rule (x, in) or (x, out) is given by $|x|$, while the size of an antiport rule $(y, out; x, in)$ is given by $|x| + |y|$.

In most of the cases the considered classes have $\delta = max$, $\phi = H$, $\gamma = gen$ and $\rho = N_{i_0}$. So we shall use a special notation for this case:
$NO_mP_n(max, H, gen, N_{i_0})[sym_r, anti_t] = NOP_n(sym_r, anti_t)$.

The result of a successful computation is the natural number that is obtained by counting the objects that are presented in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$.

In the accepting variants of symport/antiport P systems the system accepts the number corresponding to the quantity of objects given at the beginning of the computation in the predefined cell i_0.

For the tissue case the traditional definition is as follows.

Definition 3.2.2. A *tissue P system with symport/antiport* of degree $n \geq 1$ is a construct

$$\Pi = (O, G, w_1, \ldots, w_n, E, R, i_0),$$

where O is the alphabet of objects and G is the underlying directed labeled graph of the system. The graph G has $n + 1$ nodes and the nodes are numbered from 0 to n. We shall also call nodes from 1 to n cells and node 0 the environment. There is an edge between each cell i, $1 \leq i \leq n$, and the environment. Each cell contains a multiset of objects, initially cell i, $1 \leq i \leq n$, contains multiset w_i. The environment is a special node which contains symbols from E in infinite multiplicity as well as a finite multiset over $O \setminus E$, but initially this multiset is empty. The symbol $i_0 \in (1 \ldots n)$ indicates the output cell, and R is a finite set of rules (associated to edges) of the following forms:

1. (i, x, j), $0 \leq i \leq n, 0 \leq j \leq n, i \neq j$, $x \in O^+$ and not $i = 0$ & $x \in E^+$ (symport rules for the communication).

2. $(i, x/y, j)$, $0 \leq i, j \leq n, i \neq j$, $x, y \in O^+$(antiport rules for the communication).

Such definition translates easily to the network of cells as follows. We consider $\Pi = (n + 1, O, w, Inf, R')$, where as before $Inf = (E, \emptyset, \ldots, \emptyset)$ (the environment has the label 0), $w = (\lambda, w_1, \ldots, w_n)$ and R' is defined as follows:

- for any rule $(i, x/y, j) \in R$ we add to R' the rule $(x, i)(y, j) \to (y, i)(x, j)$,
- for any rule $(i, x, j) \in R$ we add to R' the rule $(x, i) \to (x, j)$.

As above, we denote by $NOtP_n(sym_p, anti_q)$ the family of all sets of numbers generated by tissue P systems with symport/antiport of degree at most n and which have symport rules of size at most p and antiport rules of size at most q, working in the derivation mode *max* and using halting condition H.

Example 3.2.3.

Consider the following symport/antiport P system:
$\Pi = (\{A, B\}, \{B\}, [_1[_2]_2]_1, \{B\}, \{A^n\}, R_1, R_2, 1)$, with $n > 0$ and
$R_1 = \{1 : (A, out; BB, in)\}$ and $R_2 = \{2 : (B, in; A, out), 3 : (B, out)\}$.

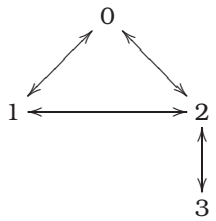
Suppose that the number of B's in membrane 1 is equal to q (initially 1) and that the number of A's in membrane 2 is equal to p (initially $p = n$). Suppose also that there are no symbols A in the first membrane and no symbols B in the second (the initial configuration satisfies this condition). Hence, the configuration of Π at this moment is (A^m, B^q, A^p) $(p + m = n)$. Suppose that $q < p$. Then at the next step only rule 2 is applicable leading to the configuration $(A^m, A^q, A^{p-q}B^q)$. On the next step rules 1 and 3 are applicable in parallel leading to the configuration $(A^{m+q}, B^{3q}, A^{p-q})$. If $q \geq p$, then after two steps the configuration $(A^n, B^{q+2p}, \emptyset)$ is obtained. By induction we deduce that Π computes the function $2n + 1$ in $2([\log_3 2n] + 1)$ steps.

Example 3.2.4.

Consider the following tissue P system $\Pi = (O, E, w_1, w_2, w_3, R, 1)$, with $O = \{A, B, C, X, Y\}$, $E = \{A, B, C\}$, $w_1 = \{X\}$, $w_2 = \{Y\}$, $w_3 = \{B\}$. The set of rules R is defined as follows:

$1 : (1, X/A, 0)$ $2 : (0, X, 1)$ $3 : (1, X/Y, 2)$ $4 : (1, A/B, 2)$ $5 : (1, Y, 0)$
$6 : (2, X/B, 3)$ $7 : (2, B/CC, 0)$ $8 : (1, C/B, 0)$ $9 : (1, AB/X, 3)$

These rules induce the graph G shown at the right:



We affirm that Π generates the set $\{2^n \mid n \geq 0\}$. Indeed, the computation in Π can be split into two stages. During the first stage $k - 1$ symbols A are brought to cell 1 using rules 1 and 2. The second stage starts by using the rule 3 and 6. At this moment there are k symbols A in cell 1 and one copy of B in cell 2. Rules 7 and 8 double the number of B's in cell 2. Suppose that there are m copies of B in cell 2. At any time it is possible to use rule 4 up to

$\min(k, m)$ times. If it is used p times, $p < k$, then this leads to a configuration when both symbols A and B are present in cell 1 and on the next step rule 9 should be used which brings symbol X to cell 1 and starts an infinite computation because there is no more symbol Y in cell 2. If rule 4 is used k times and $m \neq k$, then the computation does not stop because rules 7 and 8 are always applicable. So, the only way to stop the computation is when $m = k$, hence $k = 2^n$, $n > 0$.

This example uses the *pumping* technique which consists in two stages: first a big (unbounded) number of working symbols is introduced into the system and after that the system computes using these additional symbols. In the case above, the pumping phase introduce symbols A which permit to verify further that all symbols B were moved to cell 1.

The investigation of membrane systems with symport and antiport rapidly showed that corresponding systems are able to generate all recursively enumerable languages. Then a quest for systems having smallest descriptional complexity began. There are 4 basic descriptional complexity parameters investigated for symport/antiport P systems:

- the number of membranes,
- the size of the alphabet (the number of different objects),
- the size of rules,
- the number of rules.

The first parameter, the number of membranes, can reach its optimal value: one membrane is already sufficient to get the computational completeness with symport or antiport rules of size 3, see [29] and [25].

Theorem 3.2.5. $NOP_1(sym_3) = NOP_1(anti_3) = NRE$.[2]

Moreover, the above systems can deterministically accept *NRE*.

In the biological case symport and antiport operations involve at most two objects. This is why symport/antiport P systems having symport and antiport rules of size at most 2, called minimal symport and minimal antiport rules, are of special interest. We recall the following results.

Theorem 3.2.6. $NOP_1(sym_1, anti_1) \cup NOP_1(sym_2) \subseteq NFIN$.

Theorem 3.2.7. $NOP_2(sym_1, anti_1) = NOP_2(sym_2) = NRE$.

However, the last result is true only for non-deterministic systems. If the system is deterministic, then the following result holds.

Theorem 3.2.8. *For any deterministic P system with rules of type sym_2 and $anti_1$, the number of objects present in the initial configuration of the system cannot be increased.*

The situation changes for the case of tissue P systems where systems with 2 cells can deterministically accept *NRE*, see the results from [7].

The number of rules is investigated in Section 3.4.2, while the results on the number of objects can be consulted in [62].

[2] More precisely, in the symport case 7 additional objects remain, so the result strategy $-k$ shall be applied.

3.2.2 Generalized communication

We can generalize the idea of minimal symport and minimal antiport to a hypergraph structure: two objects change their positions (cells) in a synchronous way, for example, an object a from cell i and an object b from cell j move synchronously to cell k and cell l, respectively. Formally, in network of cell terms, this corresponds to the rule $(a,i)(b,j) \to (a,k)(b,l)$, which we also call *communication rule*. We graphically depict it as on Fig. 3.4. Depending on their form, several *restrictions on communication rules* (modulo symmetry) can be introduced; we provide below a detailed description of these variants.

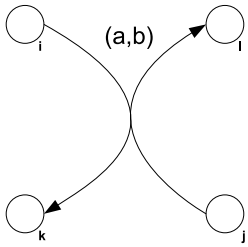


Figure 3.4: The communication rule

Definition 3.2.9. A *generalized communicating P system* (a *GCPS*) of degree n, where $n \geq 1$, is an $(n+4)$-tuple

$$\Pi = (O, E, w_1, \ldots, w_n, R, h)$$

where

1. O is an alphabet, called the *set of objects* of Π;
2. $E \subseteq O$; called the *set of environmental objects* of Π;
3. $w_i \in O^*$, $1 \leq i \leq n$, is the multiset of objects *initially associated to cell* i;
4. R is a finite set of *minimal interaction rules* (or *communication rules*) of the form $(a,i)(b,j) \to (a,k)(b,l)$, where $a, b \in O$, $0 \leq i,j,k,l \leq n$, and if $i = 0$ and $j = 0$, then $\{a,b\} \cap (O \setminus E) \neq \emptyset$; *i.e.*, $a \notin E$ and/or $b \notin E$;
5. $h \in \{1, \ldots, n\}$ is the *output cell*.

In a similar way as it was done for symport/antiport P systems GCPS can be translated to networks of cells as $\Pi = (n+1, O, w, Inf, R)$, where w and Inf are defined as for symport/antiport P systems and R is exactly the same set of rules.

***Remark* 3.2.10.** Besides the non-deterministic maximally parallel semantics that is commonly associated to P systems, other types of behavior may be associated to an interaction rule depending on the number of objects which are processed by each application of such a rule. Here we list some possible alternatives although, we will essentially deal with the usual semantics where each application of a rule processes exactly one occurrence of each object involved.

- One application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects $\min(|w_i|_a, |w_j|_b)$ objects (*i.e.*, $\min(|w_i|_a, |w_j|_b)$ occurrences of a are moved from cell 1 to cell 2, and $\min(|w_i|_a, |w_j|_b)$ occurrences of b are moved from cell 3 to cell 4) ($\langle h : h \rangle$ semantics)).

- One application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects one occurrence of a and all occurrences of b ($\langle 1 : all \rangle$ semantics).

- One application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects h_1 occurrences of a and h_2 occurrences of b, with h_1, h_2 arbitrary values such that $1 \leq h_1 \leq |w_i|_a$, $1 \leq h_2 \leq |w_j|_b$ ($\langle * : * \rangle$ semantics).

- For some given $n, m \geq 1$, one application of a rule $(a,i)(b,j) \to (a,k)(b,l)$ affects at least (resp. at most) n occurrences of a (resp. at most) and at least (resp. at most) m occurrences of b ($\langle \geq n :\geq m \rangle$ semantics (resp. $\langle \leq n :\leq m \rangle$ semantics)).

In general, the behavior of GCPS is different when various semantics are considered. However, special GCPS's can be constructed in such a way that their behavior is the same irrespective of the type of semantics (or at least for some of them).

In the following we recall the notions of the possible restrictions on the interaction rules (modulo symmetry). Let O be an alphabet and let us consider an interaction rule $(a,i)(b,j) \to (a,k)(b,l)$ with $a, b \in O$, $i,j,k,l \geq 0$. Then we distinguish the following cases:

1. $i = j = k \neq l$: the *conditional-uniport-out rule*
   (the *uout* rule, for short) sends b to cell l provided that a and b are in cell i;

2. $i = k = l \neq j$: the *conditional-uniport-in rule*
   (the *uin* rule, for short) brings b to cell i provided that a is in that cell;

3. $i = j$, $k = l$, $i \neq k$: the *symport2 rule*
   (the *sym*2 rule, for short) corresponds to the minimal symport rule, *i.e.*, a and b move together from cell i to k;

4. $i = l$, $j = k$, $i \neq j$: the *antiport1 rule*
   (the *anti*1 rule, for short) corresponds to the minimal antiport rule, *i.e.*, a and b are exchanged in cells i and k;

5. $i = k$ and $i \neq j$, $i \neq l$, $j \neq l$: the *presence-move rule*
   (the *presence* rule, for short) moves the object b from cell j to l, provided that there is an object a in cell i and i,j,l are pairwise different cells;

6. $i = j$, $i \neq k$, $i \neq l$, $k \neq l$: the *split rule*
   (the *split* rule, for short) sends a and b from cell i to cells k and l, respectively;

7. $k = l$, $i \neq j$, $k \neq i$, $k \neq j$: the *join rule*
   (the *join* rule, for short) brings a and b together in cell k;

8. $i = l$, $i \neq j$, $i \neq k$ and $j \neq k$: the *chain rule*
   (the *chain* rule, for short) moves a from cell i to cell k while b is moved from cell j to cell i, *i.e.*, to the cell where a was previously ;

9. i, j, k, l are pairwise different numbers: the *parallel-shift rule* (the *shift* rule, for short) moves a and b from two different cells to another two different cells.

A generalized communicating P system may have rules of several types as defined above. When only one of them is considered, then we call the corresponding GCPS a *minimal interaction P system* (with the given type of rules), or a GCPSMI, for short.

In the following, $NOtP_k(x)$ denotes the set of numbers generated by minimal interaction P systems of degree k and with rules of type x, $k \geq 1$ and $x \in \{uout, uin, sym2, anti1, presence, split, join, chain, shift\}$, and $NOtP_*(x)$ is the notation for $\bigcup_{k=1}^{\infty} NOtP_k(x)$.

If the number of objects in the alphabet of objects in the GCPSMI is m, then the previous notations are changed to $NO_mtP_k(x)$ and $NO_mtP_*(x)$, respectively. We call these GCPSMIs *m-symbol minimal interaction P systems* (with a given type of rules). For simplicity, we use terms 1-symbol GCPSMI and *one-symbol GCPSMI* as equivalent.

We remark that symport/antiport P systems using either minimal symport or minimal antiport rules are not GCPSMIs. This is due to the fact that in symport/antiport P systems uniport rules are allowed. A uniport rule, denoted (i, A, j), move unconditionally the symbol A from cell i to cell j. Such kind of rules does not fit the communication rule pattern, although they can be simulated in some cases.

Due to their simplicity, the generative power of minimal interaction P systems is of particular interest. In [77] and [19] it was shown that $NOtP_*(anti1) \subset NFIN$ and

Theorem 3.2.11.

$$NRE = NOtP_{30}(uin) = NOtP_{30}(uout) = NOtP_9(split) =$$
$$= NOtP_7(join) = NOtP_{36}(presence) = NOtP_{19}(shift) = NOtP_*(chain),$$

We also note that systems with $sym2$ rules can accept any recursively enumerable set of numbers with 10 cells.

