École doctorale IAEM Lorraine **Université de Metz**
Département de Formation Doctorale en Informatique **UFR MIM**

Head systems and applications to bio-informatics

THESIS

presented and publicly defended 21 June 2004

for the obtention of degree of

Doctor in Philosophy of University of Metz

speciality Computer Science

by

Serghei Verlan

Laboratoire d'Informatique Théorique et Appliquée

Acknowledgments

First of all I would like to thank the supervisor of my thesis, Maurice Margenstern, whose scientific experience guided me a lot in my work. I thank him equally for his patience with me and for his help during these last years.

I thank a lot my professor from Moldova, Yurii Rogozhin, who initiated me to the domain of molecular computing. Our discussions during his stays in Metz contributed in a direct or indirect way to results of this work, and I appreciate a lot his support.

I would also like to thank my co-authors: Rudolf Freund, Franziska Freund, Marion Oswald, Gheorghe Păun and Rosalba Zizza for original ideas that I would maybe never find without their help.

I would like to address a particular acknowledgment to Gheorghe Păun who often inspired me with numerous ideas.

I also thank all persons with whom I spend wonderful moments during conferences and who often suggested me interesting ideas, in particular Daniela Besozzi, Claudio Feretti, Pierluigi Frisco, and Claudio Zandron.

I acknowledge the NATO project PST.CLG.976912 and the project IST-2001-32008 “MolCoNet” that permitted me to participate conferences, present my works, and exchange ideas and experience. I also thank Ministry of National Education and Research of France for the financial support of my PhD as well as the “Laboratoire d'Informatique Théorique et Appliquée” of the Metz University that accommodated me, and in particular its secretary Damien Aignel who helped me often in administrative tasks.

I would also like to thank all my friends in Metz that permitted me to feel here like home.

Finally, I thank my parents who always believed in me and who supported me during these years.

Contents

Introduction

Everything is number. By taking this ancient pythagorean idea and by combining it with the fact that a number is a result of computation, we can reformulate the above sentence and say that everything is computation. We can find examples of computations in numerous domains, but the most fascinating example is the life. It is not easy to give an exact definition of this complex phenomena, but at the moment it is supposed that the most important ingredient of the life is the desoxyribonucleic acid, the DNA, which is sometimes replaced by the ribonucleic acid, or RNA. In fact, all living beings which we know have this molecule and the nature uses simple operations of copying, pasting, insertion, deletion etc. in order to manipulate it. We see that we need only small bricks, the DNA, and simple rules in order to describe certain structural aspects of living beings and maybe even the diversity of the nature. We can remark a parallel with the theory of computability, where we also use very simple elements in order to construct functions which perform complex computations. This analogy leads us to the idea that the nature computes, but maybe in a different way that we do not understand completely yet. From chaos to inorganic matter, from inorganic to organic, from unconsciousness to intelligence, perhaps the entire evolution of the universe is a history of the ever-increasing complexity of the computation. All this may appear as a metaphysical speculation, but, who knows, maybe our actual conception of computation is dependent on the evolution of the humankind, like the utilisation of base ten for counting is depending on our having of ten fingers. In the same way humans moved on to counting in other bases, maybe it is time to find new conceptions of a computation which will be different from the ones we used before.

A first step was made by L. Adleman in 1994 who showed for the first time that it is possible to use these new operations and data structures in order to solve concrete problems. His experiment shows how to solve an instance of a problem of a hamiltonian path in a graph. It is amazing, but by using DNA molecules, enzymes and other chemicals and by manipulating test tubes, it is possible to find a solution to this concrete mathematical problem. The Adleman's experiment, published in "Science", see [1], stimulated the study of new models of computing based on biology. Various aspects of biological mechanisms were studied from a computational point of view and the results are very promising, because the obtained models permit, of cause from a theoretical point of view, to overcome ordinary computers. Compact data structures, massive parallelism and modest energy consumption are the advantages which affirm the superiority, at least theoretical, of computers based on biological principles.

We remark that a theoretical study of biological systems from a computational point of view started long before 1994. In fact, already in 1987, T. Head in his article "Formal Language Theory and DNA: an Analysis of the Generative Capacity of Specific Recombinant Behaviors", [12], showed connections between the molecular biology and the theory of formal languages. His pioneer vision of the molecular recombination permitted him to observe long time before Adleman the importance and the enormous potential of biologically based computations. The systems that he introduced and which we call now Head systems, based on the splicing operation, represent even now an important branch of the domain. The idea of Head was very simple: to replace DNA molecules by words and enzymes by splicing rules. After that, everything is mixed in a test tube and the system is let to evolve. But, despite his work, we can say that the birth of the domain of (bio)-molecular computing dates to the Adleman experiment, which showed that the things considered before as fiction become from now onwards realisable.

What are the aims of this domain? The most important goal is of course to construct a computer based on biological principles. But for the moment, it is still difficult to conjecture when we will work on such computers. Another very important goal is to understand how the nature computes and, maybe, to find new computational paradigms which could be implemented even on ordinary supports. Like the study of brain permitted to create neural networks and the study of evolution emerged to creation of genetic algorithms, we hope to learn from the nature new methods of computation which we shall be able to use in the future.

DNA is not the only source of inspiration in finding new computational paradigms. We can also place ourselves at the cellular level. In fact, the living cell is an astonishing piece of machinery which the nature perfects all the time. Everything is present here: communication with the environment, signal recognition, search in the DNA database, reaction, production and consumption of energy... Shortly, the understanding of cell functionality is perhaps the key that will permit us to understand secrets of life. But if we can imagine computations with DNA, we can also imagine computations with cells. In this case, the structural aspect become more and more important, as the cell may be seen as a set of membranes nested one in another. The first model of computing based on the cell structure and on cellular processes was proposed in 1998 by Gh. Păun who is considered the father of this domain: *membrane computing*. From then on, more than a hundred of variants were proposed and their number increase permanently. The current level of the technology does not permit its implementation *in vivo* or *in vitro*, but, as we said before, one of the reasons for which we study the models issued from the biology is to find new possibilities of computation different from these we have used till now.

From one hand, the theory of formal languages is grounded on rewriting operations. From the other hand, the nature uses different operations like copy and paste as well as different data structures. This is why it is very important to reconstruct old computational paradigms in this new framework. Whether or not this has practical significance, for the computer science applications this is a premature question.

We shall stop here for further metaphysical speculations and refer to the article [16] which contains an excellent introduction to the domain, as well as a lot of philosophical reflections that we presented here only in a small part. Further, the books [44] and [43] contain a very detailed presentation of motivations and expectations concerning the area of molecular computing, membrane computing and natural computing in general.

The present thesis tries to answer certain questions above. We will concentrate on the study of certain models of molecular computing and membrane computing in order to show their equivalence with classical models like formal grammars and Turing machines. We do even a step further by showing directly, without going by classical systems, the equivalence of some considered models. Finally, we conceived a method which permits to simplify the proof of these equivalences as well as construction of specific systems. We remark that all models that we consider have a common point: they are based on the splicing operation. Therefore, we can say that we make a detailed study of this operation which is indeed very powerful by itself and, as it is shown in numerous examples in this work, only small additions are necessary to make it as powerful as the rewriting.

We also remark that our work is purely theoretical and that we consider mathematical aspects of molecular computing but not the biological one.

Now we will describe the contents of the thesis chapter by chapter.

Chapter 1

We shortly present in this chapter some definitions from the theory of formal languages and fix the notations for further use.

Chapter 2

We introduce in informal manner the systems that we investigate in this work and we review the state of the art.

Chapter 3

We introduce in this chapter the central notions of our thesis: the *splicing operation* and *Head systems*, or H systems. We consider two possible definitions of the splicing: 1-splicing and 2-splicing, and we study for the first time the relation between classes of languages based on 1-splicing and on 2-splicing. We show that between these two families there is a relation of a strict inclusion. Also, in this chapter we present numerous examples of H systems as well as of regular languages which cannot be splicing languages.

Chapter 4

We introduce in this chapter an extension of H systems: *extended H systems* and we present the "rotate-and-simulate" method which is often used in order to show an equivalence with a formal grammar. We also introduce *time-varying distributed H systems*, or TVDH systems, which is a simple and at the same time powerful model. We remark that structure of these systems is quite simple to permit their numerous utilisations, and we frequently used them afterwards. We show that it is possible to generate all recursively enumerable languages by TVDH systems with with two components, and that it is possible to simulate the behaviour of tag systems with one component. We remark that the last system has a very compact description. We present in the same chapter results obtained with the help of the computer program `TVDHsim` which permits to simulate TVDH systems. In fact, this program permitted to find errors in certain articles on TVDH systems, and we present corresponding systems as well as modifications necessary to give them a correct behaviour in the cases where it was possible. We also add that the same program helped us to acquire a big experience in construction of systems based on splicing by highlighting the errors which are common to several authors. Finally, we remark that all TVDH systems from this work were checked with the help of this program.

Chapter 5

The chapter 5 introduce an extension of TVDH systems, *enhanced time-varying distributed H systems* which have a more complex behaviour. We show that three components are enough in order to generate all recursively enumerable languages.

Chapter 6

The chapter 6 contain an analysis of the proof technique used in previous two chapters. This analysis emerges with a new method, the *method of directing molecules*, which permits to decrease surprisingly the complexity of corresponding systems by showing that they have an additional control which we put in evidence. By using this intrinsic control, we can easily decrease the number of components needed to obtain the computational power of a Turing machine. Thus, we arrive to limits of these systems by showing for them the frontier between the decidability and undecidability. At the end of this chapter we propose a generalisation of the method of directing molecules which we shall further use in Chapters 7 and 8.

Chapter 7

We consider in this chapter another extension of H systems: *test tube systems*, or communicating distributed H systems. We show that the open problem of the computational power of such systems having two tubes stimulated the apparition of numerous variants of these systems. We introduce here a new variant, *test tube systems with alternating filters*, and we show that in this case two tubes suffice in order to generate all recursively enumerable languages. Moreover, it is possible to place rules in the first tube in a way that permits to have no rules in the second tube which is used only as a garbage collector. Another contribution of this result is that it shows that H systems are already powerful enough and it is sufficient to make small modifications in order to go from regularity to universality. The method of directing molecules plays an important role in this chapter, and most of the proofs use it thoroughly.

Chapter 8

This chapter contains another variant of test tube systems, *modified test tube systems*. A small detail differentiates these systems, but it is enough to obtain with two tubes only the computational power of a Turing machine. We remark that this result is very difficult to obtain without using our method of directing molecules because the corresponding systems have an extremely complex behaviour. Also, the concept of mobile trash, when the unwanted molecules avoid the rules which may be applied to them, needs a good synchronisation between different parts of the computation.

Chapter 9

We concentrate in this chapter on *membrane systems*, or P systems, in particular on variants that use the splicing operation. We show that the original definition of splicing P systems is not complete, and we propose several variants in order to complete it. We show that the choice of the variant is important and that the computational power of splicing P systems having one membrane depends on it directly. In the same chapter we consider other variants of splicing P systems, *non-extended splicing P systems* and *splicing P systems with immediate communication*, and we show how it is possible to decrease their main complexity parameter: the number of membranes. We present a final solution, as we show a frontier between the decidability and undecidability for these systems. We also show that the method of directing molecules is applicable in this new framework.

Finally, we present some conclusions and discuss some open problems.

Chapter 1

Preliminaries

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 [15] and [47].

1.1 Words 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 ε. 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.

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:

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

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 $L_1, \ldots, L_n$ belonging to $\mathcal{F}$ also belongs to $\mathcal{F}$: $g(L_1, \ldots, L_n) \in \mathcal{F}$.

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:
$x \underset{G}{\Rightarrow} y$ if
$x = x_1ux_2$, $y = x_1vx_2$ 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 $\overset{*}{\Rightarrow}$ the reflexive and transitive closure of $\Rightarrow$. We write $x \overset{k}{\Rightarrow} y$ if there are words $w_0, \ldots, w_k$ such that $w_i \Rightarrow w_{i+1}, i \geq 0$ and $w_0 = x$ and $w_k = y$. We write $x \underset{P'}{\overset{*}{\Rightarrow}} y$, 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: $S \underset{G}{\overset{*}{\Rightarrow}} w$ is called a *sentential form* of G.

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

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

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 $u_1Au_2 \to u_1xu_2$, where $u_1, u_2 \in (N \cup T)^*$, $A \in N$ and $x \in (N \cup T)^+$.

- *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.

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

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

A *tag system* of degree $m > 0$, see [4] and [32], 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: $w \Rightarrow z$, if $z = w' P_{i_1}$.

The computation of T over the word $x \in V^*$ is a sequence of configurations $x \Rightarrow \ldots \Rightarrow y$, 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 recognises the language L if for all $x \in L$, T halts on x, and T halts only on words from L.

We note that tag systems of degree 2 are able to recognise the family of recursively enumerable languages, see [4] and [32].

We recall an important notion of a *constant* for a a regular language L introduced by Schutzenberger in [48]. We give here an equivalent definition which is simpler, see [12, 13].

Definition 1.2.1. [12, 13] The word $w \in V^*$ is a *constant* for a language L if whenever xwy and uwv belong to L, the words xwv and uwy also belong to L.

We shall frequently use the following remark.

Remark 1.2.1. Let $L \subseteq V^*$ be a regular language. It is clear that if $c \notin V$, then the word c is a constant for languages Lc, cL and LcL. Also, it is easy to see that if $c, d \notin V$, then the words c and d are constants for the language $cL + Ld$. Finally, the set $\{ca : a \in V\}$ is a finite set of constants for the language cLc.

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 $q_0 x \Rightarrow \dots w_1 q_f w_2$, 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.