Moreover, if the alphabet O is restricted to one symbol, then the following relations hold.

Theorem 3.2.12.

$$NRE = NO_1tP_*(join) = NO_1tP_*(presence) = NO_1tP_*(shift) = NO_1tP_*(chain).$$

Notice that in the case of one-symbol minimal interaction P systems, the concepts of rule types conditional-uniport-out, symport2, antiport1, and split are not applicable. Since $O = E$ in these cases the rules $(\bullet, i)(\bullet, j) \to (\bullet, k)(\bullet, l)$ do not satisfy condition 4. of Definition 3.2.9, namely, that if $i = 0$ and $j = 0$, then $\{\bullet\} \cap (O \setminus E) \neq \emptyset$. For the case of uniport-in we obtain the following result.

Theorem 3.2.13. *For any one-symbol GCPSMI Π with conditional-uniport-in rules either $N(\Pi)$ is finite or there is a natural number K such that $l \in N(\Pi)$ for every $l \geq K$.*

It is worth to note that most of the proofs are compositional: a set of special primitive blocks is proposed and the proof results from their assembly in a special

way; of cause, the implementation of each block for the target system shall be provided. Among most interesting blocks we used so far, we can cite the $1 : all$ block, which is graphically depicted as on Fig. 3.4, but simulates the $\langle 1 : all \rangle$ semantics (using $\langle 1 : 1 \rangle$ semantics). Such a block was extremely useful in constructions, in particular, for the implementation of the n^2 function. For the proof of the computational completeness we used 3 blocks: the uniport block, the main block and the zero block, see Fig. 3.5.

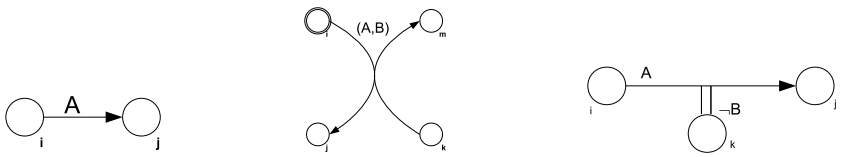


Figure 3.5: The blocks used for computational completeness proofs: (a) the uniport block, (b) the main block, (c) the zero block

The uniport block, see Fig. 3.5(a), is be denoted by an arrow between circles labeled by i and j with a object (say A) on the top of it.

The basic or main block, see Fig. 3.5(b), permits to move synchronously objects A from cell i to cell j and B from cell k to cell m. If B is not present, then an infinite loop occurs. The arrows show the direction of the move of the objects, the circles corresponds to the cells. Since the semantic of the block is not symmetric, the double circle, labeled by i, indicates the place of the symbol that triggers the computation and for which the infinite loop can occur.

The zero block, see Fig. 3.5(c), moves object A from cell i to j providing that there are no objects B in cell k. If there are objects B in cell k then the computation enters an infinite loop. The notations are analogous to the ones used in see Fig. 3.5(b), namely, the arrow denotes the direction of the movement of the object, the circles denote cells, the double line labeled with $-B$ and the circle labeled with k refers to the condition that no object B is present in cell k.

Using these blocks, a typical computational completeness proof of the minus instruction of the register machine looks like on Fig. 3.6.

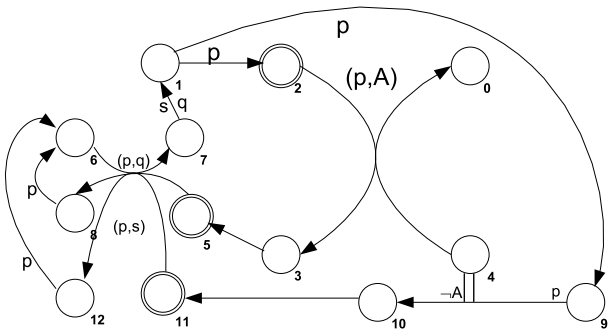


Figure 3.6: A typical construction for the simulation of a decrement instruction $(p, A-, q, s)$ of a register machine using building blocks.

3.3 The synchronization problem for P systems

In the framework of P systems usually the computational power is investigated. Here, we present a different kind of research: we adapt to P systems a known problem from the area of cellular automata and give algorithms solving it.

The synchronization problem can be formulated in general terms with a wide scope of application. We consider a system constituted of explicitly identified elements and we require that starting from an initial configuration where one element is distinguished, after a finite time, all the elements which constitute the system reach a common feature, which we call **state**, all at the same time and the state was never reached before by any element.

This problem is well known for cellular automata, where it was intensively studied under the name of the *firing squad synchronization problem* (FSSP): a line of soldiers have to fire at the same time after the appropriate order of a general which stands at one end of the line, see [31, 56, 55, 71, 66, 81]. The first solution of the problem was found by Goto, see [31]. It works on any cellular automaton on the line with n cells in the minimal time, $2n-2$ steps, and requiring several thousands of states. A bit later, Minsky found his famous solution which works in $3n$, see [56] with a much smaller number of states, 13 states. Then, a race to find a cellular automaton with the smallest number of states which synchronizes in $3n$ started. See the above papers for references and for the best results and for generalizations to the planar case, see [71] for results and references.

The synchronization problem appears in many different contexts, in particular in biology. As P systems model the work of a living cell constituted of many microorganisms, represented by its membranes, it is a natural question to raise the same issue in this context. Take as an example the meiosis phenomenon, it probably starts with a synchronizing process which initiates the division process. Many studies have been dedicated to general synchronization principles occurring during the cell cycle; although some results are still controversial, it is widely recognized that these aspects might lead to an understanding of general biological principles used to study the normal cell cycle, see [68].

We may translate FSSP in P systems terms as follows. Starting from the initial configuration where all membranes, except the root, contain same objects the system must reach a configuration where all membranes contain a distinguished symbol, F. Moreover, this symbol must appear in all membranes only during at the synchronization time.

We study the synchronization problem as defined above for two classes of P systems: transitional P systems and P systems with promoters and inhibitors.

In the sequel, we will use transitional P systems without a distinguished compartment as an output, i_0, as this is not relevant for FSSP.

We translate the FSSP to P systems as follows:

Problem 3.3.1. *For a class of P systems C find two multisets $\mathcal{W}, \mathcal{W}' \in O^*$, and two sets of rules $\mathcal{R}, \mathcal{R}'$ such that for any P system $\Pi \in C$ of degree $n \geq 2$ having*

> *$w_1 = \mathcal{W}'$, $R_1 = \mathcal{R}'$, $w_i = \mathcal{W}$ and $R_i = \mathcal{R}$ for all i in $\{2..n\}$, assuming that the skin membrane has the number 1*

it holds

- *If the skin membrane is not taken into account, then the initial configuration of the system is stable (cannot evolve by itself).*

- *If the system halts, then all membranes contain the designated symbol F which appears only at the last step of the computation.*

We give two solutions, using a deterministic and a non-deterministic strategy. It should be noticed that in the non-deterministic case the concept of the synchronization is somehow different and the traditional intuition is misleading. It should be understand that in this case the synchronization is performed only on the computational paths leading to a successful computation.

3.3.1 Non-deterministic case

In this section we discuss a non-deterministic solution to the FSSP using transitional P systems. We recall that a transitional P system has a tree structure and rules of the following type: $(u, i) \to (v_1, j_1) \dots (v_n, j_n)$, where j_k, $1 \leq k \leq n$ is either i, or a parent or child membrane of i.

The main idea of the synchronization is based on the fact that if a signal (materialized by an object) is sent from the root to a leaf then it will take at most $2h$ steps to reach the leaf and return back to the root. In the meanwhile, the root may guess the value of h and propagate it step by step down the tree. This takes also $2h$ steps: h for the guess at the root, and h to end the propagation and synchronize. Hence, if the signal sent to the leaf, having depth $d \leq h$, returns at the same moment that the root ended the propagation, then the root guessed the value d. Now, in order to finish the construction it is sufficient to cut off cases when $d < h$.

In order to implement the above algorithm in transitional P systems we use following steps (the formal description of the construction can be found in [6]).

- Mark leaves and nodes (nodes by $\overline{S}$ and leaves by S).

- From the root, send a copy of symbol a down. Any inner node must take one a in order to pass to state S'. If some node is not passed to state S' then when the signal c will come inside, it will be transformed to #.

- The end of the guess is marked by signal c. Symbols S in leaves are transformed to S''' and those in inner nodes to S''.

- In the meanwhile the height is computed with the help of C_3. If a smaller height $d \leq h$ is obtained at the root node, then either the symbol C_3 will arrive to the root node, which will contain some symbols b – then the symbol # will be introduced at the root node, or the guessed value will be d and then there will be an inner node with $\overline{S}$ or a leaf with S (because we have at most d letters a) which leads to the introduction of # in corresponding node.

The sets of rules used to implement this behavior are all equal and they are described below (the *in*! target sends the symbol to all child nodes).
Start:

$$S_1 \to (S_2, here)(C_2, here)(S, in!)(C_1, in)$$

Propagation of S:

$$S \to (\overline{S}, here)(S, in!)$$

Root counter (guess):

$$S_2 \to (S_2, here)(b, here)(a, in!) \qquad S_2 \to (S_3, here)(c, in!)$$

Propagate a:

$$\overline{S}a \to (S', here) \qquad a \to (b, here)(a, in!)$$

Propagate c:

$$cS' \to (S'', here)(c, in!) \qquad cSa \to (S''', here)$$

Decrement:

$$S''b \to (S'', here) \qquad S'''a \to (S''', here)$$
$$S'' \to (F, here) \qquad S''' \to (F, here)$$
$$S_3 b \to (S_3, here)$$

Height computing:

$$C_1 \to (C_1, in) \qquad C_2 \to (C_2, in)$$
$$C_1 C_2 \to (C_3, here) \qquad C_2 \to (\#, here)$$
$$C_3 \to (C_3, out)$$

Root firing:

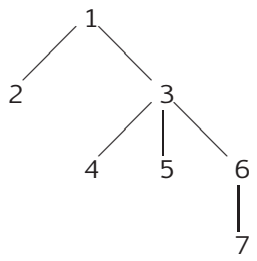
$$C_3 S_3 \to (F, here)$$

Traps:

$$c\overline{S} \to (\#, here) \qquad cS \to (\#, here) \qquad C_3 \to (\#, here)$$
$$aF \to (\#, here) \qquad bF \to (\#, here) \qquad \# \to (\#, here)$$

Example 3.3.2.

We discuss the functioning of the system on the following example. Consider a system Π with 7 membranes and the following membrane structure:



Now consider the evolution of the system Π constructed as above. We represent it in a table format where each cell indicates the contents of the corresponding membrane at the given time moment. Since the evolution is non-deterministic, we consider firstly the correct evolution and after that we shall discuss unsuccessful cases.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\overline{S}aC_2$	S	S	SC_1	
3	S_2bb	Saa	$S'a$	S	S	$\overline{S}C_2$	SC_1
4	S_2bbb	$Saaa$	$S'ba$	Sa	Sa	$\overline{S}a$	SC_1C_2
5	S_3bbb	$Saaac$	$S'bbc$	Saa	Saa	$S'a$	SC_3
6	S_3bb	$S'''aa$	$S''bb$	$Saac$	$Saac$	$S'bcC_3$	Sa
7	S_3b	$S'''a$	$S''bC_3$	$S'''a$	$S'''a$	$S''b$	Sac
8	S_3C_3	S'''	S''	S'''	S'''	S''	S'''
9	F	F	F	F	F	F	F

The system will fail in the following cases:

1. Signals C_1 and C_2 go to different membranes.
2. Some symbol $\overline{S}$ is not transformed to S' (or the deepest leaf does not contain a letter a).
3. S_3 appears in the root membrane after C_3 appears in a leaf.
4. The branch chosen by C_3 is not the longest (it has the depth d, $d < h$).

A possible evolution for the first unsuccessful case is given below.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	SaC_2	$\overline{S}a$	S	S	SC_1	
3	S_2bb	$Saa\#$	$S'a$	S	S	$\overline{S}$	SC_1

A possible evolution for the second unsuccessful case is given below.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\overline{S}aC_2$	S	S	SC_1	
3	S_2bb	Saa	$\overline{S}ba$	Sa	Sa	$\overline{S}aC_2$	SC_1
4	S_2bbb	$Saaa$	$\overline{S}bba$	Saa	Saa	$S'a$	SC_1C_2
5	S_3bbb	$Saaac$	$\overline{S}bbbc$	$Saaa$	$Saaa$	$S'ab$	aSC_3
6	S_3bb	$S'''aa$	$\#bbb$	$Saaac$	$Saaac$	$S'bbcC_3$	Saa

A possible evolution for the third unsuccessful case is given below.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\overline{S}aC_2$	S	S	SC_1	
3	S_2bb	Saa	$S'a$	S	S	$\overline{S}C_2$	SC_1
4	S_2bbb	$Saaa$	$S'ba$	Sa	Sa	$\overline{S}a$	SC_1C_2
5	S_2bbbb	$Saaaa$	$S'bba$	Saa	Saa	$S'a$	aSC_3
6	S_3bbbb	$Saaaac$	$S'bbbc$	$Saaa$	$Saaa$	$S'baC_3$	Sa
7	S_3bbb	$S'''aaa$	$S''bbbC_3$	$Saaac$	$Saaac$	$S'bbc$	Saa
8	S_3bbC_3	$S'''aa$	$S''bb$	$S'''aa$	$S'''aa$	$S''bb$	$Saac$
9	$S_3b\#$	$S'''a$	$S''b$	$S''a$	$S'''a$	$S''b$	$S'''a$

A possible evolution for the fourth unsuccessful case is given below.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2	S	SC_1				
2	S_2b	Sa	$\bar{S}aC_2$	S	SC_1	S	
3	S_2bb	Saa	$S'a$	S	SC_1C_2	$\bar{S}$	S
4	S_2bbb	$Saaa$	$S'ba$	Sa	SaC_3	$\bar{S}a$	S
5	S_3bbb	$Saaac$	$S'bbcC_3$	Saa	Saa	$S'a$	S
6	S_3bbC_3	$S'''aa$	$S''bb$	$Saac$	$Saac$	$S'bc$	Sa
7	$S_3b\#$	$S'''a$	$S''bC_3$	$S'''a$	$S'''a$	$S''b$	Sac

3.3.2 Deterministic case

Consider now the deterministic case. We take the class of P systems with promoters and inhibitors and solve Problem 3.3.1 for this class. We recall that a P system with promoters and inhibitors has a tree structure and rules of form $(u,i) \to (v_1,j_1)\dots(v_n,j_n);(p,i);(f,i)$, where j_k, $1 \le k \le n$ is either i, or a parent or child membrane of i. We will write such rules as $(u,i) \to (v_1,j_1)\dots(v_n,j_n)\,|_{p,\neg f}$.