Chapter 2

State of the art

2.1 Splicing operation

The experiment of Adleman [1] stimulated the study of different biological operations from computational point of view. Operations that he used: synthesis, amplification, separation, extraction and detection as well as similar operations were considered by different authors, and they showed how it is possible to solve different mathematical problems using these operations. E.g., polynomial solutions for SAT, DES and other problems were proposed [19]. An important characteristic of all these operations is that they correspond to real biological operations, therefore it is possible really to implement them *in vitro*. From the other side, the formalisation of these operations is difficult, as a lot of physically originated quantitative parameters must be taken in consideration. This is why some authors chose another approach. They formalise biological operations, but idealise the constraints of physical origin. The obtained operations, like insertion, deletion, splicing etc., have numerous theoretical advantages, but their practical realisation remains difficult. We remark that both approaches are interesting, but we shall especially concentrate on the second one. More precisely, we study the splicing operation and various systems based on this operation.

We should precise that the theoretical study of splicing started long time before the Adleman's experiment. In 1987 T. Head in his article [12] revealed an unexpected connection between molecular biology and the theory of formal languages. By formalising the process of action of restriction and ligation enzymes he obtained a splicing operation defined on strings of characters. Thus, splicing systems, or Head splicing systems, were introduced as a new language generating device.

We do not present here all steps that are necessary to go over to a formal operation from a biological event as all necessary details may be found in [44, 12]. We start directly by the formal description. A splicing rule over an alphabet A is the following quadruplet $(u_1, u_2; u_3, u_4), u_i \in A^*$, which we also denote as follows: $u_1\#u_2\$u_3\#u_4$, where $\{\#, \$\} \notin A$. We apply this rule to a pair of words, called also molecules, $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$ and we obtain two words: $w = x_1u_1u_4y_2$ and $z = y_1u_3u_2x_2$. So we look through initial words for the substrings indicated in the splicing rule (these substrings are called splicing sites). After that the initial strings are cut off at the place indicated by the rule and their ends are exchanged and joined. More exactly, the splicing operation exchange the ends of two words, the place where this exchange is done being indicated by the rule. We denote this by $(x, y) \vdash_r (w, z)$. In this case we say that we perform a 2-splicing. If we consider only w as a result, then we say that we perform an 1-splicing. 2-splicing is a weaker operation than 1-splicing. It can be easily simulated by the last one. This is why 2-splicing is generally used in the literature, as all results obtained for 2-splicing can be easily translated into terms of 1-splicing.

The definition of splicing given above was proposed by Gh. Păun in [37]. There are two more definitions of splicing, one proposed by T. Head in [12], and the other one proposed by D. Pixton in [33]. These definitions have similar properties. A comparison of their computational power may be found in [61]. The mostly accepted definition is the definition of Păun, and we shall concentrate on it in the rest of this work.

We remark that a molecule may contain several identical sites that contain the substrings of the splicing rule. To perform a splicing, only one of these sites is chosen in a non-deterministic way. We assume additionally that the number of molecules x and y that participate in splicing is not restricted. Thus x and y may be used again for a splicing, *i.e.* they are not consumed by the splicing. This consideration has a physical origin as the number of molecules in several grammes of a solution is of order 10^{22}.

Let V be an alphabet and let R be a set of splicing rules. We say that $\sigma = (V, R)$ is a splicing scheme. Now we can define the application of the splicing operation on a language L, $\sigma(L)$, which is the result of all possible splicings of words from L. The properties of languages with respect to the splicing operation are discussed in detail in Chapter 7.2 of [44].

2.2 Head splicing systems

Now we will show how it is possible to use the splicing operation as a language generating device.

We define the iteration of $\sigma(L)$ as follows:
$\sigma^0(L) = L$,
$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0$,
$\sigma^*(L) = \cup_{i\geq 0}\sigma^i(L)$.

A Head splicing system, or an H system, is the couple $H = (\sigma, A)$ where σ is a splicing scheme and $A \subseteq V^*$ is a set of words called axioms. The language generated by such system is $L(H) = \sigma^*(A)$. So the language generated by an H system is the result of the iterative application of the splicing operation over the set of axioms. Intuitively the work of an H system correspond to the following biological experiment: we take DNA molecules, which correspond to axioms, and enzymes, which correspond to splicing rules, we mix them in a test tube, and we wait until all reactions stop. From the other side, this process permits to generate languages starting from a finite set of initial words and from a finite set of splicing rules.

These systems were considered for the first time by T. Head who showed some interesting properties. Later, K. Culik II and T. Harju showed in [6] that the computational power of such systems is limited by the family of regular languages. After that H systems with non-finite sets of rules or non-finite sets of axioms were considered. If we denote by $H(\mathcal{F}_1, \mathcal{F}_2)$ the family of languages generated by H systems having axioms that belong to the family $\mathcal{F}_1$ and having splicing rules which belong to the family $\mathcal{F}_2$, then the following results shown in Tab. 2.1 are known, see also [44].

Table 2.1: The computational power of families $H(\mathcal{F}_1, \mathcal{F}_2)$

	FIN	REG	LIN	CF	CS	RE
FIN	FIN, REG	FIN, RE	FIN, RE	FIN, RE	FIN, RE	FIN, RE
REG	REG	REG, RE	REG, RE	REG, RE	REG, RE	REG, RE
LIN	LIN, CF	LIN, RE	LIN, RE	LIN, RE	LIN, RE	LIN, RE
CF	CF	CF, RE	CF, RE	CF, RE	CF, RE	CF, RE
CS	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE
RE	RE	RE	RE	RE	RE	RE

We placed in this table at the intersection of row $\mathcal{F}_1$ and column $\mathcal{F}_2$ the family $H(\mathcal{F}_1, \mathcal{F}_2)$. If there are two elements in the corresponding cell, then we have a strict inclusion $\mathcal{F}_3 \subset H(\mathcal{F}_1, \mathcal{F}_2) \subset \mathcal{F}_4$.

We underline three results from this table.

$H(FIN, FIN) \subset REG$,

$H(REG, FIN) = REG$,

$H(FIN, REG) \subset RE$.

These inclusions show the limits of the iterative splicing. Even if we use a regular set of axioms, we cannot go beyond regularity with a finite set of rules. The situation changes when a regular set of rules is permitted. However, this approach satisfying the demands of a theoretician seems to be far away from the biological model that supposes that these two parameters are finite. Moreover, by claiming that H systems are a language generating device like formal grammars or Turing machines, we suppose that an infinite language is generated starting from finite parameters. We shall speak later about certain solutions that will have a finite number of components while still keeping a big computational power, but they need an additional control.

The first inclusion above was showed for the first time in [6] using algebraic properties of splicing languages. After that, D. Pixton in [33] simplified the proof using finite automata. This proof may be found with full details in [14]. A new simplification of the proof for Păun's definition was done in [44]. Recently, V. Manca in [20] proposed a proof based on the decomposition of the splicing operation in three operations: cut, paste and delete.

The same inclusion poses the problem of characterisation for the family of finite splicing languages, *i.e.*, when the sets of axioms and splicing rules are finite. This problem is still open, but there are some partial solutions. For example, T. Head in [13] characterised the family of languages generated by splicing systems with rules of the form $u\#\varepsilon\$v\#\varepsilon$ or $\varepsilon\#u\$\varepsilon\#v$. These systems have interesting connections with the concept of a constant for a regular language introduced by Schutzenberger in [48]. More exactly, the words which appear as sites of splicing rules are constants for the generated language. In the same article [13], interesting connections with the theory of finite automata may be found.

Characterisations of splicing language classes may be also found in [12, 11]. S.M. Kim in [17] gave a partial answer for Head's definition of the splicing. A sufficient condition for a regular language to be a splicing language is given in PhD thesis of R. Zizza [61] and an article concerning this subject is in preparation. An overview of works in this direction is given in the cited thesis.

Another problem which was not studied before is the comparison of H systems based on 1-splicing and H systems based on 2-splicing. A remark that 2-splicing may be simulated by 1-splicing was given in [44]. But it was not known if this inclusion is strict. We studied this problem and we proved in [56] in collaboration with R. Zizza that this inclusion is strict. The proof of this result may be found in Chapter 3 of this thesis.

2.3 Extensions of H systems

The study of H systems reached its limits very quickly. This is why their various extensions of them were proposed. A first step is to introduce a terminal alphabet. An extended H system is an H system H equipped with a terminal alphabet T. The language generated by such system consists of all words over the alphabet T that are generated by H. We can produce in the framework of this model all regular languages starting from a finite number of elements.

Another idea proposed afterwards is to consider two finite sets of substrings associated to each rule. The application of the rule is controlled by these sets in the following way: the application of a rule on two words is permitted if and only if all elements of the first set appear, respectively do not appear, in the first word and all elements of the second set appear, respectively do not appear, in the second word. In other respects they function like extended H systems. The obtained systems are called H systems with permitting, respectively forbidding, context. It is possible to reach the power of a Turing machine using these two models.

In the meantime a model which is similar to extended H systems was proposed and that uses multisets of words instead of sets. Again, this model permits to simulate an arbitrary grammar, hence to generate all recursively enumerable languages.

There are other models that are modifications of H systems and that permit to obtain the computational power of a Turing machine. An overview of these models and their properties may be found in Chapter 8 of [44].

2.4 Test tube systems

As we said before, H systems do not have big computational power. In order to simulate more closely biological phenomena and to obtain a sufficient computational power, a new element, the distribution of the computation, was introduced. Most of the models are inspired simultaneously by H systems and distributed grammars, see [47]. We shall concentrate on some of them; an overview of existing models is given in [44].

One of the first models inspired simultaneously by H systems and distributed grammars is the model of *test tube* systems, or communicating distributed H systems, introduced by E. Csuhaj-Varjú, L. Kari and G. Păun in [5]. This model introduces test tubes that are composed of a set of axioms, a set of splicing rules and a set of letters, called filter. The computation in such system consists of two iteratively repeated steps. During the first step, the computing, each tube evolves as an ordinary H system. During the second step, the communication, molecules that are present in each tube are sent to all tubes. Molecules that can pass a filter of a tube, *i.e.*, which are composed only from the letters that form the filter, remain in the corresponding tube and form the initial language for the next step of computation. If a molecule can pass several filters, each of corresponding tubes receives a copy of this molecule. The molecules which cannot pass any filter remain in the tube where they were produced. The language generated by a test tube system consists of all words over the terminal alphabet which are produced by the system at some step and which are in the first tube.

In the article introducing these systems their universality was shown, but without indicating a concrete number of tubes. After that, in [59], the authors proved that 10 tubes are enough in order to generate all recursively enumerable languages. A little bit later the same authors showed that 9 tubes suffice [60]. The number of tubes necessary to obtain the computational power of Turing machines was reduced down to 6, see [39], and finally established to 3, see [34]. We remark that with one tube we obtain extended H systems, therefore we cannot produce more than regular languages. The computational power of test tube systems having two tubes is still an open problem, but Gh. Păun in [44] suppose that the languages generated by such systems belong to the family of context-free languages.

This last problem stimulated the apparition of variants of test tube systems. In the most of cases the changes occur only at the level of filtering. For example, in the variant proposed by R. Freund and F. Freund in [8], the filters are finite unions of sets and a molecule can pass a filter if and only if it is composed of elements of these sets with the condition that all elements from the same set must be present.

The obtained model is very powerful and, in this case, two tubes suffice to generate all recursively enumerable languages.

Another variant was proposed in [10] by P. Frisco and C. Zandron. This variant has a special two letters filter that contains letters or couples of letters. A molecule can pass the filter if and only if it is composed from the letters of the filter or both letters from a couple appear in the molecule. In this case two tubes also permit reach the computational power of Turing machines.

We introduced in [55] another variant of test tube systems: test tube systems with alternating filters. In this variant, the filters are replaced by tuples of filters, each filter in a tuple being a simple set of letters. At each step, the next filter from a tuple is used for the filtering. When the last filter of the tuple is reached, then we continue with the first filter.

Again two test tubes having two tuples of filters suffice in order to generate all recursively enumerable languages. It is possible to have both filters of the first couple that are identical, or to have two filters of each component that differ only in one letter. We also obtained an unexpected result: it is possible to generate all recursively enumerable languages by a system with two tubes where the second tube does not contain any rule and which is used only as a garbage collector, so the universality is obtained with "one tube and a half"! Another contribution of this result is that it shows that H systems are already powerful enough and it suffices to make small modifications in order to go from the regularity to the universality. All these results may be found in Chapter 7 of the present work.

We propose one more variant of test tube systems. This variant, modified test tube systems, differs from the original definition by a "small" detail: if a molecule can pass a filter of a tube different from the tube in which it was produced, then this molecule do not remain any more in the initial tube even if it may pass its filter. This difference, apparently not very important, permits to control the elimination of words from a tube and, consequently, permits to modified test tube systems to simulate arbitrary grammars. The results concerning these systems may be found in Chapter 8 of this thesis.

We remark that the proof of these results on variants of test tube systems uses the method of directing molecules described in our Chapter 6.

2.5 Time-varying distributed H systems

Shortly after the introduction of test tube systems another possibility for distribution of the computation was considered by Gh. Păun in [39]. He started from the biological observation that at each moment there is a set of active enzymes which behave depending on conditions of the environment. If the environment (temperature, acidity or other parameter) changes, then the set of active enzymes also change. In the proposed model, the set of splicing rules changes periodically. More exactly, the model contains a set of words, the axioms, and a finite number of sets of splicing rules, the components. At each step, the current words are spliced once using the rules of the current component, and the result of this splicing forms the set of words for the next iteration. We remark that this elimination procedure is very powerful and it permits to obtain a big computational power. The obtained systems are called time-varying distributed H systems, or TVDH systems.

Initially Gh. Păun showed that 7 components are enough in order to generate all recursively enumerable languages, see [40, 44]. This result was improved by M. Margenstern and Yu. Rogozhin who showed first that TVDH systems with 2 components are able to do universal computations, see [22], and after that they showed that 2 components are enough in order to generate all recursively enumerable languages, see [21, 24]. Their proof is based on a simulation of a Turing machine, so it is strictly sequential: only one molecule which encodes the tape and the state of the machine shall be present in the system. But this does not correspond to the parallel nature of biological transformations where several evolutions may occur in the same time. In the meanwhile, another solution was proposed by A. Păun in [35] who showed that it is possible to simulate the work of an arbitrary grammar with 4 components. In his solution several evolutions of molecules are made in the same time, hence his variant is parallel. This result was improved by M. Margenstern and Yu. Rogozhin who showed that a type-0 grammar can be simulated with 3 components, see [23]. The same authors showed that TVDH systems with one component are universal, see [27], and after that they showed that it is possible to generate all recursively enumerable languages with only one component in a sequential way, see [26]. We improved the above results by showing in collaboration with M. Margenstern and Yu. Rogozhin that it is possible to generate any recursively enumerable language in a parallel way with 2 components, see [28]. Finally, in the framework of the same collaboration, the final point was reached by showing that it is possible to generate all recursively enumerable languages in a parallel way with one component, see [29]. These two results may be found in Chapters 4 and 6 of the present thesis.

We also add that a computer program for TVDH system simulation that was developed during our master thesis was widely used for construction and error checking of the last two systems. This program also permitted to find errors in several articles concerning TVDH systems. For example, the systems in [23] and [44] need small corrections and the proof given in [35] must be completely rewritten. Further details on this subject may be found in our Chapter 4.

A study of TVDH systems from a computational point of view may be found in our master thesis [51] where TVDH systems for main arithmetic operations (addition, multiplication, exponentiation and division) and for the Ackermann function were explicitly given. An implementation of the Euclid's algorithm in TVDH systems may be found on the web site of the author, see [49]. On the same page, one can find the developed program.

The big computational power of TVDH systems is due to their definition, in particular to the fact that only one splicing is done and only the result of this splicing is kept. From a biological point of view it is more natural to suppose that the number of splicings is not limited and that none of molecules that participate in splicing is eliminated. These observations lead us to enhanced time-varying distributed H systems, or ETVDH systems, which were introduced by M. Margenstern and Yu. Rogozhin in [23] under the name of extended distributed H systems. These systems are a combination of TVDH systems and H systems, and we can treat them as TVDH systems where each component behaves like an H system. The elimination is permitted only if the considered molecule cannot enter any rule of the current component.

Now one component is not enough in order to obtain a big computational power and, in this case, the generated language is limited by the family of regular languages. But two components are sufficient to generate all recursively enumerable languages, as M. Margenstern and Yu. Rogozhin proved in [25]. Since this solution uses a simulation of a Turing machine, *i.e.*, a sequential simulation, the same problem of a parallel generation of recursively enumerable languages may be posed. We investigated this problem and we progressively found solutions, at first with 4 components [50], after that with 3 components [52], and, finally, with 2 components, see [54]. Therefore, for these systems the problem of parallel generation of recursively enumerable languages is solved. The last two results may be found in Chapters 5 and 6 of the present work.

2.6 Membrane systems

Until now we discussed H systems, their extensions, and their modifications. We note that the inspiration for these models come from the ways living beings manipulate their DNA. Another approach was developed by Gh. Păun who placed himself at the level of a living cell. By inspiring from different intra- and inter-cellular processes he created a new computational paradigm: *membrane systems*, or *P systems*.

Cell is considered as a set of nested membranes that contain objects and evolution rules. The spacial arrangement of membranes may be represented by a Venn diagram, see Fig. 2.1, or by a tree whose nodes represent membranes and whose edges are defined by the relation "be contained in", see Fig. 2.2. The same structure can be defined by a string of matching parenthesis where the corresponding parenthesis are marked by the same label. For example, the structure represented in Fig. 2.1 may be described by the following word: $[_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ [_5\ ]_5\ [_6\ ]_6\ ]_4\ ]_1$.

The base model is very abstract and it does not specify neither the nature of objects nor the nature of rules. Numerous variants specify these two parameters by obtaining a lot of different computing models. There are at the moment more than a hundred of variants and their number increases permanently. More details may be found on the P systems web page [58] which contains an excellent collection of references on membrane systems.

The compartments delimited by membranes contain multisets of objects. The nature of these objects may be very different. Firstly, we can consider non-structured objects by making an analogy with chemical substances in the interior of the cell. After that, it is possible to structure the objects and to consider them as strings of characters. There are even variants which deal with complicated objects like graphs, images etc. More details on these variants may be found in [43], as well as on the aforementioned web page [58].

The evolution rules, which are generally placed into membranes, present a big variety as well. Their complexity depend on the type of objects used. For simple objects, the rules transform a multiset of objects in another multiset of objects. For structured objects like strings of characters, rewriting or splicing rules are used.

One of the important properties of membrane systems is the communication.

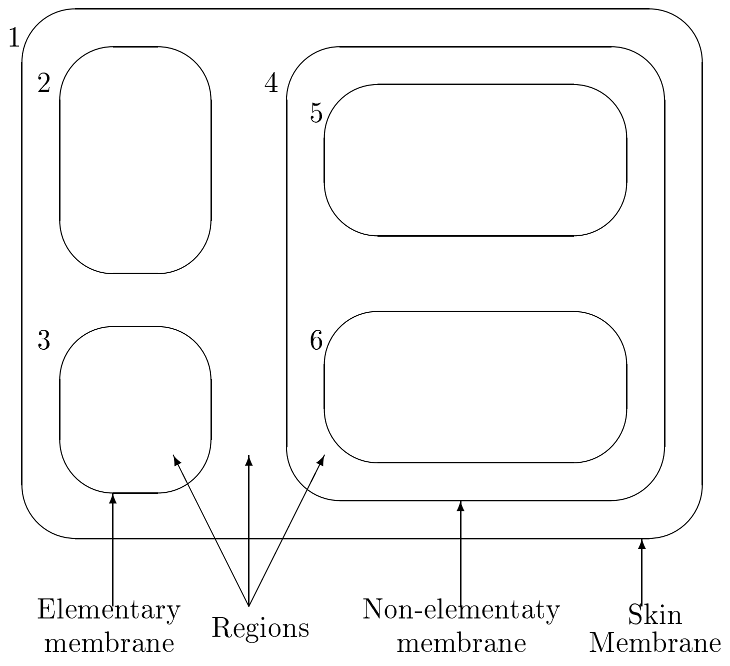


Figure 2.1: A membrane structure

All rules contain target indicators. If a rule contain the target indicator “out”, then the result of its application is sent to the membrane which is immediately upper. If the rule contain the target indicator “here”, then the result remains in the same membrane. If the target indicator “in” is present, then the result is sent to an inner membrane chosen non-deterministically.

This property of communication is so powerful, that it suffices by itself for a big computational power, as in the case of membrane systems with symport/antiport. These systems have two types of rules: symport rules, when several objects go together from one membrane to another, and antiport rules, when several objects from two membranes are exchanged. In spite of a simple definition, one membrane is enough to simulate a register machine and, consequently, to generate all recursively enumerable languages.

Another variant of membrane systems permits the modification of the membrane structure by creation or deletion (dissolution) of membranes. These systems are very interesting because they permit to solve NP-complete problems in a polynomial and even linear time, see [43, 58].

We studied a particular class of membrane systems, *splicing P systems*, which is mixture of membrane systems and H systems. These systems have strings of characters as objects and splicing rules enriched with target indicators as evolution

$$\begin{tikzcd}
& 1 \arrow[dl] \arrow[d] \arrow[dr] & & \\
2 & 3 & 4 \arrow[dl] \arrow[dr] & \\
& 5 & & 6
\end{tikzcd}$$

Figure 2.2: The tree representing the membrane structure from Fig. 2.1

rules. Moreover, contrarily to most variants of membrane systems, we use sets of words instead of multisets, in each membrane. A step of computation represents simultaneous application of all rules in each membrane followed by communication of resulting molecules depending on target indicators. The result consists of all molecules over a terminal alphabet sent outside of the skin membrane.

These systems were introduced by Gh. Păun in [41] where their universality for systems with 4 membranes was showed. The number of membranes necessary to generate all recursively enumerable languages was decreased to 3 in [45] and after that to 2, see [36]. The last result was improved from the point of view of the size of splicing rules by P. Frisco in [9].

We investigated splicing P systems having only one membrane. We found that the initial definition given by Gh. Păun in [41] is not complete and that it needs some precisions. Even the revised definition which appeared in [43] does not cover all possibilities. We completed this definition by proposing 5 variants, and we showed that the computational power depend on the variant which is used. We showed that for the variant which is closest to the definition given by Gh. Păun, the computational power is restricted by the family of regular languages. After that, we showed that by considering three other variants we obtain all recursively enumerable languages. For the remained variant we did not found a description of the generated language yet, but we suppose that it is regular. More details may be found in Chapter 9 of the present thesis.

We also studied the non-extended variant of splicing P systems. In the case of these systems, we do not consider anymore terminal alphabet: all strings sent outside of the skin membrane are considered as a result. Gh. Păun in [43] showed that by using 2 additional membranes it is possible to transform an extended P system in a non-extended one. Since there are extended P systems with 2 membranes, which generate all recursively enumerable languages, the same computational power is obtained with 4 membranes in the non-extended case. We showed that by carefully rearranging the rules it is possible to arrive to two membranes only. This result cannot be improved because with one membrane in the non-extended case we can generate only a subset of regular languages. This result was proved in [53] and it can be found in our Chapter 9.

Another variant of splicing P systems, splicing P systems with immediate communication, was proposed by C. Martín-Vide, Gh. Păun and A. Rodríguez-Patón in [30]. In this variant, the result of the application of each rule must go outside the membrane in which it was produced. Such systems are indeed a particular case of splicing P systems where all rules have the four possible combinations of target indicators "in" and "out". These systems are able to generate all recursively enumerable languages, but the original article did not indicate a fixed number of membranes. We found similarities between these systems and TVDH systems and we showed that 2 membranes suffice in order to generate all recursively enumerable languages. We also showed that with one membrane we can generate only finite languages, therefore we found a frontier between the decidability and undecidability for these systems. Moreover, we found a connection between these systems and membrane systems with splicing rules attached to membranes and not to the regions, see [7].

Now we start a formal analysis of the splicing operation and of H systems.

Chapter 3

1-splicing vs 2-splicing

We introduce in this chapter the central notions of our thesis, the splicing operation and Head splicing systems, or H systems. We study two possible definitions of splicing, 1-splicing and 2-splicing, and we show classes of languages which belong to families of 1-splicing and 2-splicing languages. These classes are obtained starting from languages cL, Lc, LcL, cLc, where $L \subseteq V^*$ is a regular language and $c \notin V$. We demonstrate for the first time that some of these classes are 1-splicing languages, but cannot be 2-splicing languages. Therefore, the family of 2-splicing languages is strictly included into the family of 1-splicing languages. We also show non-trivial classes of regular languages which cannot be splicing languages.

3.1 Formal definition of H systems

We give in this section the formal definition of H systems and repeat some known results.

We understand by an (abstract) *molecule* a word over an alphabet.

Definition 3.1.1. A *splicing rule* (over an alphabet V) is the 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: $\frac{u_1 \mid u_2}{u_3 \mid u_4}$. The strings u_1u_2 and u_3u_4 are called splicing *sites*.

We say that a word x *match* the rule r if x contains an occurrence of one of the two sites of r. We say also 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. We say also in this case that x or y may *enter* the rule r. When x and y having $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$ can enter a rule $r = u_1\#u_2\$u_3\#u_4$ we can define the application of r to the couple x, y. The result of this application is w and z where $w = x_1u_1u_4y_2$ et $z = y_1u_3u_2x_2$. We say also that x and y are spliced and w and z are the result of this splicing. We write this as follows: $(x, y) \vdash_r (w, z)$. We say also in this case that we perform a *2-splicing*. If we take only w as a result in the previous definition, we say that we perform an *1-splicing*. We write it as follows: $(x, y) \vdash_r w$. We can also denote a 2-splicing as follows:

$$\frac{x_1u_1 \mid u_2x_2}{y_1u_3 \mid u_4y_2} \vdash_r \frac{x_1u_1u_4y_2}{y_1u_3u_2x_2}.$$

We can write 1-splicing similarly.

Example 3.1.1.

Let us consider the rule r: $\frac{h \mid o}{l \mid a}$ and let us apply it to two molecules *show* and *blame*:

$$\frac{s\mathbf{h} \mid \mathbf{o}w}{b\mathbf{l} \mid \mathbf{a}me} \vdash_r \frac{s\mathbf{h}\mathbf{a}me}{b\mathbf{l}\mathbf{o}w}.$$

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

For a splicing scheme $\sigma = (V, R)$ and for a formal language $L \subseteq V^*$ we define:

$\sigma_1(L) \overset{\text{def}}{=} \{w \in V^* | \exists x, y \in L, \exists r \in R : (x, y) \vdash_r w\}$.

$\sigma_2(L) \overset{\text{def}}{=} \{w, z \in V^* | \exists x, y \in L, \exists r \in R : (x, y) \vdash_r (w, z)\}$.

Now we can introduce the iteration of the splicing operation.

$\sigma_j^0(L) = L$,

$\sigma_j^{i+1}(L) = \sigma_j^i(L) \cup \sigma_j(\sigma_j^i(L)),\ i \geq 0$,

$\sigma_j^*(L) = \cup_{i \geq 0} \sigma_j^i(L)$.

where $j \in \{1, 2\}$.

We shall prove now that the splicing operation preserve the regularity of a language.

Theorem 3.1.1. *Let $L \subseteq T^*$ be a regular language and let $\sigma = (T, R)$ be a splicing scheme. Then the language $\mathcal{L} = \sigma_1(L)$ is a regular language.*

Proof. We construct a regular grammar which generates $\mathcal{L}$. To obtain this grammar we take the regular grammar G_1 which generates L and we transform its set of productions. We add some regular productions and we eliminate some initial ones thus permitting to simulate the application of σ on L.