The idea of the algorithm is very simple. A symbol C_2 is propagated down to the leaves and at each step, being at a inner node, it sends back a signal C. At the root a counter starts to compute the height of the tree and it stops if and only if there are no more signals C. It is easy to compute that the last signal C will arrive at time $2h-1$ (there are h inner nodes, and the last signal will continue for $h-1$ steps). At the same time the height is propagated down the tree as in the non-deterministic case.

The sets of rules used to implement this behavior are all equal and they are described below.

Start:

$$S_1 \to (S_2, here), (C_2', here), (S, in!), (C_1, in!)$$

Propagation of S:

$$S \to (\bar{S}, here), (S, in!)$$

Propagation of C (height computing signal):

$$C_1 \to (C_1, in!) \qquad C_2 \to (C, here), (C_2, in!), (C, out)$$
$$C_1C_2 \to \lambda \qquad C_2' \to (C, here), (C_2, in!)$$
$$C \to (C, out)$$

Root counter:

$$S_2 \to (S_3, here) \qquad S_3 \to (S_3', here), (b, here), (a, in!)\,|_C$$
$$C \to \lambda\,|_{S_3} \qquad S_3' \to (S_3, here)\,|_C$$
$$C \to \lambda\,|_{S_3'} \qquad S_3' \to (S_4, here), (a', in!)\,|_{\neg C}$$

Propagation of a:

$$\bar{S}a \to (S', here) \qquad a \to (b, here), (a, in!)\,|_{S'}$$

End propagate of a:

$$a'S' \to (S'', here), (a', in!) \qquad a'Sa \to (S''', here)$$

Decrement:

$$S''b \to (S'', here) \qquad S'''a \to (S''', here)$$
$$S'' \to (F, here) \,|_{\neg b} \qquad S''' \to (F, here) \,|_{\neg a}$$

Root decrement:

$$S_4 b \to (S_4, here) \qquad S_4 \to (F, here) \,|_{\neg b}$$

The correctness of the construction can be argued as follows. It takes $h + 1$ steps for a symbol C_2 to reach all leaves. All this time, symbols C are sent up the tree. It takes further $h - 1$ steps for all symbols C to reach the root node, and one more step until symbols C disappear. Therefore, symbols b appear in the root node every odd step from step 3 until step $2h+1$, so h copies will be made. Together with the production of b^h in the root node, this number propagates down the tree, being decremented by one at each level. For the depth i, the number $h - i$ is represented, during propagation, by the multiplicity of symbols a (one additional copy of a is made) in the leaves and by the multiplicity of symbols b in non-leaf nodes. After $2h+2$ steps, the root node starts the propagation of the countdown (*i.e.*, decrement of symbols a or b). For a node of depth i, it takes i steps for the countdown signal (a') to reach it, another $h - i$ steps to eliminate symbols a or b, so every node fires after $2h + 2 + i + (h - i) + 1 = 3h + 3$ steps after the synchronization has started.

Example 3.3.3.

Consider a P system having same membrane structure as the system from Example 3.3.2. We present below the evolution of the system in this case.

Step	w_1	w_2	w_3	w_4	w_5	w_6	w_7
0	S_1						
1	S_2C_2'	SC_1	SC_1				
2	S_3C	SC_1C_2	$\overline{S}C_2$	SC_1	SC_1	SC_1	
3	$S_3'bC$	Sa	$\overline{S}aC$	SC_1C_2	SC_1C_2	$\overline{S}C_2$	SC_1
4	S_3bC	Sa	$S'C$	S	S	$\overline{S}C$	SC_1C_2
5	$S_3'bbC$	Saa	$S'aC$	S	S	$\overline{S}$	S
6	S_3bbC	Saa	$S'b$	Sa	Sa	$\overline{S}a$	S
7	$S_3'bbb$	$Saaa$	$S'ba$	Sa	Sa	S'	S
8	S_4bbb	$a'Saaa$	$a'S'bb$	Saa	Saa	$S'a$	S
9	S_4bb	$S'''aa$	$S''bb$	$a'Saa$	$a'Saa$	$a'S'b$	Sa
10	S_4b	$S'''a$	$S''b$	$S'''a$	$S'''a$	$S'b$	$a'Sa$
11	S_4	S'''	S''	S'''	S'''	S'	S'''
12	F	F	F	F	F	F	F

For more details we refer to [10] and [6].

3.4 Construction of small universal P systems

Many variants of P systems generate all recursively enumerable sets of numbers. This means that it is possible to construct a universal P system, *i.e.*, a fixed system that will compute any partially recursive function if a corresponding input is provided. We consider below the case of symport/antiport P systems and provide such a construction. In a similar way, there exist variants of P systems working with strings instead of objects that generate or recognize all recursively enumerable languages. We consider splicing P systems and construct a universal system of this type.

Universal devices are usually studied from the descriptional complexity point of view; in most of the cases the number of rules is taken as the main parameter, as for example for Turing machines. We continue this line and provide not only universal systems for the two classes mentioned before, but also minimize the number of used rules.

3.4.1 Small universal splicing P systems

We should start with the definition of splicing P systems.

Definition 3.4.1. A *splicing tissue P system* of degree $m \geq 1$ is a construct

$$\Pi = (V, T, G, A_1, \ldots, A_m, R_1, \ldots, R_m),$$

where V is a finite alphabet, $T \subseteq V$ is the terminal alphabet and G is the underlying directed labeled graph of the system. The graph G has m nodes (cells) numbered from 1 to m. Each node i contains a set of strings (a language) A_i over V. Symbols R_i, $1 \leq i \leq m$ are finite sets of rules (associated to nodes) of the form $(r; tar_1, tar_2)$, where r is a splicing rule: $r = u_1\#u_2\$u_3\#u_4$ and $tar_1, tar_2 \in \{here, go_j, out\}$, $1 \leq j \leq m$ are target indicators. We remark that the communication graph G can be deduced from the sets of rules. More precisely, G contains an edge (i, j), iff there is a rule $(r; tar_1, tar_2) \in R_i$ with $tar_k = go_j$, $k \in \{1, 2\}$. If one of tar_k is equal to *here*, then G contains the loop (i, i).

A *configuration* of Π is the m-tuple $(N_1, \ldots, N_m)$, where $N_i \subseteq V^*$. A *transition* between two configurations $(N_1, \ldots, N_m) \Longrightarrow (N'_1, \ldots, N'_m)$ is defined as follows. In order to pass from one configuration to another, splicing rules of each node are applied in parallel to all possible words that belong to that node. After that, the result of each splicing is distributed according to target indicators. More exactly, if there are x, y in N_i and $r = (u_1\#u_2\$u_3\#u_4; tar_1, tar_2)$ in R_i, such that $(x, y) \vdash_r (w, z)$, then words w and z are sent to nodes indicated by tar_1, respectively tar_2. We write this as follows $(x, y) \vdash_r (w, z)(tar_1, tar_2)$. If $tar_k = here$, $k = 1, 2$ then the word remains in node i (is added to N'_i); if $tar_k = go_j$, then the word is sent to node j (is added to N'_j); if $tar_k = out$, the word is sent outside of the system.

Since the words are present in an arbitrary number of copies, after the application of rule r in node i, words x and y are still present in the same node.

A *computation* in a splicing tissue P system Π is a sequence of transitions between configurations of Π which starts from the initial configuration $(A_1, \ldots, A_m)$. The result of the computation consists of all words over terminal alphabet T which are sent outside the system at some moment of the computation. We denote by $L(\Pi)$ the language generated by system Π.

We also define the notion of an *input* for the system above. An input word for a system Π is simply a word w over the non-terminal alphabet of Π. The computation of Π on input w is obtained by adding w to the axioms of A_1 and after that by evolving Π as usual. We denote by $L(\Pi, w)$ the result of the computation of Π on w.

We denote by $ELStP_m(spl, go)$ the family of languages generated by tissue splicing P systems having a degree at most m.

We consider the following restricted variant of splicing tissue P systems. A *restricted splicing tissue P system* is a special class of splicing tissue P systems which has the property that for any rule $(r; tar_1, tar_2)$ either $tar_1 = tar_2 = go_j$, or $tar_1 = tar_2 = out$ or $tar_1 = tar_2 = here$. This means that both resulting strings are moved over the same connection. In this case, we may associate splicing rules to corresponding edges. If both targets are *out*, then we can associate the splicing rule with an edge going to the special node called *out*.

The universality proof is based on a simulation of tag systems using the well known rotate-and-simulate method [58, 61, 73]. We show that the functioning of any tag system can be simulated using a restricted splicing tissue P system with 6 rules. Then its universality follows from the existence of universal tag systems.

Let $V = \{a_1, \ldots, a_{n+1}\}$ be an alphabet. Consider coding morphisms c and $\overline{c}$ defined as follows: $c(a_i) = a^i\beta$, $\overline{c}(a_i) = \beta a^i$, where $1 \leq i \leq n+1$.

Theorem 3.4.2. *Let $TS = (2, V, P)$ be a tag system and $w \in V^*$. Then, there is a restricted splicing tissue P system $\Pi = (V', T, G, A_1, A_2, A_3, R_1, R_2, R_3)$, having 6 rules, which given the word $X\beta\beta c(w)\beta Y$ as input simulates TS on input w, i.e. such that:*

1. *for any word w on which TS halts producing the result w', the system Π produces a unique result $X'c(w')Y'$, i.e. $L(\Pi) = \{X'c(w')Y'\}$.*
2. *for any word w on which TS does not halt, the system Π computes infinitely without producing a result, i.e. $L(\Pi) = \emptyset$.*

We construct the system Π as follows. Let $|V| = n+1$. Suppose that the halting symbol of TS is a_1. We use following alphabets V' and T.

$$V' = \{a, \beta, X, X', Y, Y', Z, Z'\}, \qquad T = \{X', Y', a, \beta\}.$$

The initial languages A_j, $j \in \{1, 2, 3\}$ are given as follows.

$$A_1 = \{Z'c(P_i)\overline{c}(a_i)Y \mid a_i \to P_i \in P, 2 \leq i \leq n+1\} \cup \{X\beta Z,\ ZY,\ Z'Y'\},$$
$$A_2 = \{XZ\},$$
$$A_3 = \{XZ,\ X'Z\}.$$

The set of rules R_j, $j \in \{1, 2, 3\}$ are given as follows.

$R_1 = \{1.1 : (\varepsilon\#\beta Y\$Z'\#\varepsilon; go_3, go_3);\ 1.2 : (\varepsilon\#aY\$Z\#Y; go_2, go_2);\ 1.3 : (X\beta a\#\varepsilon\$x\beta\#Z; here, here)\}$;
$R_2 = \{2.1 : (Xa\#\varepsilon\$X\#Z; go_1, go_1)\}$;
$R_3 = \{3.1 : (X\beta\beta\#aa\$X\#Z; go_1, go_1);\ 3.2 : (X\beta\beta\#a\beta\$X'\#Z; out, out)\}$.

The graph G is represented on Fig. 3.7.

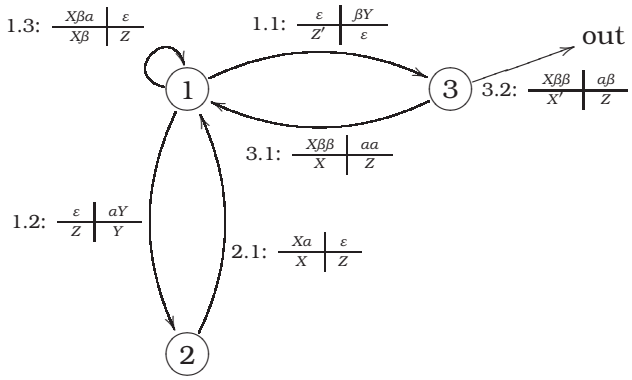


Figure 3.7: The communication graph G associated to the construction from Theorem 3.4.2.

The simulation of TS is performed as follows. For every step of the derivation in TS there is a sequence of several derivation steps in Π. The current configuration w of TS is encoded by a string $X\beta\beta c(w)\beta Y$ present in node 1 of Π (the initial configuration of Π satisfies this property). The simulation of a production $a_i \to P_i$, $2 \leq i \leq n+1$ is performed using the rotate-and-simulate method used for many proofs in this area. This method works as follows. First, suffixes $c(P_j)\overline{c}(a_j)$, $1 \leq j \leq n$ are attached to the string producing $Xa^i\beta c(a_k w')c(P_j)\beta a^j Y$. After that symbols a are decreased at both ends simultaneously. Hence, only the string for which $j = i$ will remain at the end, producing $X\beta c(a_k w')\beta Y$. After that the symbol a_k is removed (by removing corresponding a's) and a new round begins. The simulation stops when the first symbol is a_1.

3.4.2 Small universal antiport P systems

In this section we construct a universal antiport system. For the sake of commodity we use the representation of P systems in terms of maximally parallel multiset rewriting. We start by introducing corresponding terms and after that we give the construction.

Taking a antiport P system with one membrane $\Pi = (V, R, T, [_1]_1, \mathcal{I}, \mathcal{P})$ we call R the alphabet of *registers* or the *data* alphabet. A *state* configuration is the projection of a configuration of Π over $O \setminus R$ (hence the symbols over the registers alphabet are not included in the state configuration). A state configuration B is *reachable* in one step from the state configuration A if there are multisets R', R'' over R such that there exists a maximally parallel transition $AR' \Longrightarrow BR''$. We will denote this by $A \Rrightarrow B$. We remark that there might be several configurations reachable in one step from a particular configuration A.

Since for systems with one membrane an antiport rule $(u, in; v, out)$ corresponds to a multiset rewriting rule $v \to u$ we use the latter notation for the rule specification. We say that a rule $u \to v$ is a *pure state* rule if u contains no symbols from R, otherwise we call it a *register-dependent* rule.

We call the obtained systems finite state maximally parallel multiset rewriting systems (FsMPMRS). Such a system will be denoted as $\gamma = (V, R, T, \mathcal{I}, \mathcal{P})$, where

V, R, $\mathcal{I}$ and $\mathcal{P}$ are the same as for the antiport system, and $T \subseteq R$. The result of the computation is the multiset $M(\gamma)$ defined as $M(\gamma) = \{pr_T(w) \mid \mathcal{I} \Longrightarrow^* w,$ and w cannot evolve anymore$\}$.