Now we consider the regular grammar $G_1 = (N_G, T, S, P_1)$ such that $L(G_1) = L$. We do not detail the symbols of N_G because they are not relevant for our proof. We may assume that all rules of P_1 are of the form $A \to aB$ and $A \to a$, where $A, B \in N_G$ or $a \in T$. We can also assume that G_1 does not contain ε-productions and unreachable symbols.

We consider also the following sets of productions:

$$P_R = \left\{ U_S^i \to u_1^i u_4^i U_E^i : \exists r_i = \frac{u_1^i \mid u_2^i}{u_3^i \mid u_4^i} \in R \right\}.$$

P_1': contains all productions from P_1 where a prime character (') is added to each non-terminal symbol.

$$P_c = \left\{ \begin{array}{c} A \to U_S^i \\ U_E^i \to F' \ (F \neq \varepsilon) \\ U_E^i \to \varepsilon \ (F = \varepsilon) \end{array} : (A, B, C, D, E, F) \in ParseRule_i \right\}.$$

where the set $ParseRule_i$ is defined by:

$$ParseRule_i = \Big\{(A,B,C,D,E,F) : \exists A \overset{*}{\underset{G_1}{\Rightarrow}} u_1^i B \overset{*}{\underset{G_1}{\Rightarrow}} u_1^i u_2^i C,\ \exists D \overset{*}{\underset{G_1}{\Rightarrow}} u_3^i E \overset{*}{\underset{G_1}{\Rightarrow}} u_3^i u_4^i F,$$
$$\exists r_i = \frac{u_1^i \mid u_2^i}{u_3^i \mid u_4^i} \in R\Big\}.$$

In particular, C and F may be equal to ε. If $F = \varepsilon$, then the second production of P_c becomes $U_E^i \to \varepsilon$.

Briefly speaking, P_R contains productions which permit to simulate the application of rules $r_i \in R$, P_1' contains a copy of initial productions and P_c makes the connection between P_1 and the previous two sets.

Let us consider $P = P_1 \setminus \{A \to a \in P_1\} \cup P_1' \cup P_R \cup P_c$.

We take $G = (N,T,S,P)$, where $N = N_G \cup N_G' \cup \{U_S^i, U_E^i : \exists r_i \in R\}$. It is easily seen that G is a regular grammar. The rules of P_1 and P_1' are already in the necessary form, while rules of P_R and P_c can be easily transformed into the necessary form by using standard techniques like elimination of renamings and of ε-productions.

We note that in order to obtain a word in G we have to use either a production from P_1', or a production of the form $U_E^i \to \varepsilon$ from P_c. This follows from the fact that we eliminated all terminal productions of P_1 from P.

Fig. 3.1 shows how we transform the grammar G_1 into G. We substitute two derivations in G_1, indicated by a dashed line, by a derivation in G, indicated by a solid line, which corresponds to the splicing of initial words. Furthermore, the initial derivation became inaccessible in G as well.

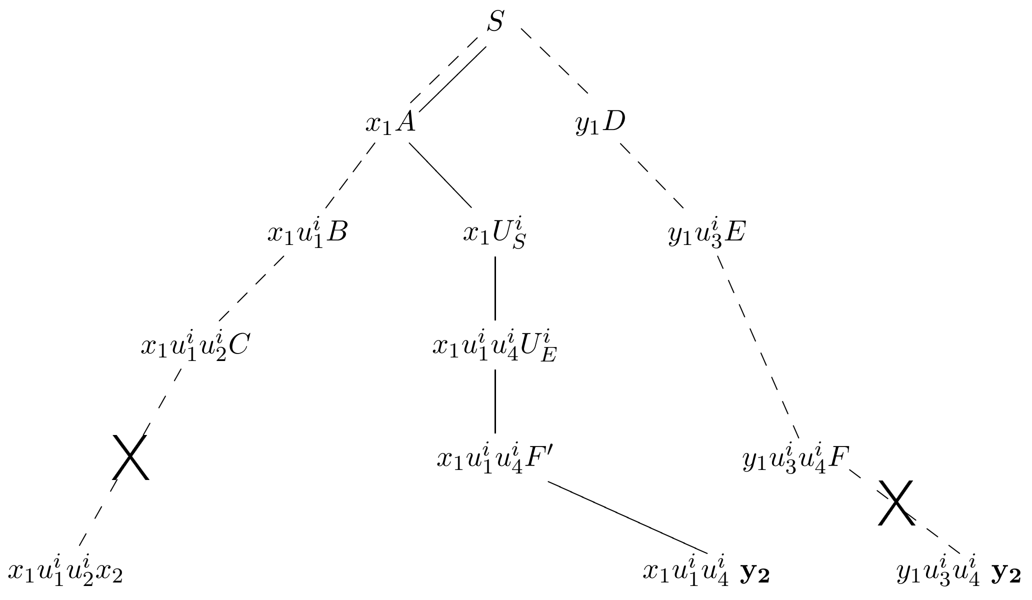


Figure 3.1: Transformation of G_1 into G

We affirm that $L(G) = \sigma_1(L)$.

The Fig. 3.1 shows the idea of the proof as well. A solid line represents a derivation in G, while a dashed line represents derivations in G_1. For one inclusion we shall construct a solid line derivation from two dashed ones. For another inclusion we shall reconstruct two dashed line derivations from the solid one. The rule of R which was applied can be reconstructed from these derivations.

1. $\mathcal{L} \subseteq L(G)$**.**

Let w be in $\mathcal{L}$. In this case $w \in T^*$ and there are words $x_1u_1^iu_2^ix_2$ and $y_1u_3^iu_4^iy_2$ in $\mathcal{L}$ and a rule $\dfrac{u_1^i \mid u_2^i}{u_3^i \mid u_4^i}$ in R such that $(x, y) \vdash_r w = x_1u_1^iu_4^iy_2$.

As $x_1u_1^iu_2^ix_2$ is in $\mathcal{L}$, we have the following derivation in G_1:
$S \underset{P_1}{\overset{*}{\Rightarrow}} x_1A \underset{P_1}{\overset{*}{\Rightarrow}} x_1u_1^iB \underset{P_1}{\overset{*}{\Rightarrow}} x_1u_1^iu_2^iC \underset{P_1}{\overset{*}{\Rightarrow}} x_1u_1^iu_2^ix_2$.

Similarly, $y_1u_3^iu_4^iy_2 \in \mathcal{L}$ implies that there exists the following derivation:
$S \underset{P_1}{\overset{*}{\Rightarrow}} y_1D \underset{P_1}{\overset{*}{\Rightarrow}} y_1u_3^iE \underset{P_1}{\overset{*}{\Rightarrow}} y_1u_3^iu_4^iF \underset{P_1}{\overset{*}{\Rightarrow}} y_1u_3^iu_4^iy_2$.

Then there is the following derivation in G:
$S \underset{P_1}{\overset{*}{\Rightarrow}} x_1A \underset{P_c}{\overset{1}{\Rightarrow}} x_1U_S^i \underset{P_R}{\overset{1}{\Rightarrow}} x_1u_1^iu_4^iU_E^i \underset{P_c}{\overset{1}{\Rightarrow}} x_1u_1^iu_4^iF' \underset{P_1'}{\overset{*}{\Rightarrow}} x_1u_1^iu_4^iy_2$, as if $F \underset{P_1}{\overset{*}{\Rightarrow}} y_2$, then $F' \underset{P_1'}{\overset{*}{\Rightarrow}} y_2$.

We also have $x_1u_1^iu_4^iy_2 \in T^*$, consequently, $w \in L(G)$.

2. $L(G) \subseteq \mathcal{L}$**.**

We note that it is not possible to obtain a terminal word in G not using productions of P_1', or one of productions $U_E^i \to \varepsilon$ from P_c. Similarly, in order to reach a non-terminal symbol having a prime, or U_E^i, we have to use productions from P_R and P_c.

We recall also that the idea of the proof is to transform a derivation in G into two derivations in G_1, which produce two words, to which a splicing rule may be applied.

Let w be in $L(G)$. Then there is a rule $U_E^i \to F'$ in P_c (the case of the rule $U_E^i \to \varepsilon \in P_c$ is similar) such that there is the following derivation of w:
$S \underset{P_1}{\overset{*}{\Rightarrow}} x_1A \underset{P_c}{\overset{1}{\Rightarrow}} x_1U_S^i \underset{P_R}{\overset{1}{\Rightarrow}} x_1u_1^iu_4^iU_E^i \underset{P_c}{\overset{1}{\Rightarrow}} x_1u_1^iu_4^iF' \underset{P_1'}{\overset{*}{\Rightarrow}} x_1u_1^iu_4^iy_2 = w$.

Consequently, there is a rule $r_i = \dfrac{u_1^i \mid u_2^i}{u_3^i \mid u_4^i}$ in R and the following two derivations in G_1:

$S \underset{P_1}{\overset{*}{\Rightarrow}} x_1A \underset{P_1}{\overset{*}{\Rightarrow}} x_1u_1^iB \underset{P_1}{\overset{*}{\Rightarrow}} x_1u_1^iu_2^iC \underset{P_1}{\overset{*}{\Rightarrow}} x_1u_1^iu_2^ix_2$.

$S \underset{P_1}{\overset{*}{\Rightarrow}} y_1D \underset{P_1}{\overset{*}{\Rightarrow}} y_1u_3^iE \underset{P_1}{\overset{*}{\Rightarrow}} y_1u_3^iu_4^iF \underset{P_1}{\overset{*}{\Rightarrow}} y_1u_3^iu_4^iy_2$.

The existence of A, B, C, D, E, F is guaranteed by the definition of P_c. We can also generate the second word, because D is accessible from the axiom.

So, we obtain $x = x_1u_1^iu_2^ix_2$ and $y = y_1u_3^iu_4^iy_2$ which belong to $L(G_1)$. In this case, x and y belong to L.

Therefore, we can apply r_i to x and y: $\dfrac{x_1u_1^i \mid u_2^ix_2}{y_1u_3^i \mid u_4^iy_2} \vdash_{r_i} x_1u_1^iu_4^iy_2$.

As $w \in T^*$, we conclude that w is in $\mathcal{L}$.

□

If we iterate the process of construction of G starting from G_1 and using each time the obtained grammar instead of G_1, and if we do not eliminate productions of P_1, then we obtain a grammar which simulates the iteration of σ on L. The proof of this affirmation is non-trivial and can be found in [44], where this result is proved in terms of finite automata.

Definition 3.1.2. [12, 14] A *Head splicing system* or an H system is the pair $H = (\sigma, A) = ((V, R), A)$, written shortly as $H = (V, A, R)$, where V is a finite alphabet, $A \subseteq V^*$ is a set of initial words, called axioms, and $R \subseteq V^* \times V^* \times V^* \times V^*$ is a set of splicing rules.

We say that the H system H is finite if its sets A and R are finite.

The *language generated* by the splicing Head system H based on 2-splicing, respectively 1-splicing, is $L(H) \overset{\text{def}}{=} \sigma_2^*(A)$, respectively $L(H) \overset{\text{def}}{=} \sigma_1^*(A)$.

Therefore, the language generated by the H system H is the set of all molecules produced by the iterative application of rules from R to copies of molecules obtained before, and starting with A as initial set.

Example 3.1.2.

In this example we use 2-splicing.

Let us consider the system $H = ((V, R), A)$ where $V = \{a, b\}$, $R = \left\{ r : \begin{array}{c|c} a & b \\ \hline a & a \end{array} \right\}$ and $A = \{abaa\}$.

We start with $abaa$. Then, $\sigma_2^0(A) = \{abaa\}$. We can apply the rule r to two instances of this molecule: $(a|baa, aba|a) \vdash_r (aa, ababaa)$, where we emphasised the splicing sites by $|$. We obtain $\sigma_2^1(A) = \{abaa, aa, ababaa\}$. At the next step we have the following applications:

$(a|baa, a|a) \vdash_r (aa, abaa)$,

$(a|baa, ababa|a) \vdash_r (aa, abababaa)$,

$(a|babaa, a|a) \vdash_r (aa, ababaa)$,

$(a|babaa, aba|a) \vdash_r (aa, abababaa)$,

$(a|babaa, ababa|a) \vdash_r (aa, ababababaa)$.

This gives us $\sigma_2^2(A) = \{abaa, aa, ababaa, abababaa, ababababaa\}$

We see that each splicing adds $(ab)^*$ at the left of the existing molecule. Therefore, the language generated by this system is $L(H) = (ab)^*aa$.

Now let us consider the same system, but taking $A = \{aab\}$. The functioning of this new system H' is different. At first we apply the rule r to aab and itself: $(aa|b, a|ab) \vdash_r (aaab, ab)$. After that we have following possibilities:

$(a|b, a|ab) \vdash_r (aab, ab)$,

$(a|b, a|aab) \vdash_r (aaab, ab)$,

$(aa|b, a|aab) \vdash_r (aaaab, ab)$,

$(aaa|b, a|aab) \vdash_r (aaaaab, ab)$,

We can easily see that we add a at the left of existing molecules. Therefore, the language generated by H' is $L(H') = a^+b$.

Remark 3.1.1. Let $S = (V, A, R)$ and $S' = (V, A, R')$ be two H systems and $R \subseteq R'$. Then $L(S) \subseteq L(S')$.

We denote by $H_1(FIN, FIN)$ the family of languages generated by finite H systems based on 1-splicing. Similarly, we denote by $H_2(FIN, FIN)$ the family of languages generated by finite H systems based on 2-splicing. The language generated by an H system based on 1-splicing, respectively 2-splicing, is called 1-splicing language, respectively 2-splicing language.

In the remaining of this chapter we consider non-finite regular languages and finite H systems only.