For more precise details on the definition and the translation to and from antiport P systems we refer to [8].

Graphical notation.

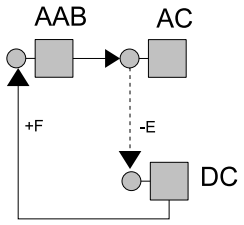
We introduce a graphical notation for FsMPMRS. We represent a state configuration by a filled square with a dot attached to it and rules by arrows. We also suppose that pure state rules (denoted by solid lines) precede register-dependent rules (denoted by dashed lines). Now, in order to represent the relations between state configurations we will depict the relation $\Rrightarrow$. Without entering into technical details, applying any maximally parallel transition to a configuration X means to start from the square $pr_{O\setminus R}(X)$ and follow arrows to circles as long as possible, keeping track of symbols from R; when it is no longer possible, consider the square to which the last circle is attached.

Example 3.4.3.

Consider the system $\gamma = (\{A, B, C, D\}, \{E, F\}, \{F\}, \{AABEE\}, \mathcal{P})$, where $\mathcal{P}$ contains the following rules:

$$r_1 : AB \to C$$
$$r_2 : AE \to D$$
$$r_3 : DC \to AABF$$

Clearly, the system γ is a FsMPMRS that computes the multiset {FF}. Indeed, there are three state configurations AAB, AC and CD and there are no rules involving only E or F in the left-hand side. In a graphical way this system is represented as follows:



For example, in order to compute the first step using the diagram we proceed as follows. The initial configuration $AABEE$ corresponds to the state configuration AAB. Now rule r_1 can be applied bringing us to the circle attached to the square AC. Since r_2 is applicable we can continue to the circle attached to the square DC and take into account that one E is removed. Finally, since there are no more outgoing arrows, we stop at the square DC and our configuration is DCE.

We remark that the introduced graphical notation is different from the Molecular Interactions Maps [44] or Kitano [42] notations, used for specifications in systems biology, because it permits to follow the functioning of the system.

The Basic Simulation Technique

In this section we concentrate on a simple simulation of register machines by FsMPMRS. This simulation is done as follows. We represent the current configuration of a register machine M by a multiset (initially $\mathcal{I}$). In particular, the contents of a register $R_i \in R$ is represented by the number of symbols R_i which are present. The simulation of any incrementing or decrementing instruction of M is done by an appropriate set of rules.

Our construction is based on a simulation of a special universal register machine U_{32} having 32 instructions taken from [45]. This construction may be rewritten in terms of $[RiP]$ (increment) and $\langle RiZM \rangle$ (decrement) instructions, which gives 22 rules (9 incrementing instructions and 13 decrementing), see Fig. 3.8.

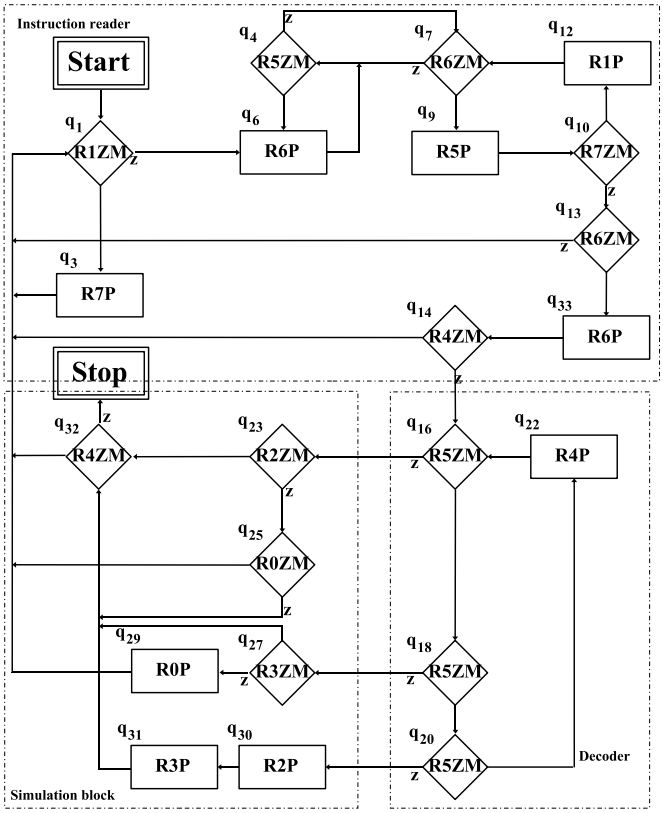
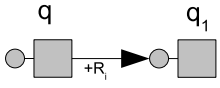


Figure 3.8: The universal register machine U_{32} with $[RiP]$ and $\langle RiZM \rangle$ instructions.

The basic simulation strategy consists in a simulation of rules of an arbitrary register machine by multiset rewriting rules using the smallest number of the latter ones. Any incrementing rule $(q, [RiP], q_1)$ of register machine can be directly simulated by the rule

$$q \to R_i q_1, \tag{3.1}$$

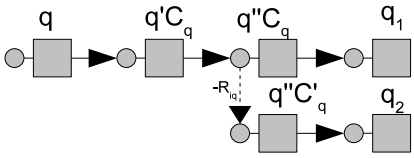
This corresponds to the following flowchart:



Any decrementing rule $(q, \langle Ri_qZM \rangle, q_1, q_2)$ can be simulated using five rules:

$$\begin{array}{lll} q \to q' C_q, & & \\ q' \to q'', & C_q R_{i_q} \to C'_q, & \\ q'' C_q \to q_1, & q'' C'_q \to q_2 & \end{array} \tag{3.2}$$

This corresponds to the following flowchart:



This simulation is done as follows. Symbol q introduces symbols q' and C_q (the last one is called the *checker* for the state q).

After that symbol C_q tries to decrease register R_{i_q} and if it succeeds then it becomes C'_q. Now, depending on this information symbol q'', which replaced q', will choose the corresponding new state.

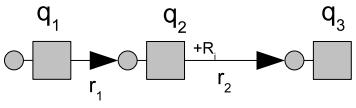
The choice between configurations $q'' C_q$ and $q'' C'_q$ depends on the presence of symbol R_{i_q}, *i.e.*, if register R_{i_q} is zero.

Applied to U_{32} this translation gives an FsMPMRS with 73 rules. We remark that these rules are of size at most 3. In the following sections we show different techniques which reduce the number of rules for the price of increasing their size.

Basic minimization strategies

In the following we present two basic minimization strategies. One of them is based on structural improvements and the other one is based on encodings. We present them in a general form and after that we show how they apply to the system that simulates U_{32}.

State Elimination This minimization strategy performs an elimination of linear fragments in the flow-chart (by performing a kind of speed-up). Suppose that there are following two pure state rules, $r_1 = (q_1 \to q_2)$ and $r_2 = (q_2 \to q_3 R_i)$. This corresponds to the flowchart in the picture.



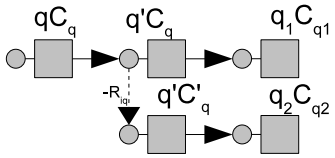
We observe that rules r_1 and r_2 may be combined and state q_2 may be eliminated by introducing a new rule $r = (q_1 \to q_3 R_i)$. In a similar way, any linear chain of pure state rules may be collapsed to a single rule (the size may be increased for each additional rule). We shall further refer to this technique as *intermediate state elimination*.

For U_{32} we observe that using intermediate state elimination technique we can reduce (3.2) to following rules (we also renamed q' to q and assume that the initial

state is encoded as $q_o C_{q_0}$):

$$\begin{aligned} q &\to q', & C_q R_{i_q} &\to C'_q, \\ q' C_q &\to q_1 C_{q_1}, & q' C'_q &\to q_2 C_{q_2} \end{aligned} \tag{3.3}$$

Graphically this can be represented as follows:



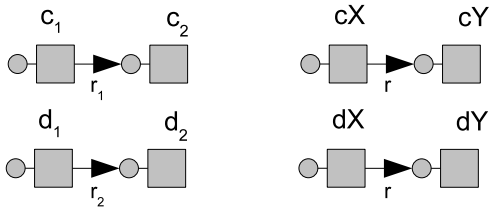
We observe that compared to the previous picture of the decrement simulation the state q was eliminated and now the simulation starts with the state qC_q.

Moreover, we observe that for U_{32} in a most of the cases a decrementing instruction $(q, \langle RiZM \rangle, q_1, q_2)$ is followed by an incrementing instruction $(q_1, [Rk_1P], q_3)$ or $(q_2, [Rk_2P], q_4)$. Hence, one can simulate the incrementing instruction during the simulation of the previous decrementing instruction (by eliminating the unneeded state in between). For example, last two rules from (3.3) become

$$q' C_q \to q_3 R_{k_1} C_{q_3}, \qquad q' C'_q \to q_4 R_{k_2} C_{q_4}. \tag{3.4}$$

Of course, this increases the size of rules up to 5.

Gluing rules Now we consider a technique that minimize the number of rules by performing transitions between the configurations by fewer rules. Informally, transitions $c_1 \overset{r1}{\to} c_2$ and $d_1 \overset{r2}{\to} d_2$ can be performed by the same rule $X \to Y$ if the configurations are represented in a suitable way: $c_1 = cX$, $c_2 = cY$, $d_1 = dX$, $d_2 = dY$. In this case, we say that r_1 and r_2 may be *glued*. The following picture illustrates this:



In a more formal way one must find a suitable encoding of state configurations such that:

- No state configuration is a submultiset of another state configuration.
- There should be at least 2 transitions that may be glued.

We would like to remark that it is only possible to glue transitions that increment registers equivalently, in particular, transitions that do not increment any register.

In what follows we apply the idea of gluing rules to the FsMPMRS system obtained by the basic simulation technique.

Phases Consider now the rules (3.3). If we represent the state q by qS and the state q' by qS' then the first rule from (3.3) may be glued for all states q, *i.e.*, instead of $|Q|$ rules $q \to q'$ we obtain one rule $S \to S'$. We call the symbol S the *phase*, hence there will be two phases S and S'. The rules from (3.3) are replaced by:

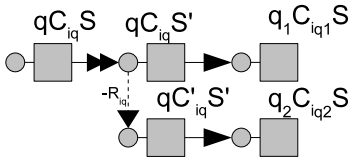
$$\begin{aligned} &C_q R_{i_q} \to C'_q, && (3.5)\\ &qS'C_q \to q_1 C_{q_1} S, \qquad && qS'C'_q \to q_2 C_{q_2} S \end{aligned}$$

Independent Checkers Another minimization idea comes from the observation that the information encoded in the checker C_q from (3.5) is redundant. If we take the set of first rules from (3.5) for all $q \in Q$, we observe that it is possible to glue rules that decrement the same register in the following way. We encode the sequence qC_q, respectively qC'_q, by symbols qC_{i_q}, respectively qC'_{i_q}, where i_q is the number of the register decreased by the instruction q of M. Now we may eliminate the first rule from (3.5) by introducing rules $C_i R_i \to C'_i$, $1 \le i \le |R|$. By convention, we will say that a state q of machine M is encoded by symbols $qC_{i_q}S$ and we will say that C_{i_q} is the checker for the state q. This transforms (3.5) into the following:

$$qS'C_{i_q} \to q_1 C_{i_{q_1}} S, \qquad qS'C'_{i_q} \to q_2 C_{i_{q_2}} S, \tag{3.6}$$

where $C_{i_{q1}}$ and $C_{i_{q2}}$ represent checkers for states q_1 and q_2. Of course this introduces $|R|$ new rules, but finally we gain more because of the elimination of one rule for each $q \in Q$ from (3.5).

Graphically the above two transformations can be represented as follows (the double-headed arrow represents the rule $S \to S'$ common for all simulation blocks):



Further Minimization of U_{32} Simulation

In this section we show how to minimize the simulation of U_{32}. We start with the simulation using rules (3.6) and we do structural improvements based on some observations on the functioning of the system. After that we show how to glue most of the remaining rules by giving a suitable encoding of state configurations.

The structural improvements presented in this section are in some sense a generalization of the intermediate state elimination technique.

Reducing decoder block The first important improvement may be done by considering the decoder part of the machine (see the flowchart on Fig. 3.8). In fact, this block does a division of R_5 by three. This behavior may be simulated by 5 rules which try to decrease register R_5 by 3 and make the choice of the next state depending on the result of this subtraction. The state q_{16} is now encoded by $q_{16}C_5C_5C_5S$.

State Elimination for Decrementing Rules Using state elimination technique it is possible to combine several rules corresponding to a decrement of registers $0-3$ and 7, because these registers are decremented only once. However, for technical reasons 3 phases should be introduced instead of 2. In this case, phase 2 (marked by S') will be treated analogously to phase 1 (marked by S) and the move to the next state will be done in phase 3 (marked by S''). Moreover, the phase change may still be done by one rule. For this it is enough to replace S by XXX, S' by XXT and S'' by XTT and the rules $S \to S'$ and $S' \to S''$ by the rule $XX \to XT$.

The resulting flowchart can be seen at Fig. 3.9. We use following conventions. The double-headed arrow represents the rule $S \to S'$ (and $S' \to S''$) that changes the phase. Rules that decrement registers ($C_i R_i \to C_i'$) are represented by arrows starting with a perpendicular bar and labeled by $D0$–$D7$ enclosed in circle. We also do not depict in this case the corresponding decrementing register. Rules that increment registers are depicted by arrows with a dashed line and the incremented register(s) are depicted beside the line. These rules are labeled by a letter enclosed in a diamond. All other rules (which do not increment/decrement registers) are labeled by a number enclosed in a square.