Proposition 3.1.2. *[14, 44]* $H_2(FIN, FIN) \subseteq H_1(FIN, FIN)$.

This proposition is based on the following reasoning. An H system is called symmetric if for any rule $r = u_1\#u_2\$u_3\#u_4 \in R$, R contain the rule $r' = u_3\#u_4\$u_1\#u_2$. It is easy to observe that an H system based on 2-splicing is implicitly supposed to be symmetric. Consequently, every language produced by a system from the family $H_2(FIN, FIN)$ can be produced by a system in $H_1(FIN, FIN)$ which is symmetric. This argument was used in [14].

This property is one of the reasons why 2-splicing is used mainly in the literature: all results obtained for 2-splicing can be easily reformulated in terms of 1-splicing.

Now we show the limits of finite splicing systems.

Theorem 3.1.3. *[33, 14, 44]* $H_1(FIN, FIN) \subseteq REG$.

This result holds even if a regular set of axioms is used. The proof of this result may be found in [33] and [14]. The book [44] contains a proof of this result based on a simulation of the splicing operation by a finite automata. This proof is much easier than the previous ones. It uses a construction similar to the one used during the proof of Theorem 3.1.1, but translated in terms of finite automata.

Now we show that the previous inclusion is a strict one.

Proposition 3.1.4. $L_{aa} = (aa)^* \in REG \setminus H_1(FIN, FIN)$.

Proof. We shall prove this statement by contradiction. Let $S = (V, A, R)$ be an H system which generates L_{aa}. We have $V = \{a\}$. Let also $r \in R$, $r = a^k\#a^m\$a^p\#a^q$. Consider an integer $i \geq 1$ such that $k + m + i > p + q$ and that $k + m + i$ is even. Let us consider now the word $x = a^{k+m+i}$. By hypothesis $x \in L(S)$. We have now two cases depending on the parity of $k + q$.

1. $k + q$ is even. We can have the following application of r to x and to itself.
   $(aa^k|a^m a^{i-1}, a^{k+m+i-(p+q)}a^p|a^q) \vdash_r a^{k+q+1} \notin L_{aa}$.
   This contradicts the hypothesis because S is closed with respect to the splicing.
2. $k + q$ is odd. We can have the following application of r to x and to itself.
   $(a^k|a^m a^i, a^{k+m+i-(p+q)}a^p|a^q) \vdash_r a^{k+q} \notin L_{aa}$.
   This gives us a contradiction.

□

Therefore, we obtained the following result.

Theorem 3.1.5. *$H_1(FIN, FIN) \subset REG$.*

3.2 Examples of classes of splicing languages

We shall concentrate below on languages obtained by union of a regular language $L \subseteq V^*$ with the languages Lc, cL, LcL and cLc, where $c \notin V$, *i.e.* c is a constant for these languages. The next theorem is a corollary of a more general result proved by T. Head in [13]. It shows that some of the languages above are splicing languages.

Theorem 3.2.1. *[13] Let $L \subseteq V^*$ be a regular language and $c, d \notin V$. Then the languages Lc, cL, LcL, cLc and $cL + Ld$ are in $H_2(FIN, FIN)$. Moreover, each rule of the H system which generates Lc, respectively cL, LcL, cLc and $cL+Ld$, is of form $m\#\varepsilon\$m'\#\varepsilon$ or $\varepsilon\#m\$\varepsilon\#m'$, where m and m' are constants for Lc, respectively cL, LcL, cLc and $cL + Ld$. The words m and m' contain also c in the case of languages Lc, cL, LcL and cLc.*

Remark 3.2.1. We note that the author used 1-splicing in [13]. But, as it was shown in [2, 3], this result holds in case of 2-splicing.

3.2.1 2-splicing languages

Now we present some classes of 2-splicing languages that can be obtained starting from the languages described above.

Proposition 3.2.2. *Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then the regular languages $\mathcal{L} = L + Lc^*$ and $\mathcal{L}' = L + c^*L$ are 2-splicing languages.*

Proof. Let us consider the language $\mathcal{L}$. From Theorem 3.2.1 we obtain that there is an H system $S = (V \cup \{c\}, A, R)$ based on 2-splicing which generates Lc. Let $\bar{S} = (V \cup \{c\}, A, R \cup \{r\})$, where $r = \varepsilon\#c\$c\#\varepsilon$. It is clear that $L(S) \subseteq L(\bar{S})$, see Remark 3.1.1.

It is easy to see that $L(\bar{S}) = \mathcal{L}$. If we apply r to lc and to lc, $l \in L$, we obtain l and lcc. If we apply once more r to lc and to lcc, we obtain $lccc$ and so on.

More formally, at first we shall show by induction that $\mathcal{L} \subseteq L(\bar{S})$. Let w be in $\mathcal{L}$. If w is in Lc, we obtain that w is in $L(S)$ which is a subset of $L(\bar{S})$. If w is in L, then there is a word wc in Lc and we can generate it in $\bar{S}$. After that we can apply r to wc and itself: $(wc, wc) \vdash_r (w, wcc)$ obtaining w.

Now, let us suppose that w is in Lc^i, $i > 0$, *i.e.* $w = w'c^i$, where $w' \in L$. Let us show that $w'c^{i+1} \in L(\bar{S})$. As wc and wc^i are in $L(\bar{S})$ by the induction hypothesis, we obtain wc^{i+1} by applying r to these two words: $(wc, wc^{i-1}) \vdash_r (w, wc^i)$.

Conversely, let w be in $L(\bar{S})$. We use the induction on the number k of iterations of σ used to produce w. If $k = 0$, then the word w is in A which is a part of $L(S)$

which is equal to Lc. Let us consider now a word w produced in $k+1$ iterations of σ. Let the last application of σ be the following: $(x, y) \vdash_{r'} (w_1, w_2)$ and $w \in \{w_1, w_2\}$. By the induction hypothesis x and y are in $\mathcal{L}$. If r' is a rule from R, we get that x and y are in Lc^+, because the sites of r' contain c, see Theorem 3.2.1. Consequently, w is also in Lc^+. If $r' = r$, we obtain that $x = x_1x_2$, with $x_1 \in Lc^*$ and $x_2 \in c^+$ and $y = y_1y_2$, with $y_1 \in Lc^+$ and $y_2 \in c^*$. In this case $w_1 = x_1y_2$ belongs to Lc^* and $w_2 = y_1x_2$ belongs to Lc^+, *i.e.*, w is in $\mathcal{L}$.

We can show in a similar manner that $\mathcal{L}'$ is a 2-splicing language. □

Proposition 3.2.3. *Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then, regular languages $\mathcal{L} = L + Lc$ and $\mathcal{L}' = L + cL$ are 1-splicing languages.*

Proof. We use the same construction as in the previous proposition.

As $c \notin V$, c is a constant for Lc. Therefore, there an H system based on 1-splicing $S = (V \cup \{c\}, A, R)$ such that $Lc = L(S)$ (Theorem 3.2.1). Similarly to Proposition 3.2.2, we consider the rule $r = \varepsilon\#c\$c\#\varepsilon$ and the system $\bar{S} = (V \cup \{c\}, A, R \cup \{r\})$. It is obvious that $L + Lc = L(\bar{S})$, because rules of R permit to produce Lc, while r permits to produce L. We note that it is important to use 1-splicing instead of 2-splicing in order to produce L. □

We do not know if this result remains true if 2-splicing is used, but we suppose this.

Conjecture 3.2.1. Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then, regular languages $\mathcal{L} = L + Lc$ and $\mathcal{L}' = L + cL$ are 2-splicing languages.

We shall show now one possibility to solve this conjecture. Let us suppose that it is possible to find an integer n and words $z_i \in V^*$ and $w_i \in L$, $1 \leq i \leq n$ such that for all $xw_iy \in L$ the following two conditions are satisfied:

a) $xw_i \in L$

b) $\forall w \in L : w = w'z_i \Rightarrow wy = w'z_iy \in L$

If it is possible to find such z_i and w_i, then after adding words w_i to axioms the rules $z_i\#c\$w_i\#\varepsilon$ permit to generate L starting from Lc. Similar conditions may be formulated for the generation of L from cL.

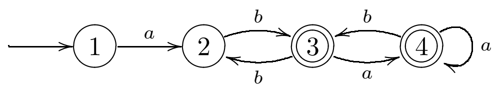
The motivation of the previous statement is the following. First we observe that we can generate the language Lc, see Theorem 3.2.1. After that, the main idea is to produce a word x of L from the word $xc \in Lc$ by erasing c at the end. This may be done by using rules $z_i\#c\$w_i\#\varepsilon$. Conditions a) and b) guarantee the consistence of obtained words because both resulting words must belong to L or Lc.

Example 3.2.1.

Languages $a^* + a^*b$ and $a^* + ba^*$ are examples, which confirm the conjecture above. For the first language it is sufficient to take $i = 1$, $w_1 = a$ and $z_1 = \varepsilon$ in order to satisfy both conditions above. The rule $\varepsilon\#b\$a\#\varepsilon$ then permits to produce a^* from a^*b. We note that the language a^*b may be produced by an H system based on 2-splicing (Theorem 3.2.1).

Example 3.2.2.

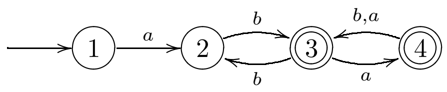
Let us consider the regular language L recognised by the following finite automata:



If we take $i = 2$ and $z_1 = a$, $z_2 = b$, $w_1 = ababa$, $w_2 = abbb$, then both previous conditions are satisfied. In this case the rules $a\#c\$ababa\#\varepsilon$ and $b\#c\$abbb\#\varepsilon$ permit to produce L from Lc. We note that Lc may be generated by an H system based on 2-splicing (Theorem 3.2.1).

Example 3.2.3.

Let us consider the regular language L recognised by the following finite automata:



In this case $i = 4$ and the following 4 rules $ba\#c\$ababa\#\varepsilon$, $aa\#c\$abbb\#\varepsilon$, $ab\#c\$abbb\#\varepsilon$ and $bb\#c\$abbb\#\varepsilon$ satisfy both previous conditions. Therefore, they permit to produce L from Lc. We note that Lc may be generated by an H system based on 2-splicing (Theorem 3.2.1).

3.2.2 Classes of 1-splicing languages that cannot be 2-splicing languages

The next proposition shows a class of regular languages which cannot be 2-splicing languages.

Proposition 3.2.4. *Let $L \subseteq V^*$ be a regular language and $c, d \notin V$. Then, the language $\mathcal{L} = L + cL + Ld$ cannot be a 2-splicing language.*

Proof. We shall prove this result by contradiction. We suppose that there is an H system based on 2-splicing $S = (V, A, R)$ such that $L(S) = \mathcal{L}$. We consider any rule $r \in R$ and we show that using this rule we obtain either a word which does not belong to $\mathcal{L}$ as it will contain two occurrences of letters from V', where $V' = \{c, d\}$, or both resulting words will contain one occurrence of c or d, *i.e.*, we cannot produce words from L by using this rule. We also notice the closure property of $\mathcal{L}$: $\mathcal{L} = \sigma_2(\mathcal{L})$ where σ is the H scheme (V, R).

Now let us consider a rule $r = u_1\#u_2\$u_3\#u_4 \in R$ and let $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, $w_1 = x_1u_1u_4y_2$, $w_2 = y_1u_3u_2x_2$ be such that we can have the following application of r: $(x, y) \vdash_r (w_1, w_2)$.

It is clear that $|u_1u_2|_{V'} \leq 1$ and $|u_3u_4|_{V'} \leq 1$. We have the following cases with respect to the number of c and d in the components u_i of the rule r, $1 \leq i \leq 4$:

1. $|u_i|_{V'} = 0$. We obtain a contradiction if we take x in cL and y in Ld, as w_1 will contain c and d in the same time.

2. $|u_1u_2|_{V'} = 0$ and $u_3u_4 \in \mathit{Fact}(cL)$. We have two subcases:
   (a) $u_3 \neq \varepsilon$. If we take x in Ld and y in cL, then we obtain that $|w_2|_{V'} = 2$, which is a contradiction.
   (b) $u_3 = \varepsilon$. If we take x and y in cL, we have a contradiction, as the resulting word w_1 will contain two letters c.

3. We can reason in a similar way for other combinations for which one of sites contain a letter from V' and another one does not contain any letter of V'.

4. $u_1u_2 \in \mathit{Fact}(cL)$ and $u_3u_4 \in \mathit{Fact}(cL)$. We have 3 subcases:
   (a) $u_3 = \varepsilon$ and $u_1 \neq \varepsilon$ (or $u_1 = \varepsilon$ and $u_3 \neq \varepsilon$). If we take x and y in cL, then we obtain a resulting word containing two occurrences of c.
   (b) $u_1, u_3 = \varepsilon$. Both resulting words are identical to the initial ones.
   (c) $u_1, u_3 \neq \varepsilon$. Both resulting words belong to cL.
   We see that we cannot produce a word from L by using rules of last two types.

5. We can reason in a similar manner for the case when $u_1u_2 \in \mathit{Fact}(Ld)$ and $u_3u_4 \in \mathit{Fact}(Ld)$.

6. $u_1u_2 \in \mathit{Fact}(cL)$ and $u_3u_4 \in \mathit{Fact}(Ld)$. We have 4 subcases:
   (a) $u_1, u_4 \neq \varepsilon$. We obtain a contradiction if we take x in cL and y in Ld, as w_1 contains both c and d.
   (b) $u_1 \neq \varepsilon, u_4 = \varepsilon$. We obtain that $|w_1|_c = 1$ and $|w_2|_d = 1$.
   (c) $u_1 = \varepsilon, u_4 \neq \varepsilon$. We obtain that $|w_1|_d = 1$ and $|w_2|_c = 1$.
   We see that we cannot produce a word from L by using rules of last two types.
   (d) $u_1, u_4 = \varepsilon$. A contradiction is obtained if we take x in cL and y in Ld, as the resulting word w_2 contains both c and d.