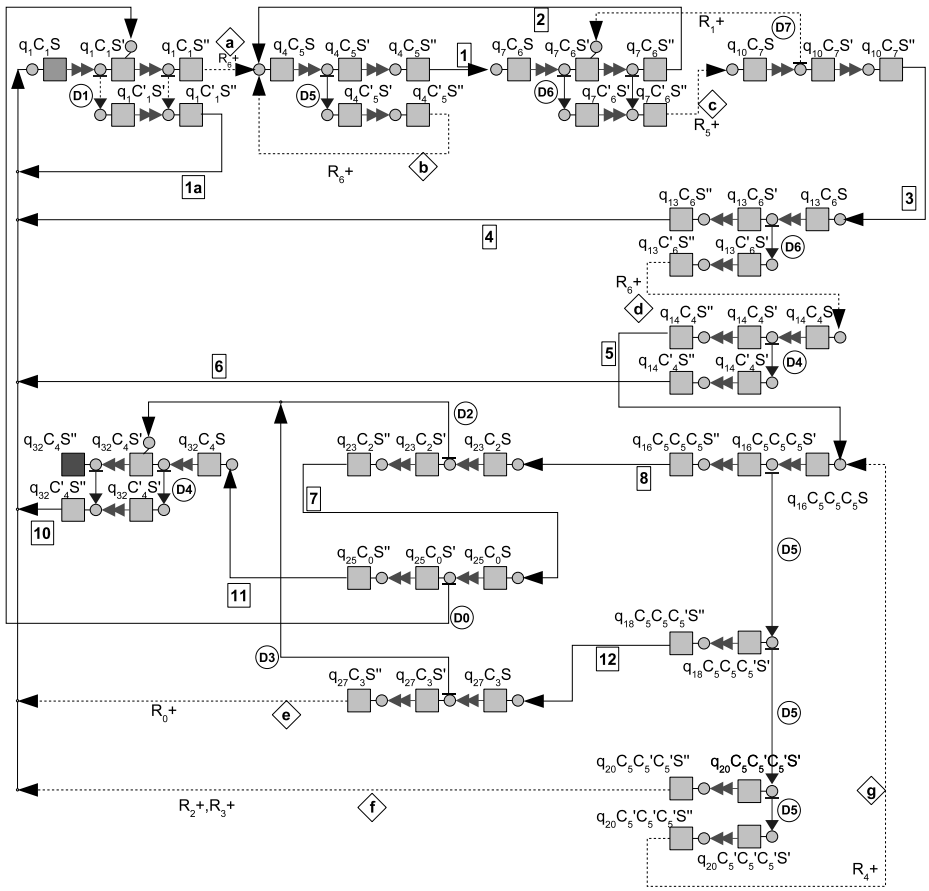


Figure 3.9: The flowchart of the FsMPMRS simulating U_{32} after decoder block reduction and state elimination for decrementing rules.

Encoding optimization Finally, the most substantial decrease can be obtained by a proper encoding of the q_iC_q part of state configurations. The idea is that using maximal parallelism some subparts of q_iC_q can be evolved by a small number of common rules. Without entering into technical detail that can be consulted in [8], we found an encoding of the part of the flowchart that permits to perform 9 rules by only 3 new rules. This encoding is based on 2 phase changing rules $A = (IS'' \to JS)$ and $B = (JJMS'' \to JJNS)$ and on one non-phase rule $C = (LP \to LQ)$. It is depicted on Fig. 3.10.

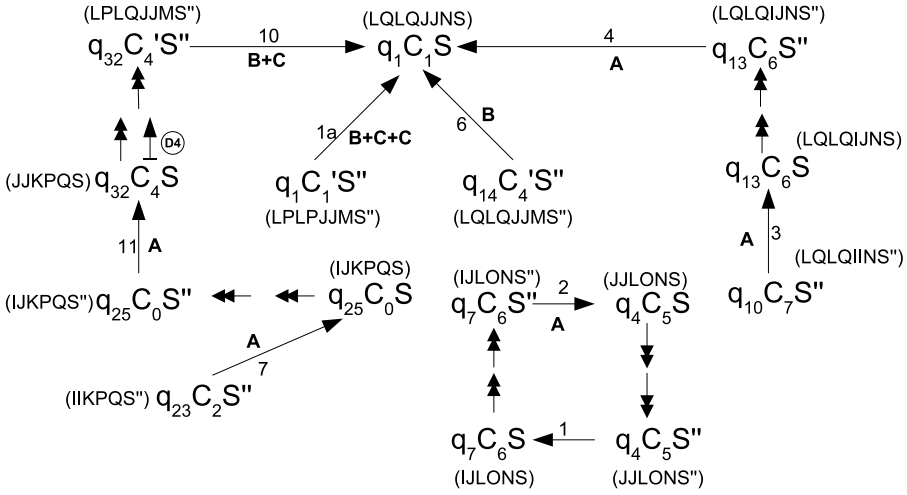


Figure 3.10: Part of the flowchart of the FsMPMRS simulating U_{32} showing only glued rules and the corresponding encoding.

Fig. 3.11 shows the flowchart obtained after applying all ideas mentioned before. Arrows ending with a diamond correspond to rule C, while arrows ending with a square correspond to rule A. The sparse arrow corresponds to rule B.

Formal Description of the System

In this section we give the formal description of the obtained system.

We constructed the system $\gamma = (O, R, \{R_1\}, \mathcal{I}, \mathcal{P})$, where

$$\begin{aligned} O &= R \cup \{C_3, C_5', C_6'\} \cup \{q_{16}, q_{27}\} \cup \{T, I, J, K, L, M, N, O, P, Q, T, X\}, \\ R &= \{R_i \mid 0 \le i \le 7\}, \\ \mathcal{I} &= LQLQJJNXXXR_0^{i_0} \cdots R_7^{i_7}. \end{aligned}$$

Here $i_0, \ldots, i_7$ is the contents of registers 0 to 7 of U_{32} and $LQLQJJNXXX$ is the encoding of the initial state q_1C_1S. The set of rules $\mathcal{P}$ is the following:

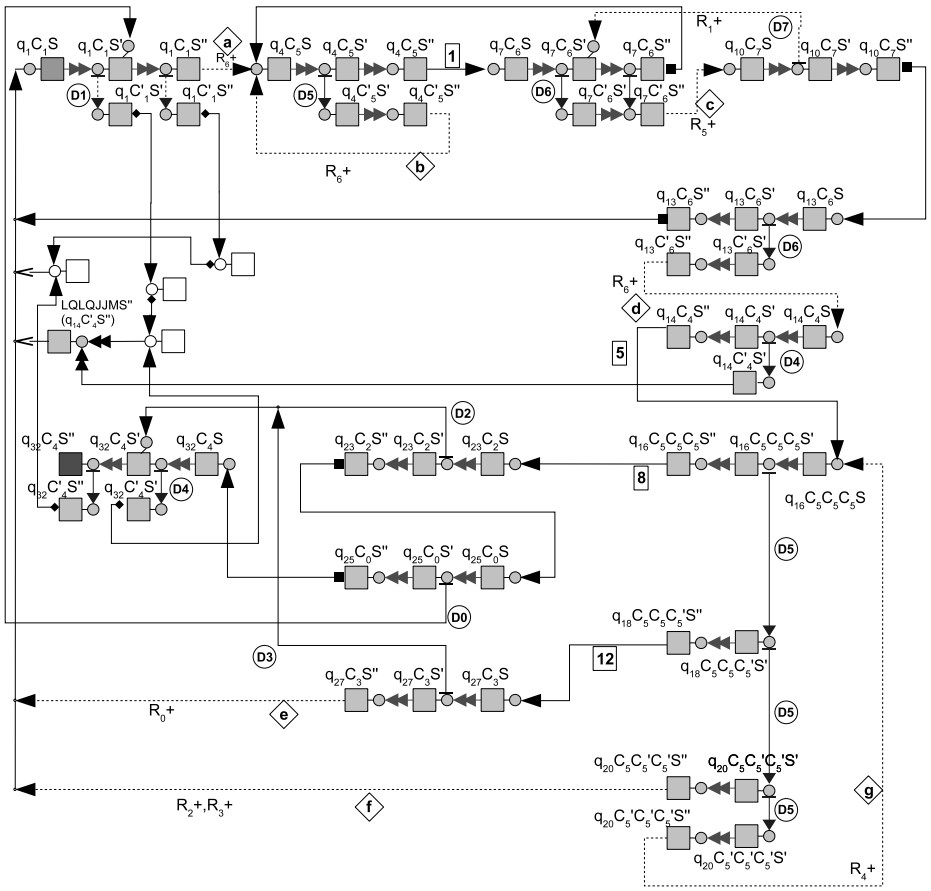


Figure 3.11: The flowchart of the FsMPMRS simulating U_{32} with glued rules.

No	*Rule*	*No*	*Rule*
phase	$XX \to XT$		
*D*0	$IJKPQR_0 \to LQLQJJM$	*a*	$LQLQJJNTT \to JJLOR_6XX$
*D*1	$LQLQJJNR_1 \to LPLPJJMR_7$	*b*	$LC'_5TT \to JJLOR_6XX$
*D*2	$IIKPQR_2 \to JJKPQ$	*c*	$OC'_6TT \to IILQLQNR_5XX$
*D*3	$q_{27}C_3R_3 \to JJKPQ$	*d*	$QLQNC'_6TT \to JJKQQR_6XX$
*D*4	$JJKR_4 \to JJLLM$	*e*	$q_{27}C_3TT \to LQLQJJNR_0XX$
*D*5	$JJOR_5 \to C'_5$	*f*	$q_{16}JJOC'_5C'_5TT \to LQLQJJNR_2R_3XX$
*D*6	$IJLR_6 \to C'_6$	*g*	$q_{16}C'_5C'_5C'_5TT \to q_{16}JJOJJOJJOXX$
*D*7	$IILQLQNR_7 \to IJLOR_1$		
A	$ITT \to JXX$	1	$JJLOTT \to IJLOXX$
B	$JJMTT \to JJNXX$	5	$JJKQQTT \to q_{16}JJOJJOJJOXX$
C	$LP \to LQ$	8	$q_{16}JJOJJOJJOTT \to IIKPQMXX$
		12	$q_{16}JJOJJOC'_5TT \to q_{27}C_3XX$

As a corollary of the previous discussion and the system above we obtain:

Theorem 3.4.4. *There exists an universal antiport P system (or FsMPMRS system) having 23 rules.*

Chapter 4

Length-separation model

The motivation for the length separation model comes from a widely used laboratory technique named *gel electrophoresis*. It is usually performed for analytical purposes at the final stage of the experiment and permits to separate DNA molecules based on their lengths. This technique is based on the fact that DNA molecules are negatively charged. Thus, if they are placed in an electric field, they will move towards the positive electrode. In an ideal solution all molecules will have same speed. If one places molecules in a gel it will act as a molecular sieve, small molecules will move easier (faster) through the gel than big ones, obviously groups of same length will move with the same speed. When the first molecules reach the positive end of the gel, the electric field is deactivated.

In laboratory DNA molecules are placed in several lanes. Thus several separations can be performed, see Fig. 4.1. Usually one of lanes contains molecules of a known length and it is used for calibration. Hence, it is possible to find molecules of a given length in a solution. The precision of this process depends on the gel and the size of molecules. Finally, molecules can be excised from the gel and used in other experiments.

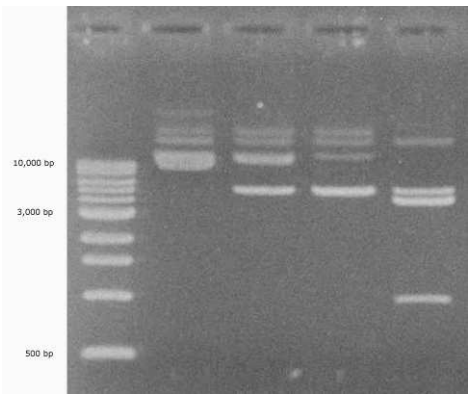


Figure 4.1: Gel electrophoresis

We build our model around the idea of separation of strings (corresponding to DNA molecules) by length, which becomes its main ingredient. In an informal way, our model corresponds to the following experiment. Let us suppose that there is a set of test tubes. Each of these test tubes can transform DNA molecules (cut, ligate,

multiply etc). The tubes are selective and they can do their transformations only on specific molecules (for example, in a tube DNA molecules can be cut with a specific enzyme, hence only molecules having a corresponding site will be modified). Taking a tube, one can add some amount of DNA molecules to it. After the transformation, all molecules from a tube are put in a gel electrophoresis. After the separation, the gel is cut at some points corresponding to some predefined molecular lengths. Hence, molecules will be grouped by length intervals. After that, molecules are extracted from the gel and distributed among other test tubes depending on their molecular length interval. The above process can be iterated and, since all tubes are organized in a network, some interesting transformations may be done. The initial DNA molecules are put in some fixed tube, and the transformed molecules are collected in the output tube.

The translation to the formal language theory can be done as follows. Consider a distributed computing system having the structure of a directed graph. The nodes of this graph contain processing units that generate (in one step) a language starting from a finite set of strings. The edges of the graph are labeled by simple numeric predicates like $= k$, $> k$ or $< k$. The computation in such a model consists of a two-steps repeated procedure. During the *evolution* (first) step the set of strings in all nodes is subjected to the corresponding operation and a new set of strings is obtained in each node. During the *communication* (second) step, the strings are redistributed in parallel between nodes according to numerical predicates. More precisely, if an edge is labeled by the predicate $= k$, then all strings having length k are moved from the initial node to the destination node.

The obtained model is similar to CD grammar systems [14] or networks of language processors [17], where nodes are called components, and to test tube systems [15], where nodes are called test tubes. However, the communication mechanism which uses the length of the string is very different from previous works, where some set inclusion conditions (like presence or absence of symbols) are tested.

4.1 Formal definition

In the following we define the notion of *length-separating splicing test tube systems*. This notion inherits the distributed architecture and most of the main features of the known variants of test tube systems based on splicing, but the communication among the splicing schemes (test tubes) is defined in a significantly different manner, *i.e.*, based on filtering by length. We chose splicing as an example. A similar construction can be done with any other string operation.

Before turning to the model, we need some auxiliary notions. Let V be an alphabet.

1. Let $\pi_{=k} : V^* \to \{\underline{true}, \underline{false}\}$, where $k \geq 1$, be a mapping (a predicate) defined by

$$\pi_{=k}(w) = \begin{cases} \underline{true} & \text{if } |w| = k, \\ \underline{false} & \text{otherwise.} \end{cases}$$

Mappings (predicates) $\pi_{\leq k}$, $\pi_{\geq k}$, $\pi_{>k}$, $\pi_{<k}$, $\pi_{\neq k}$ are defined by modifying the condition $|w| = k$ to $|w| \leq k$, $|w| \geq k$, $|w| > k$, $|w| < k$, and $|w| \neq k$, respectively.

2. For the sake of completeness, we define $\pi_{\geq 0} : V^* \to \{\underline{true}, \underline{false}\}$ with $\pi_{\geq 0}(w) = \underline{true}$ for any w in V^*.

3. Let $\pi_{\max} : V^* \times 2^{V^*} \to \{\underline{true}, \underline{false}\}$ where

$$\pi_{\max}(w, L) = \begin{cases} \underline{true} & \text{if } w \in L \text{ and } |w| \geq |u|, \text{ for any } u \in L \\ \underline{false} & \text{otherwise.} \end{cases}$$

4. Let $\pi_{\min} : V^* \times 2^{V^*} \to \{\underline{true}, \underline{false}\}$ where

$$\pi_{\min}(w, L) = \begin{cases} \underline{true} & \text{if } w \in L \text{ and } |w| = \min_{w' \in L} |w'| \\ \underline{false} & \text{otherwise.} \end{cases}$$

5. We define $\pi_{\neg \max} : V^* \times 2^{V^*} \to \{\underline{true}, \underline{false}\}$ and $\pi_{\neg \min} : V^* \times 2^{V^*} \to \{\underline{true}, \underline{false}\}$ as the negations of predicates of $\pi_{\max}$ and $\pi_{\min}$, respectively.

If no confusion arises, instead of $\pi_{\leq k}$, $\pi_{\geq k}$, $\pi_{>k}$, $\pi_{<k}$, $\pi_{=k}$, $\pi_{\neq k}$ and $\pi_{\max}$, $\pi_{\min}$, $\pi_{\neg \max}$, $\pi_{\neg \min}$, we might also use the notations $\leq k$, $\geq k$, $> k$, $< k$, $= k$, $\neq k$ and max, $\neg$ max, min, $\neg$ min, respectively.

A *length-separating splicing test tube system* is a tuple $\Delta = (n, V, T, G, A, \mathcal{R}, i_0)$, where V is an alphabet, $T \subseteq V$ is the terminal alphabet, G is a labeled graph, called the *communication graph* of Δ, whose nodes are also called *test tubes* (tubes for short), n is the size of the graph G, and i_0 is a node of G, $1 \leq i_0 \leq n$, called the *output tube*. $A = (A_1, \ldots, A_n)$ is the initial configuration of Δ, where A_i is a finite subset of V^*, for $1 \leq i \leq n$, called the set of *axioms* at node i, and $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_n)$, where each $\mathcal{R}_i$, $1 \leq i \leq n$, is a finite set of splicing *rules* over V^*, *i.e.*, the set of rules associated to node i of G.

Each edge from node i to node j of G, for $1 \leq i,j \leq n$, denoted by (i,j), is labeled by a mapping $p_{i,j}$ from the set $\{\pi_{\leq k}, \pi_{\geq k}, \pi_{>k}, \pi_{<k}, \pi_{=k}, \pi_{\neq k} \mid k \geq 1\} \cup \{\pi_{\geq 0}, \pi_{\max}, \pi_{\neg \max}, \pi_{\min}, \pi_{\neg \min}\}$. (Notice that according to the definition, a direct cycle from a node to itself is also possible.)

Furthermore, we require that no word in V^* can satisfy more than one predicate associated to different edges going out from the same node. Thus, the labels define a subpartition of the set of natural numbers.

The *computation* in Δ is a sequence of two subsequent steps, a computation step and a communication step, which are repeated iteratively and change the configuration of the system.

By a *configuration* of Δ, above, we mean an n-tuple $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$, where $\mathcal{L}_i \in V^*$, $1 \leq i \leq n$.

The *computation* step consists in an iterative application of $\mathcal{R}_i$, $\sigma^*_{\mathcal{R}_i}$, at each node i of G to strings found there.

We say that configuration $(\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$ is obtained from $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$ by a computation step in Δ, denoted by $(\mathcal{L}_1, \ldots, \mathcal{L}_n) \Longrightarrow_{comp} (\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$, if $\mathcal{L}'_i = \sigma^*_{\mathcal{R}_i}(\mathcal{L}_i)$ holds for $1 \leq i \leq n$.

During the *communication* step, the actual contents of the test tubes, *i.e.*, the set of strings found at the nodes, is re-distributed according to the communication graph G and to the labels of the edges, *i.e.*, to the associated predicates.

In order to define this step more formally, we need an auxiliary notion. Let $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$ be a configuration of Δ. We say that $w \in \mathcal{L}_i$ can be communicated from node i to node j in Δ if (i,j) is an edge in G such that

- $p_{i,j}(w) = \underline{true}$ if $p_{i,j} \in \{\pi_{\geq 0}, \pi_{\leq k}, \pi_{\geq k}, \pi_{>k}, \pi_{<k}, \pi_{=k}, \pi_{\neq k} \mid k \geq 1\}$ and
- $p_{i,j}(w, \mathcal{L}_i) = \underline{true}$ if $p_{i,j} \in \{\pi_{\max}, \pi_{\neg\max}, \pi_{\min}, \pi_{\neg\min}\}$.

We say that configuration $(\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$ is obtained from $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$ by a communication step in Δ, denoted by $(\mathcal{L}_1, \ldots, \mathcal{L}_n) \Longrightarrow_{comm} (\mathcal{L}'_1, \ldots, \mathcal{L}'_n)$, if $\mathcal{L}'_i$ consists of all words $w \in V^*$ which satisfy one of the following conditions:

- $w \in \mathcal{L}_j$ and w can be communicated from node j to node i according to $p_{j,i}$,
- $w \in \mathcal{L}_i$ and there is no edge (i,j), $1 \leq i,j \leq n$, in G such that w can be communicated to node j from node i according to $p_{i,j}$.

Thus, after performing the corresponding iterated splicing, the words are communicated to other nodes such that each word is sent to exactly one node. For example, if edge (i,j) is labeled by $\leq k$ (respectively, $\geq k$, $< k$,$> k$,$= k$, $\neq k$), then a word w at node i with $|w| \leq k$ (respectively, $|w| \geq k$, $|w| < k$, $|w| > k$, $|w| = k$, $|w| \neq k$) is sent to node j. Predicate $p_{i,j}(w, \mathcal{L}_i)$ where $p_{i,j} = \max$ (respectively, $p_{i,j} = \neg\max$) makes possible w to be communicated to node j if w is (respectively, is not) a string of maximal length at node i. Similarly, predicate min (respectively $\neg\min$) is for communicating a string w to node j if w is (respectively, is not) a string of smallest length at node i.

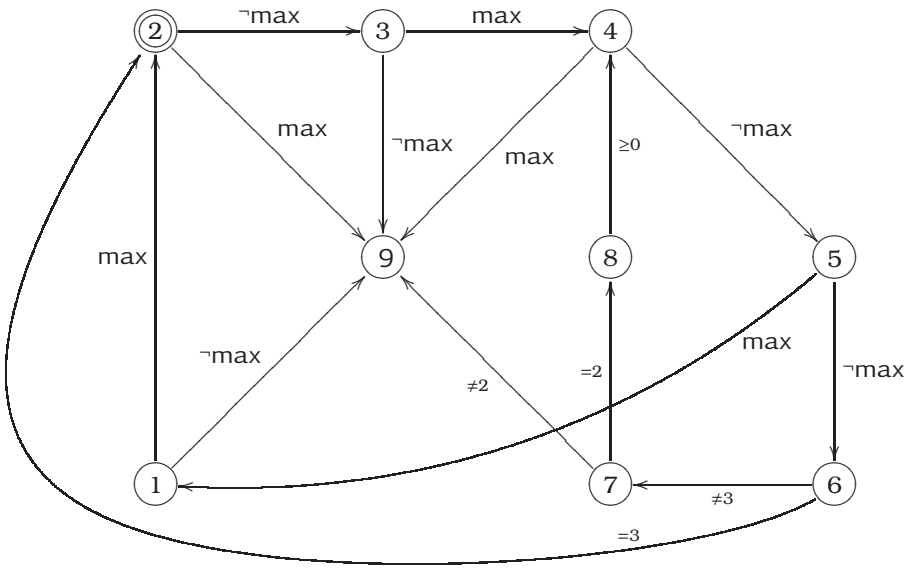
The result of a computation is the set of strings over the terminal alphabet collected at the output tube, i_0, after a communication step in Δ, *i.e.*,

$$L(\Delta) = \{w \in T^* \mid (A_1, \ldots, A_n) \Longrightarrow_{comp} (\mathcal{L}_1^{(1)}, \ldots, \mathcal{L}_n^{(1)}) \Longrightarrow_{comm} (\mathcal{L}_1^{(2)}, \ldots, \mathcal{L}_n^{(2)}) \ldots \Longrightarrow_{comm} (\mathcal{L}_1^{(2s)}, \ldots, \mathcal{L}_n^{(2s)}), \text{ for some } s \geq 0 \text{ such that } w \in \mathcal{L}_{i_0}^{(2s)}\}.$$

We illustrate the above notions by an example.

Example 4.1.1.

Let us define Δ as follows. Consider the following communication graph G and predicates associated to the edges. Notice that node 9 has only incoming edges.



Let $T = \{a, b, c\}$, let the output tube be 2, and let the axioms to be given as $A_1 = \{c'ab, cab'\}$, $A_6 = \{XXZ, ZYY\} \cup \{c'Z, Zb'\}$, with all other axiom sets being empty. Let us define the rule sets associated to the tubes as follows:

$$\mathcal{R}_1 = \left\{ \begin{array}{c|c} a & b' \\ \hline c' & a \end{array} \right\}$$

$$\mathcal{R}_2 = \left\{ \begin{array}{c|c} c & a \\ \hline XX & Z \end{array}, \begin{array}{c|c} a & b \\ \hline Z & YY \end{array} \right\}.$$

$$\mathcal{R}_4 = \left\{ \begin{array}{c|c} XX & a \\ \hline c' & Z \end{array}, \begin{array}{c|c} a & YY \\ \hline Z & b' \end{array} \right\}.$$

All other rule sets are empty, thus no operation is performed in the corresponding tubes.

Starting from $c'ab$ and cab' at tube 1, string $caab$ is obtained which is sent to tube 2 (satisfying communication condition max). At tube 2, it is transformed to strings $XXaab$, $caaYY$ and $XXaaYY$. Then, $XXaaYY$ is sent to tube 9, and the other strings are sent to tube 3. After that, these latter strings arrive at tube 4, since they satisfy condition max. In this latter tube we obtain strings $c'aab$, $caab'$, $XXaab$ and $caaYY$. As before, we send $XXaab$ and $caaYY$ to the "trash" tube (9), and then strings $c'aab$ and $caab'$ via tube 5, arrive at tube 1. At the same time, strings XXZ, ZYY, $c'Z$ and Zb' return to their original location. The above procedure can be repeated. Hence, we obtain the language $\{ca^{2^n}b \mid n > 0\}$, which is a well-known non-context-free context-sensitive language.

4.2 Results on length-separating test tube systems based on splicing

In this section we give the idea how length-separating splicing test tube systems can simulate Turing machines.

We first define a coding function ϕ as follows. For any configuration w_1qw_2 of a Turing machine $M = (Q, T, a_0, q_0, F_M, \delta)$, we define $\phi(w_1qw_2) = X^6w_1qw_2Y^6$, where X and Y are symbols not in T.

Theorem 4.2.1. *Let $M = (Q, T, a_0, s_0, F_M, \delta)$ be a Turing machine and w be an input of M. Then, we can construct a length-separating splicing test tube system $\Delta = (11, V, T', G, A, \mathcal{R}, 10)$ which, given the axiom $\phi(w)$ at a distinguished tube, simulates the behavior of M on input w, i.e., in the following sense:*

1. *For any word w on which M halts in configuration w_1qw_2, Δ produces a unique output $\phi(w_1qw_2)$.*

2. *For any word w, on which M does not halt, Δ computes the empty language.*

The proof of this theorem uses a system having the structure shown on Fig. 4.2.

Informally, system Δ simulates each step of M as follows. Suppose that the current configuration of M is $w_1q_ia_kw_2$ and M contains an instruction (q_i, a_k, R, q_j, a_l). Firstly, the string $X^6w_1q_ia_kw_2Y^6$ (the "main" string) is split, in front of q_i, into two parts: $X^6w_1L_1$ and $R_1q_1a_kw_2Y^6$ which go to node 2. At this node q_ia_k is replaced by q_ja_l and the two parts of the configuration are joined. If necessary, the tape can be extended to the right. The resulting string $X^6w_1q_ja_lw_2Y^6$ goes to node 3 and further to node 1, where a simulation of a new step of the Turing machine may begin. The case of a move to the left is treated analogously. The elimination of the

incorrect strings is done in tubes 1 and 2 using max and ¬max predicates. Indeed, after splitting the "main" string into two parts, the longest string (which is the main string) is discarded by sending it to tube 8 and after that to tube 11. In tube 2, after the replacement and joining, the longest string represents the new configuration and it is preserved. The remaining of the system insures that all additional strings used in rules return to their original places at the appropriate moment of time. The system produces a result only if M halts on the initial string w.

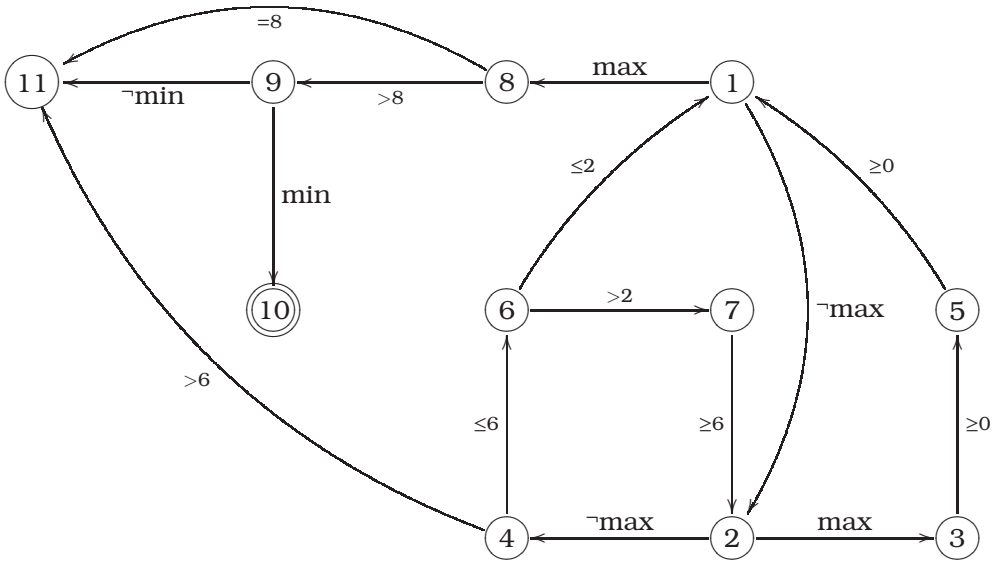


Figure 4.2: Communication graph for the system from Theorem 4.2.1

The construction used in this theorem is similar to certain constructions given in [52] and [72]. Using max and ¬max predicates it is possible to filter out strings which do not correspond to a correct simulation of the Turing machine. Hence, only strings representing (through encoding) configurations of M will be kept.