Shortly speaking, we have rules of two types. Rules of the first type produce words having one occurrence of c or d. Consequently, these words cannot belong to L. If we suppose that we can generate L by using rules of another type, then we can show for each rule of this type an example of words such that by applying this rule to them we will produce a word having two occurrences of letters from V'. This means that the corresponding word does not belong to $\mathcal{L}$. But this contradicts to the fact that $\mathcal{L}$ is closed with respect to the splicing operation. Consequently, we obtain a contradiction with the fact that $\mathcal{L} = L + cL + Ld$ is a 2-splicing language. □

We can easily derive the same result in the case when $|V'| = 1$, *i.e.*, $c = d$.

Corollary 3.2.5. *Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then, the language $\mathcal{L}' = L + cL + Lc$ cannot be a 2-splicing language.*

The following theorems show that the languages above belong to the family $H_1(FIN, FIN)$. This means that they are in $H_1(FIN, FIN) \setminus H_2(FIN, FIN)$.

Theorem 3.2.6. *Let $L \subseteq V^*$ be a regular language and $c, d \notin V$. Then, the language $\mathcal{L} = L + cL + Ld$ is in $H_1(FIN, FIN) \setminus H_2(FIN, FIN)$.*

Proof. Proposition 3.2.4 gives us that the language $\mathcal{L} = L + cL + Ld$ cannot be in $H_2(FIN, FIN)$. Now it is enough to show that $\mathcal{L}$ is in $H_1(FIN, FIN)$. Since c is a constant for cL and since d is a constant for Ld, cL and Ld are 2-splicing languages, see Theorem 3.2.1, and, consequently, 1-splicing languages, see Proposition 3.1.2. Let $S_c = (V \cup \{c\}, A_c, R_c)$ be the H system which generates cL, *i.e.*, $L(S_c) = cL$, and let $S_d = (V \cup \{d\}, A_d, R_d)$ be the H system which generates Ld, *i.e.* $L(S_d) = Ld$. The language $cL + Ld$ is a 2-splicing language as well, see Theorem 3.2.1. We affirm that the system $S = (V \cup \{c, d\}, A_c \cup A_d,\ R_c \cup R_d \cup \{r\})$, where $r = \varepsilon\#d\$d\#\varepsilon$, generates $\mathcal{L}$, *i.e.* $\mathcal{L} = L(S)$. We note that we can also use the rule $r = \varepsilon\#c\$c\#\varepsilon$. Indeed, we can generate cL and Ld by using rules from R_c and R_d. Finally, the rule $r = \varepsilon\#d\$d\#\varepsilon$ which may be applied to a pair of words from Ld only, permits to generate words from L by 1-splicing. We note that the obtained system S is based on 1-splicing, since r cannot be used to perform a 2-splicing. □

The previous result remain true if we take $\mathcal{L}' = L + cL + Lc$. More exactly, we observe that the proof that $\mathcal{L} = L + cL + Ld$ is a 1-splicing language uses the hypothesis that c and d are two constants for $\mathcal{L}$. From the other side, it is possible to show that $\mathcal{L}' = L + cL + Lc$ is a 1-splicing language because the sets $\{ca \mid a \in V\}$ and $\{ac \mid a \in V\}$ are two finite sets of constants for $\mathcal{L}'$. In this case, in order to generate L we can use the set of rules $r_a = a\#c\$ac\#\varepsilon$, respectively $r_a = \varepsilon\#ca\$c\#a$, for all $a \in V$ in order to generate L. Such a rule r_a is applied to two words from Lc, respectively cL, and produce a word from L.

Theorem 3.2.7. *Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then the language $\mathcal{L} = L + LcL$ cannot be a 2-splicing language.*

Proof. We shall prove this result by contradiction. We suppose that there is an H system based on 2-splicing $S = (V, A, R)$ such that $L(S) = \mathcal{L}$. We consider any rule $r \in R$ and we show that using this rule we obtain either a word which does not belong to $\mathcal{L}$ as it will contain two occurrences of c, or both resulting words will contain one occurrence of c. This means that we cannot produce words from L by using this rule. We also notice the closure property of $\mathcal{L}$: $\mathcal{L} = \sigma_2(\mathcal{L})$ where σ is the H scheme (V, R).

Now let us consider a rule $r = u_1\#u_2\$u_3\#u_4 \in R$ and let $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, $w_1 = x_1u_1u_4y_2$, $w_2 = y_1u_3u_2x_2$ such that we can have the following application of r: $(x, y) \vdash_r (w_1, w_2)$.

It is clear that $|u_1u_2|_c \leq 1$ and $|u_3u_4|_c \leq 1$. Therefore, we have the following cases with respect to the number of c in components u_i of the rule r, $1 \leq i \leq 4$:

1. $|u_i|_c = 0, i = 1, \ldots, 4$.
   We obtain a contradiction if we take x, y such that $|x_1|_c = |y_2|_c = 1$, because $|w_1|_c = 2$.

   We affirm that it is possible to find such x and y because of the form of $\mathcal{L}$. More precisely, if we can apply r to x and to y and if $|x_1|_c = 0$, then necessarily $x = x_1u_1u_2w'cw''$. Since $x_1u_1u_2w' \in L$, there is a word $x' = w_3cx_1u_1u_2w' \in LcL$ having $|x'_1|_c = |w_3cx_1|_c = 1$. Then the rule r may be applied to x' and to y.

2. $|u_1|_c = |u_2|_c = 0$ and $|u_3u_4|_c = 1$

   (a) $|u_3|_c = 1$ ($|u_4|_c = 0$): if we take x such that $|x_2|_c = 1$, we obtain a contradiction, as $|w_2|_c = 2$.

   (b) $|u_3|_c = 0$ ($|u_4|_c = 1$): if we take x such that $|x_1|_c = 1$, we obtain a contradiction again, as this time $|w_1|_c = 2$.

2′. Similarly, we obtain a contradiction in the case when $|u_3|_c = |u_4|_c = 0$ and then $|u_1u_2|_c = 1$.

3. $|u_1|_c = 1$ ($|u_2|_c = 0$ and $|u_3u_4|_c = 1$). We have two subcases:

   (a) $|u_3|_c = 1$ ($|u_4|_c = 0$): then $|w_1|_c = |w_2|_c = 1$, and we cannot produce a word in L using this rule.

   (b) $|u_4|_c = 1$ ($|u_3|_c = 0$): we have a contradiction, as $|w_1|_c = 2$.

3′. Similarly if $|u_2|_c = 1$ ($|u_1|_c = 0$ and $|u_3u_4|_c = 1$).

Shortly, we have rules of two types. Rules of the first type produce words having one occurrence of c. Therefore, these words cannot belong to L. If we suppose that we can generate L by using rules of another type, then we can present for each rule of this type an example of words such that by applying this rule to them we will produce a word having two occurrences of c. This means that the corresponding word does not belong to $\mathcal{L}$. But this contradicts the fact that $\mathcal{L}$ is closed with respect to the splicing operation. Consequently, we obtain a contradiction with the fact that $\mathcal{L} = L + LcL$ is a 2-splicing language.

□

We do not know if the above language is a 1-splicing language, but we suppose that it is the case.

Conjecture 3.2.2. Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then the language $L + LcL$ is in $H_1(FIN, FIN)$, but not in $H_2(FIN, FIN)$.

One of the possibilities to solve this conjecture, *i.e.*, whether the language $L+LcL$ is in $H_1(FIN, FIN)$, is directly related to the solution proposed for Conjecture 3.2.1 whether $L + Lc \in H_2(FIN, FIN)$. We recall that Conjecture 3.2.1 is true if we can find $z_i \in V^*$ and $w_i \in L$ such that for all xw_iy in L the following two conditions are satisfied:

a) $xw_i \in L$,

b) $\forall w \in L : w = w'z_i \Rightarrow wy = w'z_iy \in L$.

In our case, rules $z_i\#c\$cw_i\#\varepsilon$ permit to produce L from LcL. The motivation of this assertion is the same as for Conjecture 3.2.1, but in this case the problem is even simpler as condition a) is always true.

Example 3.2.4.

$\mathcal{L} = a^* + a^*ba^* \in H_1(FIN, FIN) \setminus H_2(FIN, FIN)$.

From Theorem 3.2.7 we obtain that $\mathcal{L} \notin H_2(FIN, FIN)$. Now we use the previous remark and put $w_1 = a$ and $z_1 = \varepsilon$. It is easy to see that condition b) is satisfied. The rule $\varepsilon\#b\$ba\#\varepsilon$ permits to generate a^* from a^*ba^*. To finish the proof we note that a^*ba^* is a 2-splicing language and, consequently, a 1-splicing language, see Theorem 3.2.1.

Example 3.2.5.

If we consider the language L from example 3.2.2, we obtain immediately that $L + LcL$ is a 1-splicing language. The same result holds for the language L from example 3.2.3.

3.2.3 Examples of classes of regular languages that are not 1-splicing languages

We present in this section a class of regular languages which cannot be 1-splicing languages. This class is constructed from the language cLc, $L \subseteq V^*$, $c \notin V$.

Theorem 3.2.8. *Let $L \subseteq V^*$ be a regular language and $c \notin V$. Then, the language $\mathcal{L} = L + cLc$ cannot be a 1-splicing language.*

Proof. We shall prove this result by contradiction. We suppose that there is an H system based on 1-splicing $S = (V, A, R)$ such that $L(S) = \mathcal{L}$. We consider any rule r of R and we show that using this rule we obtain either a word which does not belong to $\mathcal{L}$ as it will contain one occurrence of c, or both resulting words will contain two occurrences of c. This means that we cannot produce words from L by using this rule. We also notice the closure property of $\mathcal{L}$: $\mathcal{L} = \sigma_1(\mathcal{L})$ where σ is the H scheme (V, R).

Now let us consider a rule $r = u_1\#u_2\$u_3\#u_4 \in R$ and let $x = x_1u_1u_2x_2 \in \mathcal{L}$ and $y = y_1u_3u_4y_2 \in \mathcal{L}$. We denote by w the result of the application of r to these two words.

We note that $|u_1u_2|_c \leq 2$ et $|u_3u_4|_c \leq 2$. Therefore, we have to consider only the cases when every $|u_i|_c$, $i = 1 \dots 4$, varies between 0 and 2. We can throw away cases where $|u_1|_c > 0$ or $|u_4|_c > 0$, because in these cases we cannot produce a word from L.

Then we fix $|u_1|_c = |u_4|_c = 0$ and we consider $0 \leq |u_2|_c, |u_3|_c \leq 2$. This gives us 9 cases. We number cases by two digits; the first digit indicates the number of c in u_2 and the second digit indicates the number of c in u_3.

Case 00: (*i.e.*, $|u_2|_c = 0 = |u_3|_c$). Then there are x', y' in L such that $(cx'c, y') \vdash_r w$ and $|w|_c = 1$, therefore $w \notin \mathcal{L}$.

Case 01: (*i.e.*, $|u_2|_c = 0$ et $|u_3|_c = 1$). We take an y which contains the site u_3u_4 of the rule r.
If $u_4y_2 \neq \varepsilon$, then there is x' in L such that $(x', y) \vdash_r w$ and $|w|_c = 1$ as $|u_4y_2|_c = 1$.
If $u_4y_2 = \varepsilon$, then there is x' in L such that $(cx'c, y) \vdash_r w$ as $w = x_1u_1$ and $|w|_c = 1$.

In both cases $w \notin \mathcal{L}$.

Case 02: (*i.e.*, $u_4 = \varepsilon$). Then there is x' in L such that $(cx'c, y) \vdash_r w$ and $|w|_c = 1$ as $w = x_1u_1$. Hence, $w \notin \mathcal{L}$.

Case 10: This case is similar to case 01. We take y' in L and we choose y' or $cy'c$ depending on x and we obtain that $|w|_c = 1$, *i.e.*, $w \notin \mathcal{L}$.

Case 11: we have 4 subcases ($x, y \in \mathcal{L}$):

a) $x_1u_1 = u_4y_2 = \varepsilon$: we obtain $w = \varepsilon$.

b) $x_1u_1, u_4y_2 \neq \varepsilon$: as $|u_2|_c = |u_3|_c = 1$ we obtain that $|x_1u_1|_c = |x_1|_c = 1$ and $|u_4y_2|_c = |y_2|_c = 1$. Then, $|w|_c = 2$ and w cannot be in L.

c,d) either $x_1u_1 = \varepsilon$, or $u_4y_2 = \varepsilon$. If $x_1u_1 = \varepsilon$, we have $w = u_4y_2$ and $|w|_c = 1$. If $u_4y_2 = \varepsilon$, we have $w = x_1u_1$ et $|w|_c = 1$. In both cases $w \notin \mathcal{L}$.

Case 12: (*i.e.*, $u_4 = \varepsilon$). Then there is an x' in L such that $(cx'c, y) \vdash_r w$ and either $w = \varepsilon$ (if $x_1u_1 = \varepsilon$), or $|w|_c = 1$.

Case 20: (*i.e.*, $u_1 = \varepsilon$). Similarly to case 02 we can find an y' in L such that $(x, cy'c) \vdash_r w$ and $|w|_c = 1$.

Case 21: (*i.e.*, $u_1 = \varepsilon$). Similarly to case 12, we obtain that there is an y' in L such that $(x, cy'c) \vdash_r w$ and either $w = \varepsilon$ (if $u_4y_2 = \varepsilon$), or $|w|_c = 1$.

Case 22: We produce ε.