4.3 Restricted Variants

In this section we consider some variants of the model by imposing restrictions on the type of predicates. The first result shows that systems that use only $\pi_{\max}$, $\pi_{\min}$ and $\pi_{\geq 0}$ predicates do not have a decrease of their computational power:

Theorem 4.3.1. *It is possible to transform the length-separating splicing test tube system* $\Delta = (11, V, T', G, A, \mathcal{R}, 10)$ *constructed as in Theorem 4.2.1 into a length-separating splicing test tube system* $\Delta' = (13, V, T', G', A, \mathcal{R}, 10)$ *having communication graph* G' *labeled only by predicates from the set* $\{\pi_{\geq 0}, \pi_{\max}, \pi_{\neg\max}, \pi_{\min}, \pi_{\neg\min}\}$ *such that* $L(\Delta) = L(\Delta')$.

The construction given in Theorem 4.2.1 can be modified by taking into account that a small number of strings transit tubes 2 and 4. In this case it is possible to replace a number predicate by one or several consecutive min predicates.

The computational power of systems not using $\pi_{\max}$ or $\pi_{\min}$ predicates is an open problem. We conjecture that these systems are not very powerful:

Conjecture 4.3.2. The language generated by any length-separating splicing test tube system Δ having communication graph labeled only by predicates from the set $\Pi = \{\pi_{\geq 0}, \pi_{\leq k}, \pi_{\geq k}, \pi_{>k}, \pi_{<k}, \pi_{=k}, \pi_{\neq k} \mid k \geq 1\}$ is regular.

Our conjecture is supported by the following observation. It is easy to see that for any $k \geq 0$, $\mathcal{L}_i^{(k)}, i \in O$ is a regular language. Indeed, for any $k, i \geq 0$ if language $\mathcal{L}_i^{(k)}$ is obtained by a computational step, then it is regular because the splicing operation preserves the regularity of a language [61]. If $\mathcal{L}_i^{(k)}$ is obtained by a communication step, then it is also regular because any predicate performs a regular partition of a language. Hence, each $\mathcal{L}_i^{(k)}$ is the difference of the union of the regular languages corresponding to the incoming edges of node i and the union of the regular languages corresponding to the outgoing edges of node i. Since the finite union and the finite difference of regular languages are regular languages, language $\mathcal{L}_i^{(k)}$ is regular.

Moreover, we can observe that for any finite combination of predicates from the set Π, there is a constant $l > 0$ such that all words having length greater than l cannot be distinguished by such a combination. This assertion does not hold in the case of the $\pi_{\max}$ or the $\pi_{\min}$ predicate. By these observations, we guess that using separation by fixed length, after some steps C there will be the following equality: $\mathcal{L}_i^{(k)} = \mathcal{L}_i^{(k+h)}$ for $1 \leq i \leq n$, $k > C$ and $h > 0$.

4.4 Discussion

The reader can observe that, in fact, the main ingredient of the presented model is the *communication* controlled by the length of the words. Thus, the idea of a length separating test tube system can also be given in a *general form*, *i.e.*, using an arbitrary operation, γ, defined on words (or sets of words.) Moreover, we may associate different operations to different test tubes, thus obtaining a *hybrid system*. In this way, we need only to change the definition of the computing step. The study and the comparison of length-separating *test tube systems based on different operations* would be of interest.

It would also be interesting to investigate systems having different restrictions, in particular, systems where only *one elementary rule per test tube* is allowed. In the splicing case, this would mean that only *one restriction enzyme* is used. We think that this restriction may better reflect the reality.

These systems can also be considered as *transducers*, which transform a set of strings into another set. In this case, several systems may be chained one after another and their actions can be combined. From a practical point of view, this also means that $V = T$, *i.e.*, because in this case no final filtering of the strings (terminal strings) is done. Notice, that the distinction of the terminal alphabet is traditionally used only for selecting some particular strings among the results of the computation and it is not an inherent property of the construction.

Another important point for the definition of the model, especially if it is used as a transducer, is the detection of a *halting* configuration. In most of the constructions from DNA computing the end of the computation is defined by halting. Usually, this condition means that the system cannot evolve, *i.e.*, there is no applicable rule in the system. However, from practical point of view, it is difficult to detect the fulfillment of this condition. Therefore, we propose to use a special

acknowledgment test tube. When anything arrives in this tube, the computation is considered to be finished and the result is read in the output test tube. It is also possible to combine the acknowledgment and the output test tubes. In this case the computation stops when a first molecule arrives in that tube.

Another idea is to permit that the strings satisfy more than one predicates belonging to outgoing edges of the same node. According to this model, if a string validates several predicates, then a copy of it is sent to all the corresponding tubes. In the simplest case, when two connections with $\pi_{\geq 0}$ predicate are present, this corresponds to the duplication of the contents of the test tube.

We would like to remark that the length-driven communication principle of our model may be also generalized by substituting the length parameter by some other parameter depending on the word. In this case, in definitions from Section 4.1 the length function $|w|$ shall be replaced by an arbitrary function $f(w)$, $f : V^* \to \mathbb{N}$. This approach is very general, and several existing models, e.g. splicing test tube systems [15], can be expressed in these terms.

The formalization that we provided can be further used to investigate different properties of models based on gel electrophoresis. Moreover, since some ideas from the previous section may be realized in practice, it can be used for engineering transducers (protocols) that will do some particular transformation on DNA molecules in laboratory. However, in this case, size exclusion chromatography becomes more interesting as it permits to separate molecules much faster. We recall that size exclusion chromatography uses a column filled with a porous material and the molecules to be separated arrive at the top of the column and under the pressure move towards the bottom. Bigger molecules go directly to the other end, while smaller molecules enter the pores of the material and take more time to arrive to the end, see Fig. 4.3. By doing a good time calibration it is possible to accurately separate molecules of different length in very short period of time.

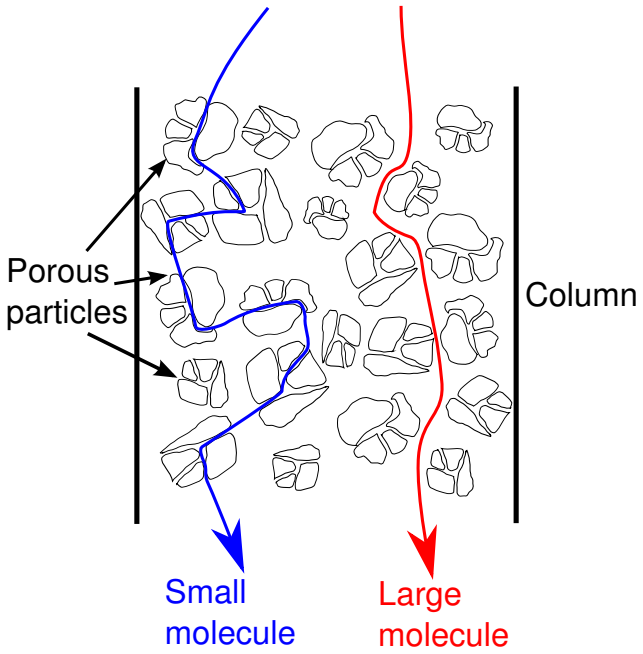


Figure 4.3: The principle of size exclusion chromatography

4.5 Bibliographical remarks

The idea of the length separation was used from the beginning of DNA computing. For example, it is used in the third step of the Adleman's experiment [1]. There are also algorithms like in [32] based on the length separation which use it at the end of the computation in order to confirm or select the result. In [41], a method called length-only discrimination based on the generate-and-search approach but relying on the length of the sequence is presented and experimentally confirmed. However, in none of these places the length separation is considered as the main ingredient of the model, but rather a subsidiary operation.

Chapter 5

Directions for further research

This chapter summarizes the most interesting open questions raised by the research presented in this thesis. More suggestions can be found in the text of previous chapters as well as in the proposed references.

Insertion-deletion

Our study of insertion-deletion systems raised new open problems, in particular, related to the computational completeness of one-sided insertion-deletion systems with different size parameters. We give below a list of systems not investigated so far. In order to simplify the presentation we consider systems of size $(i, j, k; 1, 1, 0)$, $i \in \mathbb{N}^+, j, k \in \mathbb{N}$ *i.e.*, having one-sided minimal deletion rules. The results from tables 2.1 and 2.2 suggest that probably systems having $i + j + k = 4$ are computationally complete, while systems having $i + j + k = 3$ are not. However, this conjecture needs a further verification. In a similar way, we conjecture that systems of size $(i, j, k; 2, 0, 0)$, *i.e.*, having context-free deletion of two symbols are computationally complete if $i + j + k = 3$ and not if $i + j + k = 2$. Since our results are symmetric with respect to the size of the insertion and deletion, we conjecture similar results for systems of size $(1, 1, 0; i, j, k)$ and $(2, 0, 0; i, j, k)$.

Another interesting topic is the computational power of same-side one-sided insertion-deletion systems, for example, of size $(m, n, 0; 1, 1, 0)$ or $(1, 1, 0; m, n, 0)$, $m > 0$, $n \geq 0$. Our preliminary investigations on this type of systems show that the ideas used for previous constructions cannot be applied in this case. Using different type of arguments it is possible to show that systems of size $(1, 1, 0; 3, 2, 0)$ strictly include the family of regular languages, however their computational power is still an open question.

The verification of above conjectures can be tedious and we think that in this case the method of the direct simulation can be extremely useful.

For the graph-controlled case a special attention can be further drawn to the number of used components. Our computational completeness results make use of 5 components, we suggest that this number can be reduced to 4 or 3 components. In the case of systems with priorities it would be interesting to find a small bound on the number of used components.

As in the case of graph-controlled insertion-deletion systems it would be interesting to adapt the concept of matrix, ordered and random-context grammars to insertion-deletion systems. We think that as in the graph-controlled case, the com-

putational power should strictly increase. Our preliminary investigations showed that this is the case for the matrix insertion-deletion systems.

The unusual transformations of strings by insertion and deletion operations can be further explored for the construction of random number generators or in the area of genetic algorithms.

P systems

We give below some interesting research directions suggested by our research in the area of P systems.

The first direction we suggest concerns the study of the generalized communicating P systems. In our opinion a particular attention shall be drawn to the modularity of the concept. The given proofs and examples feature a set of primitive building blocks which serve as a basis for the final construction. We think that it is a very promising topic and that the research of more complex building blocks and of ways of their assembly shall be continued. Such investigation will necessarily involve concepts similar to the ones used in Petri Nets or Process Algebra and will permit to nicely bridge these areas and P systems.

Another direction that can be explored is the fact that generalized communicating P systems can be seen as a computational model based on the synchronization of signals (represented by objects) in a network. In the minimal case only two signals synchronize, this can be extended to a finite number. Such a viewpoint can permit to model different properties of networks, for example, the computational or informational flow.

The investigation of the semantics of P systems is a long-term project that can permit to efficiently classify different variants of P systems and establish equivalences between them. Our approach gives a very deep insight on the functioning of various models, leading to their better understanding, giving the possibility to express complicated behaviors in simpler terms. One of the promising directions for further research is to consider non-disjoint partitions for derivation modes involving partitions of rules. This would imply that one rule can be member of several partitions, hence its selection can permit to fulfill derivation mode requirements for several partitions at the same time. Such an approach allows all possibilities for a rule application thus giving a full control on it.

It is also very important to give a similar formal definition for P systems having a dynamic structure, *i.e.*, where the number of membranes can evolve at each computational step. This will permit to complete the picture and to start deeper investigations on corresponding classes.

The study of small-size universal P systems is important from the philosophical point of view. The quest for small universal devices gives us the possibility to explore the borderline separating complicated and simple behaviors. It is amazing to find out that simple, limited machines or processes can behave in the most complex non-predictable way. We suggest reading the book [80] that explores such kind of ideas. We remark that the universality results we obtained cannot be directly compared to descriptional complexity results for Turing machines [63] or register machines [45], but it seems that the mechanisms offered by the framework of P systems allow a decrease of the number of rules.

Another important point that needs to be highlighted is the fact the antiport

P systems are universal for the family of recursively enumerable sets of numbers. In contrast to the string case, the only investigation about the descriptional complexity for this family was done using register machines, see [45]. Our approach permits to complete these investigations and opens the possibility to improve the aforementioned results on register machines, because our system has 23 rules, while the universal register machine U_{32} has 25 branching points.

Finally, we think that, in the future, one of the main types of investigations in the area of P systems will be the implementation and adaptation of different known algorithms. This can give a different viewpoint on corresponding algorithms and why not a more efficient implementation.

Length-separation model

By summarizing the discussion from Section 4.4 we would like to highlight several directions for the future research. In our opinion the idea of hybrid systems shall be explored further, in particular, using tube operations different from splicing. We suggest, for example, the use of the operations of cut and recombination [28], rewriting, or insertion-deletion.

The second fruitful idea is to use tubes dedicated to one operation. This permits to bridge our investigations with real life and gives the possibility to plan experiments *in vitro*.

The third research direction concerns the use of the length-separation idea in the theory of formal languages. It would be interesting to investigate different computational models where the control set conditions are replaced by the length separation. A generalization of the concept to arbitrary functions is also very promising. For example, the length function can be replaced by the modulo function of the string length, which can eventually permit not to use max and min predicate. This generalization could also give a general framework for existing models and the length-separation variants.

Conclusions

This thesis gives a summary of two of our main research activities during the last 5 years. The study of insertion-deletion systems is a typical study in the formal language theory. Since insertion and deletion rules can be seen as restrictions of rewriting rules, our results are relevant for the theory of formal languages, as they propose new classes of grammars having (very) restricted types or rules. Beside the fact that some of these classes are computationally complete, it is probably much more interesting to find out that there are classes which are not. This introduces additional levels in the Chomsky hierarchy and can further provide new ideas for grammar and automata construction.

Our study of P systems is less unitary, since the underlying topic is very vast. In this thesis we have chosen the subjects that, in our opinion, merit more attention. We are convinced that the formal definition of P systems is an important achievement for the area that would further permit to rigorously define and compare new models. Despite of its simplicity, the obtained result took about 2 years in order to be formulated in a convenient way. The generalized communicating model is the fruit of our observations on the minimal symport/antiport; its clear intuitive definition, nice building properties and easy correspondence to Petri Nets make it attractive for further research and applications. Finally, the study of universal P systems is a kind of investigation that should be done by a researcher working in the area of theoretical computer science. This gives an important understanding of the fundamental concepts of the computability theory and permits to acquire a strong intuition for this area, not taking into account interesting philosophic questions raised by such kind of research.

Beside the above two activities, which count for most of our research efforts during these years, we chose to present the length-separation model, which, in our opinion, is extremely interesting. Two features of this model especially captured our attention. The first one is that despite of its simplicity, surprisingly, the length-separation mechanism was not considered in the theory of formal languages. The second feature, confirmed by the discussions with specialists, is that the model is not too abstract and can be implemented in laboratory, *in vitro*. This exciting property is a strong argument for the further investigation of the model.

The preparation of this thesis gave a possibility to look back at the performed research and to highlight the most interesting and important achievements. What looks simple now, was not so trivial some time before. We can enjoy the results only if we see the whole road leading to them. This is probably the most important conclusion suggested by the writing of this thesis.

Bibliography

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, **266**, (1994), 1021–1024.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell, Fourth Edition*. Garland, 2002.

[3] A. Alhazov, R. Freund, M. Oswald, and S. Verlan. Partial halting in P systems using membrane rules with permitting contexts. In J. O. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*. Springer, 2007, volume 4664 of *Lecture Notes in Computer Science*, 110–121.

[4] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. P systems with minimal insertion and deletion. In R. Gutiérrez-Escudero, M. A. Gutiérrez-Naranjo, G. Păun, I. Pérez-Hurtado, and A. Riscos-Nunez, editors, *Proc. of Seventh Brainstorming Week on Membrane Computing Sevilla, February 2-6, 2009*. Fénix Editora, Sevilla, 2009, volume I, 9–21. Also accepted to *Theoretical Computer Science*.

[5] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. *New Trends in Formal Language Theory Inspired by Natural Computing: Small Size Insertion and Deletion Systems*, Imperial College Press, chapter 1. Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory. 2010, 459–524. In publication.

[6] A. Alhazov, M. Margenstern, and S. Verlan. Fast synchronization in P systems. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing - 9th International Workshop, WMC 2008, Edinburgh, UK, July 28-31, 2008, Revised Selected and Invited Papers*. Springer, 2008, volume 5391 of *Lecture Notes in Computer Science*, 118–128.

[7] A. Alhazov, Y. Rogozhin, and S. Verlan. Minimal cooperation in symport/antiport tissue P systems. *International Journal of Foundations of Computer Science*, **18**(1), (2007), 163–180.

[8] A. Alhazov and S. Verlan. Minimization strategies for maximally parallel multiset rewriting systems. Technical Report 862, TUCS, 2008. Also accepted to *Theoretical Computer Science*.

[9] R. Benne. *RNA-Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood, Chichester, West Sussex, 1993.

[10] F. Bernardini, M. Gheorghe, M. Margenstern, and S. Verlan. How to synchronize the activity of all components of a P system? *International Journal of Foundations of Computer Science.*, **19**(5), (2008), 1183–1198.

[11] G. Ciobanu, L. Pan, G. Păun, and M. J. Pérez-Jiménez. P systems with minimal parallelism. *Theoretical Computer Science*, **378**(1), (2007), 117 – 130.

[12] G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.

[13] J. Cocke and M. Minsky. Universality of tag systems with P=2. *Journal of the ACM*, **11**(1), (1964), 15–20.

[14] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Elektronische Informationsverarbeitung und Kybernetik*, **26**(1/2), (1990), 49–63.

[15] E. Csuhaj-Varjú, L. Kari, and G. Păun. Test tube distributed systems based on splicing. *Computers and Artificial Intelligence*, **15**(2–3), (1996), 211–232.

[16] E. Csuhaj-Varjú, M. Margenstern, G. Vaszil, and S. Verlan. On small universal antiport P systems. *Theoretical Computer Science*, **372**(2-3), (2007), 152–164.

[17] E. Csuhaj-Varjú and A. Salomaa. Networks of parallel language processors. In G. Păun and A. Salomaa, editors, *New Trends in Formal Languages*. Springer, 1997, volume 1218 of *Lecture Notes in Computer Science*, 299–318.

[18] E. Csuhaj-Varjú and S. Verlan. On length-separating test tube systems. *Natural Computing*, **7**(2), (2008), 167–181.

[19] E. Csuhaj-Varjú and S. Verlan. On generalized communicating P systems with minimal interaction rules. *Theoretical Computer Science*. 2010. In press.

[20] M. Daley, L. Kari, G. Gloor, and R. Siromoney. Circular contextual insertions/deletions with applications to biomolecular computation. In *SPIRE/CRIWG*. 1999, 47–54.

[21] M. Domaratzki and A. Okhotin. Representing recursively enumerable languages by iterated deletion. *Theoretical Computer Science*, **314**(3), (2004), 451–457.

[22] R. Freund, A. Alhazov, Y. Rogozhin, and S. Verlan. Communication P systems. In Păun et al. [62], 118–143.

[23] R. Freund, O. Ibarra, G. Păun, and H.-C. Yen. Matrix languages, register machines, vector addition systems. In M. Gutiérrez-Naranjo, A. Riscos-Núñez, F. R. Campero, and D. Sburlan, editors, *Proceedings of the Third Brainstorming Week on Membrane Computing*. University of Sevilla, 2005, 155–168.

[24] R. Freund, M. Kogler, and S. Verlan. P automata with controlled use of minimal communication rules. In H. Bordihn, R. Freund, M. Holzer, M. Kutrib, and F. Otto, editors, *Workshop on Non-Classical Models of Automata and Applications, NCMA 2009, Wroclaw, Poland*. Oesterreichische Computer Gesellschaft, 2009, 107–120.

[25] R. Freund and A. Păun. Membrane systems with symport/antiport rules: Universality results. In G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers*. Springer, 2002, volume 2597 of *Lecture Notes in Computer Science*, 270–287.

[26] R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*. Springer, 2007, volume 4860 of *Lecture Notes in Computer Science*, 271–284.

[27] R. Freund and S. Verlan. P systems working in the k-restricted minimally parallel mode. In E. Csuhaj-Varjú, R. Freund, M. Oswald, and K. Salomaa, editors, *International Workshop on Computing with Biomolecules, August 27th, 2008, Wien, Austria*. Oesterreichische Computer Gesellschaft, 2008, volume 244, 43–52.

[28] R. Freund and F. Wachtler. Universal systems with operations related to splicing. *Computers and Artificial Intelligence*, **15**(4), (1996), 273–294.

[29] P. Frisco and H. J. Hoogeboom. Simulating counter automata by P systems with symport/antiport. In G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers*. Springer, 2002, volume 2597 of *Lecture Notes in Computer Science*, 288–301.

[30] B. Galiukschov. Semicontextual grammars. *Matem. Logica i Matem. Lingvistika*, (1981), 38–50. Tallin University, (in russian).

[31] E. Goto. *A Minimum Time Solution of the Firing Squad Problem*, volume 298 of *Course Notes for Applied Mathematics*. Harvard University, 1962.

[32] F. Guarnieri, M. Fliss, and C. Bacroft. Making DNA add. *Science*, **273**(12), (1996), 220–223.

[33] D. Haussler. *Insertion and Iterated Insertion as Operations on Formal Languages*. Ph.D. thesis, Univ. of Colorado at Boulder, 1982.

[34] D. Haussler. Insertion languages. *Information Sciences*, **31**(1), (1983), 77–89.

[35] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, **49**(6), (1987), 737–759.

[36] T. Head. Splicing languages generated with one sided context. In *Computing with Bio-Molecules. Theory and Experiments*. 1998, 158–181.

[37] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 2nd edition, 2001.

[38] L. Kari. *On Insertion and Deletion in Formal Languages*. Ph.D. thesis, University of Turku, 1991.

[39] L. Kari, G. Păun, G. Thierrin, and S. Yu. At the crossroads of dna computing and formal languages: Characterizing RE using insertion-deletion systems. In *Proc. of 3rd DIMACS Workshop on DNA Based Computing*. Philadelphia, 1997, 318–333.

[40] L. Kari and G. Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, **131**(1), (1996), 47–61.

[41] Y. Khodor, J. Khodor, and T. F. K. Jr. Experimental conformation of the basic principles of length-only discrimination. In N. Jonoska and N. C. Seeman, editors, *DNA Computing, 7th International Workshop on DNA-Based Computers, DNA7, Tampa, Florida, USA, June 10-13, 2001, Revised Papers*. Springer, 2001, volume 2340 of *Lecture Notes in Computer Science*, 223–230.

[42] H. Kitano. A graphical notation for biochemical networks. *Biosilico*, **1**, (2003), 169–176.

[43] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, Princeton University Press, Princeton, NJ. 1956, 3–41.

[44] K. W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, **10**, (1999), 2703–2734.

[45] I. Korec. Small universal register machines. *Theoretical Computer Science*, **168**(2), (1996), 267–301.

[46] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of insertion-deletion (P) systems with rules of size two. Accepted to *Natural Computing*. 2010, in publication.

[47] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Further results on insertion-deletion systems with one-sided contexts. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*. Springer, 2008, volume 5196 of *Lecture Notes in Computer Science*, 333–344.

[48] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. One-sided insertion and deletion: Traditional and P systems case. In E. Csuhaj-Varjú, R. Freund, M. Oswald, and K. Salomaa, editors, *International Workshop on Computing with Biomolecules, August 27th, 2008, Wien, Austria*. Druckerei Riegelnik, 2008, 53–64.

[49] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of P systems with small size insertion and deletion rules. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008*. 2009, volume 1 of *EPTCS*, 108–117.

[50] S. Marcus. Contextual grammars. *Revue Roumaine de Mathématique Pures et Appliquées*, **14**, (1969), 1525–1534.

[51] M. Margenstern, G. Păun, Y. Rogozhin, and S. Verlan. Context-free insertion-deletion systems. *Theoretical Computer Science*, **330**(2), (2005), 339–348.

[52] M. Margenstern and Y. Rogozhin. A universal time-varying distributed H system of degree 2. *Biosystems*, **52**, (1999), 73–80.

[53] C. Martín-Vide, G. Păun, and A. Salomaa. Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science*, **205**(1-2), (1998), 195–205.

[54] A. Matveevici, Y. Rogozhin, and S. Verlan. Insertion-deletion systems with one-sided contexts. In J. O. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*. Springer, 2007, volume 4664 of *Lecture Notes in Computer Science*, 205–217.

[55] J. Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, **50**, (1987), 183–238.

[56] M. Minsky. *Computations: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffts, NJ, 1967.

[57] E. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, **65**(2), (1943), 197–215.

[58] G. Păun. Regular extended H systems are computationally universal. *Journal of Automata, Languages and Combinatorics*, **1**(1), (1996), 27–36.

[59] G. Păun. *Marcus Contextual Grammars*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[60] G. Păun. *Membrane Computing. An Introduction*. Springer–Verlag, 2002.

[61] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms*. Springer, 1998.

[62] G. Păun, G. Rozenberg, and A. Salomaa. *The Oxford Handbook Of Membrane Computing*. Oxford University Press, 2009.

[63] Y. Rogozhin. Small universal turing machines. *Theoretical Computer Science*, **168**(2), (1996), 215–240.

[64] Y. Rogozhin and S. Verlan. On the rule complexity of universal tissue P systems. In R. Freund, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*. Springer, 2005, volume 3850 of *Lecture Notes in Computer Science*, 356–362.

[65] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages, 3 volumes*. Springer Verlag, Berlin, Heidelberg, New York, 1997.

[66] H. Schmid and T. Worsch. The firing squad synchronization problem with many generals for one-dimensional ca. In J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics, IFIP 18th*

World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France. Kluwer, 2004, 111–124.

[67] W. D. Smith. DNA computers in vitro and in vivo. In R. Lipton and E. Baum, editors, *Proceedings of DIMACS Workshop on DNA Based Computers*. Amer. Math. Society, 1996, DIMACS Series in Discrete Math. and Theoretical Computer Science, 121–185.

[68] P. Spellman and G. Sherlock. Reply whole-cell synchronization – effective tools for cell cycle studies. *Trends in Biotechnology*, **22**(6), (2004), 270–273.

[69] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. In M. Hagiya and A. Ohuchi, editors, *DNA Computing, 8th International Workshop on DNA Based Computers, DNA8, Sapporo, Japan, June 10-13, 2002, Revised Papers*. 2002, volume 2568 of *Lecture Notes in Computer Science*, 269–280.

[70] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. *Natural Computing*, **2**(4), (2003), 321–336.

[71] H. Umeo, M. Maeda, and N. Fujiwara. An efficient mapping scheme for embedding any one-dimensional firing squad synchronization algorithm onto two-dimensional arrays. In S. Bandini, B. Chopard, and M. Tomassini, editors, *Cellular Automata, 5th International Conference on Cellular Automata for Research and Industry, ACRI 2002, Geneva, Switzerland, October 9-11, 2002, Proceedings*. Springer, 2002, volume 2493 of *Lecture Notes in Computer Science*, 69–81.

[72] S. Verlan. Communicating distributed H systems with alternating filters. In N. Jonoska, G. Păun, and G. Rozenberg, editors, *Aspects of Molecular Computing. Essays Dedicated to Tom Head on the Occasion of His 70th Birthday*, Springer, volume 2950 of *Lecture Notes in Computer Science*. 2004, 367–384.

[73] S. Verlan. *Head Systems and Applications to Bioinformatics*. Ph.D. thesis, University of Metz, 2004.

[74] S. Verlan. On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, **12**(1-2), (2007), 317–328.

[75] S. Verlan. Look-ahead evolution for P systems. In G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24-27, 2009. Revised Selected and Invited Papers*. Springer, 2009, volume 5957 of *Lecture Notes in Computer Science*, 479–485.

[76] S. Verlan, F. Bernardini, M. Gheorghe, and M. Margenstern. Computational completeness of tissue P systems with conditional uniport. In H. J. Hoogeboom, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 7th International Workshop, WMC 2006, Leiden, The Netherlands, July 17-21, 2006, Revised, Selected, and Invited Papers*. Springer, 2006, volume 4361 of *Lecture Notes in Computer Science*, 521–535.

[77] S. Verlan, F. Bernardini, M. Gheorghe, and M. Margenstern. Generalized communicating P systems. *Theoretical Computer Science*, **404**(1-2), (2008), 170–184.

[78] S. Verlan and P. Frisco. Splicing P systems. In Păun et al. [62], 198–226.

[79] S. Verlan and Y. Rogozhin. New choice for small universal devices: Symport/antiport P systems. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008*. 2008, volume 1 of *EPTCS*, 235–241.

[80] S. Wolfram. *A New Kind of Science*. Wolfram Media Inc, 2002.

[81] J.-B. Yunès. Seven-state solutions to the firing squad synchronization problem. *Theoretical Computer Science*, **127**(2), (1994), 313–332.

[82] The P systems web page: `http://ppage.psystems.eu/`.