Shortly speaking, rules are of three types. Rules of the first type permit to produce only ε; rules of the second type permit to obtain words w such that $|w|_c = 2$. Consequently, these words cannot belong to L. Rules of the third type permit to obtain different words, but if we suppose that we can generate L by using rules of this type, then we can find for each rule of this type an example of words from L and cLc such that by applying this rule to them we will produce a word having one occurrence of c. This means that the corresponding word does not belong to $\mathcal{L}$. This contradicts the fact that $\mathcal{L}$ is closed under splicing. Consequently, we obtain a contradiction with the fact that $\mathcal{L} = L + cLc$ is 1-splicing language.

□

3.3 Conclusions

We introduced in this chapter the splicing operation and the H systems. We compared the generative power of H systems based on 1-splicing and of H systems based on 2-splicing and we have shown that the set $H_1(FIN, FIN) \setminus H_2(FIN, FIN)$ is not empty. The languages which we used to obtain this result are constructed from constant languages where the constant is defined by a letter $c \notin V$. We note that we can replace this letter by words which are constants for the considered language. Even if these languages seem to be more interesting, their underlying structure is identical to the structure of languages presented before.

H systems introduced in this chapter are very interesting but their computational power is not very big. This is why we shall enrich them with control mechanisms and distribution of computing. This will permit them to reach the power of Turing machines. We shall progressively complicate the obtained systems and the next chapter will present a first extension of Head splicing systems.

Chapter 4

Time-varying distributed H systems

In this chapter we introduce an extension of H systems: extended H systems. We show that these systems are a little more powerful that H systems and that they can generate the family of regular languages when a finite set of axioms and rules is used or the family of recursively enumerable languages if a regular set of rules is used. We also describe the "rotate-and-simulate" method which is often used in the H systems area. This method permits to simulate an arbitrary grammar, by consequent, it can be used to prove a big computational power of some systems. We also introduce another extension of H systems: time varying distributed H systems, or TVDH systems. These systems are based on the biological observation that enzymes, simulated by splicing rules, are not accessible all the time, but only at certain moments. We show that two sets of rules which we call later *components* suffice to generate all recursively enumerable languages by simulating a type-0 grammar. The proof of this result uses the "rotate-and-simulate" method. After that we show that TVDH systems with one component are able to do universal computations. At the end of the chapter we show several examples obtained with the help of a computer program developed during the master thesis of the author. More exactly, we controlled several published articles on TVDH systems and we found with the help of our software that they contain errors. We also show how it is possible to correct these errors in some cases.

4.1 Extended H systems

4.1.1 Definitions and results

Definition 4.1.1. An *extended H system* is the quadruplet $\gamma = (V, T, A, R)$, where V is a finite alphabet, $T \subseteq V$ is the terminal alphabet, $A \subseteq V^*$ is a finite set of initial words, called axioms, and R is a set of splicing rules.

The language generated by the extended H system γ is:

$L(\gamma) = \sigma^*(L) \cap T^*$, where $\sigma = (V, R)$.

So, the language generated by an extended H system γ consists of all words over the terminal alphabet T which are generated by the H system $H = (V, A, R)$.

We denote by $EH(\mathcal{F}_1, \mathcal{F}_2)$ the family of languages generated by extended H systems having the set of axioms which belongs to the family $\mathcal{F}_1$ and having the set of rules which belongs to the family $\mathcal{F}_2$.

Extended H systems are a little more powerful than ordinary H systems.

Theorem 4.1.1. *$REG \subseteq EH(FIN, FIN)$*

Proof. We present here the proof given in [44].

Let $L \subseteq T^*$ be a regular language and let $G = (N, T, S, P)$ be the regular grammar which generates it.

We construct the following extended H system:

$$\gamma = (N \cup T \cup \{Z\}, T, A_1 \cup A_2 \cup A_3, R_1 \cup R_2),$$

where

$$\begin{aligned}
A_1 &= \{S\},\\
A_2 &= \{ZaY \mid X \to aY \in P, X, Y \in N, a \in T\},\\
A_3 &= \{ZZa \mid X \to a \in P, X \in N, a \in T\},\\
R_1 &= \left\{ \frac{\varepsilon \mid X}{Z \mid aY} \mid X \to aY \in P, X, Y \in N, a \in T \right\},\\
R_2 &= \left\{ \frac{\varepsilon \mid X}{ZZ \mid a} \mid X \to a \in P, X \in N, a \in T \right\}.
\end{aligned}$$

If we splice the word ZxX using a rule from R_1, then the result will be of form $ZxaY$ and ZX. It is easy to see that starting from these words we cannot obtain a terminal word. Indeed, we cannot eliminate the symbol Z if these words are used as a first term of splicing. Moreover, no rule is applicable to the word ZxX having $|x| \neq 1$, if we try to use it as a second term of splicing. By consequent, the only possibility to obtain a terminal word is to start from S, use rules from R_1 and end with a rule from R_2. Thus, we can see that the first term of splicing is the one obtained by the previous splicing and the second term of splicing belongs to A_2, or A_3 at the last step. This corresponds to a derivation in G. So, we obtain that $L(\gamma) = L(G) = L$. $\square$

It is clear that the inverse inclusion holds because the family of regular languages is closed with respect to intersection.

4.1.2 The "rotate-and-simulate" method

Now it suffice to make a small step further in our framework in order to obtain a big computational power.

Theorem 4.1.2. $RE \subseteq EH(Fin, REG)$

Proof. We present here the proof given in [44].

Let $G = (N, T, S, P)$ be a type-0 grammar. Let us consider $U = N \cup T \cup \{B\}$, where B is a new symbol. We construct the following extended H system

$$\gamma = (V, T, A, R),$$

where

$$V = N \cup T \cup \{X, X', B, Y, Z\} \cup \{Y_\alpha \mid \alpha \in U\},$$

$$\begin{aligned} A = &\{XBSY,\ ZY, XZ\} \\ &\cup \{ZvY \mid u \to v \in P\} \\ &\cup \{ZY_\alpha,\ X'\alpha Z \mid \alpha \in U\}, \end{aligned}$$

and R contains the following groups of rules:

$$\begin{array}{rlll} \textit{Simulate}: & 1. & Xw\#uY\$Z\#vY, & \text{for } u \to v \in P, w \in U^*, \\ \textit{Rotate}: & 2. & Xw\#\alpha Y\$Z\#Y_\alpha, & \text{for } \alpha \in U, w \in U^*, \\ & 3. & X'\alpha\#Z\$X\#wY_\alpha, & \text{for } \alpha \in U, w \in U^*, \\ & 4. & X'w\#Y_\alpha\$Z\#Y, & \text{for } \alpha \in U, w \in U^*, \\ & 5. & X\#Z\$X'\#wY, & \text{for } w \in U^*, \\ \textit{Terminate}: & 6. & \varepsilon\#ZY\$XB\#wY, & \text{for } w \in T^*, \\ & 7. & \varepsilon\#Y\$XZ\#\varepsilon. & \end{array}$$

We claim that $L(\gamma) = L(G)$.

Let $\sigma = (V, R)$. We examine the functioning of σ, more exactly the possibilities to obtain a word in T^*.

No string of A is in T^*. All rules in R involve a string containing the symbol Z, but this symbol will not appear in the first string produced by splicing. Moreover, the second result of the splicing will contain Z and it can match only the site of rule 7. But it is easy to see that we cannot obtain a terminal string in this case. Therefore, at each step we have to splice a word from A with the string produced at a previous step, excepting the first step when the axiom $XBSY$ is used.

The symbol B marks the beginning of sentential forms of G simulated by σ.

By rules in group 1 we can simulate rules of P. Rules in groups 2 – 5 move symbols from the right hand end of the current string to its left hand end. This permits to simulate rules of P at the right hand end of the string produced by σ. However, because B is always present and marks the place where the string of G begins, we know at each moment which is that string. Namely, if the current string in σ is of the form $\beta_1 w_1 B w_2 \beta_2$, for some markers β_1, β_2 of type X, X', Y, Y_α with $\alpha \in U$, and $w_1, w_2 \in (N \cup T)^*$, then $w_2 w_1$ is a sentential form of G.

We start from $XBSY$, hence from the axiom of G, marked at the left hand end with B and bracketed by X, Y.

Let us see how the rules 2–5 work. Suppose we take a string $Xw\alpha Y$, for some $\alpha \in U$, $w \in U^*$. By a rule of type 2 we get:

$$(Xw|\alpha Y, Z|Y_\alpha) \vdash (XwY_\alpha, Z\alpha Y).$$

The symbol Y_α memorises the fact that α was erased from the right hand end of $w\alpha$. Only a rule of type 3 can be applied to XwY_α:

$$(X'\alpha|Z, X|wY_\alpha) \vdash (X'\alpha wY_\alpha, XZ).$$

We note that the same symbol α removed at the previous step is now added in front of w. Again there is only one way to continue, namely by using a rule of type 4. We get:

$$(X'\alpha w|Y_\alpha, Z|Y) \vdash (X'\alpha wY, ZY_\alpha).$$

If we use now a rule of type 7, removing Y, then X' (and B) can never be removed, so the string cannot be turned to a terminal one. We have to use a rule of type 5:

$$(X|Z, X'|\alpha wY) \vdash (X\alpha wY, X'Z).$$

We have started from $Xw\alpha Y$ and we obtained $X\alpha wY$. We can iterate these steps as long as we want, so any circular permutation of the string between X and Y can be produced. Moreover, what we obtain are exactly the circular permutations and nothing more (for instance, at every step we still have one and only one occurrence of B).

To every string XwY we can also apply a rule of type 1, providing w ends with the left hand member of a rule in P. Any rule of P can be simulated in this way, at any place we want in the corresponding sentential form of G, by preparing the string as above, with the help of rules of groups 2–5.

Consequently, for every sentential form w of G there is a string $XBwY$, produced by σ, and, conversely, if Xw_1Bw_2Y is produced by σ, then w_2w_1 is a sentential form of G.

The only way to remove the symbols which are not in T from the strings produced by σ is to use rules in groups 6 and 7. More precisely, the symbols XB can be removed only in the following conditions:

(1) Y is present (hence the work is blocked if we use first rule 7, removing Y: the string cannot participate in any further splicing, and it is not terminal).

(2) The current string bracketed by X, Y and it consists of terminal symbols only.

(3) the symbol B is just after X.

In such a case we can remove XB and after that Y, and what we obtain is a string in T^*. From the previous discussion, it is clear that such a string is in $L(G)$, hence $L(\gamma) \subseteq L(G)$. Conversely, each string in $L(G)$ can be produced in this way, hence $L(G) \subseteq L(\gamma)$. This gives the equality $L(G) = L(\gamma)$, which completes the proof. □

In what follows we shall use many variants of these technique called "rotate-and-simulate". This technique originating from the field of rewriting was adapted by Gh. Păun in [38] for the case of H systems.

4.2 TVDH systems

The approach above is purely mathematic as it uses infinite objects. It is difficult to see for the moment its utility from biological point of view. Below we present another model which permits to have a big computational power while keeping the finite character of the system. In order to do this it suffices to introduce a distribution in the computation as well as a control procedure.

Definition 4.2.1. A *time-varying distributed H system*, or TVDH system, of degree n is the following $n + 3$-tuple

$$D = (V, T, A, R_1, R_2, \dots, R_n),$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet, $A \subseteq V^*$ is a finite set of axioms and R_i, $1 \le i \le n$, called components, are finite sets of splicing rules.

At each moment $k = n \cdot j + i$, where $j \ge 0,\ 1 \le i \le n$, only rules of component R_i are used to splice current words. More precisely, we define

$L_1 = A$,
$L_{k+1} = \sigma_i(L_k)$, for $i \equiv k - 1 \pmod n + 1,\ k \ge 1, 1 \le i \le n,\ \sigma_i = (V, R_i)$.

Indeed, at each step k the current words, L_k, are spliced one time using rules of the component R_i, $i \equiv k-1 \pmod n + 1$ and only the result of this splicing forms the next set of words, L_{k+1}. All other words are eliminated.

We say that the component R_i of a TVDH system rejects the word w, if w cannot enter any rule of R_i. In this case we write $w \uparrow_{\mathrm{R}_i}$. We can omit R_i if the context permits us to do so. In particular, w is rejected by R_i if w cannot match any rule from R_i.

The language generated by a TVDH system D consists of all words over the terminal alphabet produced at some step of computation.

$$L(D) \overset{\text{def}}{=} (\cup_{k \ge 1} L_k) \cap T^*.$$

We denote by VDH_n the family of languages generated by TVDH systems of degree at most n.

Example 4.2.1.

Let us consider the following TVDH system:
$D = (V, T, A, R)$, where $V = T = \{a, b, c\}$, $A = \{cab\}$, $R = \left\{ r = \frac{c \mid a}{a \mid b} \right\}$.

We have $L_1 = \{cab\}$. We can apply r to cab and to itself: $(c|ab, ca|b) \vdash_r (cb, caab)$. This gives us $L_2 = \{cb, caab\}$. Now we can apply r to $caab$ and to itself: $(c|aab, caa|b) \vdash_r (cb, caaaab)$, which gives $L_2 = \{cb, caaaab\}$. We can easily see that $L_k = \{cb, ca^{2^k}b\}$, hence the language generated by D is: $L(D) = \{cb, ca^{2^n}b\},\ n \ge 0$ which is not a context-free language.

4.3 The computational power of TVDH systems

We can see that TVDH systems are quite powerful, even with one component. In this chapter we show that two components suffice in order to reach the computational power of Turing machines. We remark that the same result may be obtained with only one component and we present this proof in Chapter 6, see Theorem 6.3.1.

Theorem 4.3.1. *Let $G = (N, T, P, S)$ be any formal grammar. Then, there is a TVDH system $D_G = (V, T, A, R_1, R_2)$ of degree 2 which simulates G and $L(G) = L(D_G)$.*

Proof.

Definition of the system

We define $D_G = (V, T, A, R_1, R_2)$ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \dots, a_n\}$ ($a_n = B$) and $B \notin N \cup T$.

In what follows we assume that: $1 \le \mathbf{i} \le n$, $1 \le \mathbf{j} \le n-1$, $2 \le \mathbf{k} \le n$ and $\mathbf{a} \in N \cup T \cup \{B\}$.

The alphabet V is defined by $V = N \cup T \cup \{B\} \cup \{X, Y, Z, Z', Z'', X_\mathbf{i}, Y_\mathbf{i}, X'_\mathbf{j}, Y'_\mathbf{j}, X''_\mathbf{j}, Y''_\mathbf{j}, X', Y', X'', Y'', C_1, C_2, D_1, D_2\}$

The terminal alphabet T is the same as for the grammar G.

Axioms are defined by: $A = \{XSBY\} \cup \{ZY_\mathbf{i}, ZY'_\mathbf{j}, ZY''_\mathbf{j}, X'Z, X''Z, ZY, X_\mathbf{i}a_\mathbf{i}Z, X'_\mathbf{j}Z, X''_\mathbf{j}Z, ZY', ZY'', XZ, X_\mathbf{j}Z, C_1Z', D_1, Z''C_2, D_2\} \cup \{ZvY_\mathbf{j}, \exists u \to va_\mathbf{j} \in P\}$.

The components are defined as follows.

Component R_1:

$$1.1: \begin{array}{c|c} \varepsilon & uY \\ \hline Z & vY_\mathbf{j} \end{array}, \quad \exists u \to va_\mathbf{j} \in P; \qquad 1.1': \begin{array}{c|c} \varepsilon & a_\mathbf{i}uY \\ \hline Z & Y_\mathbf{i} \end{array}, \quad \exists u \to \varepsilon \in P;$$

$$1.2: \begin{array}{c|c} \varepsilon & a_\mathbf{i}Y \\ \hline Z & Y_\mathbf{i} \end{array}; \qquad 1.3: \begin{array}{c|c} \mathbf{a} & Y_\mathbf{k} \\ \hline Z & Y'_{\mathbf{k}-1} \end{array}; \qquad 1.4: \begin{array}{c|c} \mathbf{a} & Y'_\mathbf{j} \\ \hline Z & Y''_\mathbf{j} \end{array};$$

$$1.5: \begin{array}{c|c} X_1 & \mathbf{a} \\ \hline X' & Z \end{array}; \qquad 1.6: \begin{array}{c|c} X' & \mathbf{a} \\ \hline X'' & Z \end{array}; \qquad 1.7: \begin{array}{c|c} \mathbf{a} & Y'' \\ \hline Z & Y \end{array};$$

$$1.8: \begin{array}{c|c} \mathbf{a} & BY'' \\ \hline Z'' & \varepsilon \end{array}; \qquad 1.9: \begin{array}{c|c} \mathbf{a} & Y''_\mathbf{j} \\ \hline Z & Y_\mathbf{j} \end{array}; \qquad 1.10: \begin{array}{c|c} C_1 & Z' \\ \hline \varepsilon & D_1 \end{array};$$

Component R_2:

$$2.1: \begin{array}{c|c} X & \mathbf{a} \\ \hline X_\mathbf{i}a_\mathbf{i} & Z \end{array}; \quad 2.2: \begin{array}{c|c} X_\mathbf{k} & \mathbf{a} \\ \hline X'_{\mathbf{k}-1} & Z \end{array}; \quad 2.3: \begin{array}{c|c} X'_\mathbf{j} & \mathbf{a} \\ \hline X''_\mathbf{j} & Z \end{array}; \quad 2.4: \begin{array}{c|c} \mathbf{a} & Y_1 \\ \hline Z & Y' \end{array};$$

$$2.5: \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z & Y'' \end{array}; \quad 2.6: \begin{array}{c|c} X'' & \mathbf{a} \\ \hline X & Z \end{array}; \quad 2.7: \begin{array}{c|c} X'' & \mathbf{a} \\ \hline \varepsilon & Z' \end{array}; \quad 2.8: \begin{array}{c|c} X''_\mathbf{j} & \mathbf{a} \\ \hline X_\mathbf{j} & Z \end{array}; \quad 2.9: \begin{array}{c|c} Z'' & C_2 \\ \hline D_2 & \varepsilon \end{array};$$

Components R_1 and R_2 contain also following rules:

$\begin{array}{c|c} \alpha & \varepsilon \\ \hline \alpha & \varepsilon \end{array}$ for any axiom $\alpha \in A$, except $XSBY$.

We claim that $L(D_G) = L(G)$.

We shall prove this assertion in the following way. First we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(D_G)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. By consequent our assertion will be proved.

Our simulation is based on the "rotate-and-simulate" method, see Section 4.1.2. We use a technique similar to the one used in [23, 40, 44]. We start with the word Xwa_iY in component R_1. After the application of rule 1.2, component R_2 receives XwY_i. In R_2 the rule 2.1 is applied and R_1 receives words $X_ja_jwY_i$ $(1 \leq j \leq n)$. After that point the system works in a cycle where indices i and j decrease simultaneously. Words for which $j \neq i$ are eliminated and only words of type $X_1a_iwY_1$ remain. After that we obtain the word Xa_iwY, so we rotated the word Xwa_iY. If $a_iw = a_iw'B$ and $a_iw' \in T^*$, then this word which belongs to the result is also produced.

The flow-chart of the computation

The computation in D_G follows the flow-chart shown in the Fig. 4.1. The vertices of the flow-chart show a configuration of molecules during the computation. We enumerate all configurations and their numbers are in the upper right corner. In configurations, the symbol **w** is treated as a variable, and it may have different values in different configurations. For example, if in configuration 1 the symbol **w** is equal to $w'a_i$ then in configuration 2 it may have the value w'. We shall show that the computation follows the flow-chart from the Fig. 4.1, *i.e.* all molecules produced in one configuration will be eliminated except molecules from the next configuration.

We note that the rule 1.10 permits to produce the word Z' starting from axioms C_1Z' and D_1, hence this word will appear in the second component. Similarly, the rule 2.9 permits to produce the word Z'' which will appear in the first component. This trick is used in order to avoid an erroneous computation. More details about this may be found in subsection 4.5.1.

Rotation

Let us consider the word Xwa_iY $(w \in (N \cup T \cup \{B\})^*)$, $1 \leq i \leq n$. We are in configuration 1. We shall show now how the rotation of wa_i is performed.

$(Xw|a_iY, Z|Y_i) \vdash_{1.2} (XwY_i, Za_iY \uparrow)$.
The word Za_iY cannot enter any rule of R_2, so it is eliminated.

Now we are in configuration 2.

$(X|wY_i, X_ka_k|Z) \vdash_{2.1} (\mathbf{X_ka_kwY_i}, XZ)$, $1 \leq i, k \leq n$.
The word XZ is an axiom.

If $i = 1$, we may also have the following computation:

$$\begin{tikzcd}
\mathbf{w} & X''w \arrow[l] & X''\mathbf{w}Y''\;{}^{7} \arrow[l] \arrow[dl] & \\
 & X\mathbf{w}Y''\;{}^{8} \arrow[d] & & X''\mathbf{w}Y'\;{}^{6} \arrow[ul] \\
\text{début} \arrow[r, Rightarrow] & X\mathbf{w}Y\;{}^{1} \arrow[d] & & X'\mathbf{w}Y'\;{}^{5} \arrow[u] \\
 & X\mathbf{w}Y_i\;{}^{2} \arrow[dr] & & X'\mathbf{w}Y_i\;{}^{4} \arrow[u] \\
 & & X_j\mathbf{w}Y_i\;{}^{3} \arrow[ur] \arrow[dr] & \\
 & X''_{j-1}\mathbf{w}Y_{i-1}\;{}^{13} \arrow[ur] & & X_j\mathbf{w}Y'_{i-1}\;{}^{9} \arrow[d] \\
 & X''_{j-1}\mathbf{w}Y''_{i-1}\;{}^{12} \arrow[u] & & X'_{j-1}\mathbf{w}Y'_{i-1}\;{}^{10} \arrow[dl] \\
 & & X'_{j-1}\mathbf{w}Y''_{i-1}\;{}^{11} \arrow[ul] &
\end{tikzcd}$$

Figure 4.1: The flow-chart of the computation

$(Xw|Y_1, Z|Y') \vdash_{2.4} (XwY' \uparrow, ZY_1)$.
The word ZY_1 is an axiom and the word XwY' is eliminated.

We are in configuration 3. There are 4 possible cases:

a) $X_j a_s w Y_1$; b) $X_1 a_s w Y_i$; c) $X_j a_s w Y_i$; d) $X_1 a_s w Y_1$; $i, j > 1$, $1 \leq s \leq n$

a) $\mathbf{X_j a_s w Y_1}, j > 1$.

$X_j a_s w Y_1 \uparrow$.

The word $X_j a_s w Y_1$ cannot enter a rule of R_1, therefore it is eliminated.

b) $\mathbf{X_1 a_s w Y_i}, i > 1$.

We have now two possibilities:

(i) $(X_1 a_s w|Y_i, Z|Y'_{i-1}) \vdash_{1.3} (X_1 a_s w Y'_{i-1} \uparrow, ZY_i)$.
The word ZY_i is an axiom. The word $X_1 a_s w Y'_{i-1}$ cannot enter any rule of R_2, so it is eliminated.

(ii) $(X_1|a_s w Y_i, X'|Z) \vdash_{1.5} (X' a_s w Y_i \uparrow, X_1 Z)$.
The word $X_1 Z$ is an axiom. The word $X' a_s w Y_i$ cannot enter any rule of R_2, hence it is eliminated.

In both cases the computation does not produce any new words.

c) $\mathbf{X_j a_s w Y_i}, i, j > 1$.

$(X_j a_s w|Y_i, Z|Y'_{i-1}) \vdash_{1.3} (X_j a_s w Y'_{i-1}, ZY_i)$.

The word ZY_i is an axiom.

We are in configuration 9.
$(X_j|a_swY'_{i-1}, X'_{j-1}|Z) \vdash_{2.2} (X'_{j-1}a_swY'_{i-1}, X_jZ)$.
The word X_jZ is an axiom.

We are in configuration 10.
$(X'_{j-1}a_sw|Y'_{i-1}, Z|Y''_{i-1}) \vdash_{1.4} (X'_{j-1}a_swY''_{i-1}, ZY'_{i-1})$.
The word ZY'_{i-1} is an axiom.

We are in configuration 11.
$(X'_{j-1}|a_swY''_{i-1}, X''_{j-1}|Z) \vdash_{2.3} (X''_{j-1}a_swY''_{i-1}, X'_{j-1}Z)$.
The word $X'_{j-1}Z$ is an axiom.

We are in configuration 12.
$(X''_{j-1}a_sw|Y''_{i-1}, Z|Y_{i-1}) \vdash_{1.9} (X''_{j-1}a_swY_{i-1}, ZY''_{i-1})$.
The word ZY''_{i-1} is an axiom.

We are in configuration 13.
Now there are two possible continuations.
(i) $(X''_{j-1}a_sw|Y_1, Z|Y') \vdash_{2.4} (X''_{j-1}a_swY' \uparrow, ZY_1)$.
The word ZY_1 is an axiom and the word $X''_{j-1}a_swY'$ is eliminated.

(ii) $(X''_{j-1}|a_swY_{i-1}, X_{j-1}|Z) \vdash_{2.8} (\mathbf{X_{j-1}a_swY_{i-1}}, X''_{j-1}Z)$.
The word $X''_{j-1}Z$ is an axiom. We note that we arrived again in configuration 1. It is easy to see that indices of X and Y were decremented simultaneously. We can continue in this way until one of indices will become 1, which represent one of cases (a), (b) or (d).

d) $\mathbf{X_1a_swY_1}$.
$(X_1|a_swY_1, X'|Z) \vdash_{1.5} (X'a_swY_1, X_1Z)$.
The word X_1Z is an axiom.

We are in configuration 4.
$(X'a_sw|Y_1, Z|Y') \vdash_{2.4} (X'a_swY', ZY_1)$.
The word ZY_1 is an axiom.

We are in configuration 5.
$(X'|a_swY', X''|Z) \vdash_{1.6} (X''a_swY', X'Z)$.
The word $X'Z$ is an axiom.
We are in configuration 6.
We have now three possibilities:

(i) $(X''|a_swY', X|Z) \vdash_{2.6} (Xa_swY' \uparrow, X''Z)$.
The word $X''Z$ is an axiom. The word Xa_swY' which cannot enter any rule of R_1 is eliminated.
(ii) $(X''|a_swY', |Z') \vdash_{2.7} (a_swY' \uparrow, X''Z' \uparrow)$.
The words a_swY' and $X''Z'$ cannot enter any rule of R_1, therefore they are eliminated.
(iii) $(X''a_sw|Y', Z|Y'') \vdash_{2.5} (X''a_swY'', ZY')$.
The word ZY' is an axiom.
We are in configuration 7.

(*) $(X''a_sw|Y'', Z|Y) \vdash_{1.7} (X''a_swY, ZY'')$.
The word ZY'' is an axiom.

We are in configuration 8.

(**) $(X''|a_swY, X|Z) \vdash_{2.6} (\mathbf{Xa_swY}, X''Z)$.
The word $X''Z$ is an axiom.

We arrived in configuration 1, so we rotated a_s. It was possible to apply rules 1.8 and 2.7 in cases marked by (*) and (**), but these application will be discussed later.

Simulation of productions of the grammar

If there is a production $u \to va_i$ (or $u \to \varepsilon$) and a word $Xw'uY$ $(Xw''a_iuY)$, then we can apply the rule 1.1 (1.1′).

$(Xw'|uY, Z|vY_i) \vdash_{1.1} (Xw'vY_i, ZuY \uparrow)$.
The word ZuY cannot enter any rule of R_2, so it is eliminated.

$(Xw''|a_iuY, Z|Y_i) \vdash_{1.1'} (Xw''Y_i, Za_iuY \uparrow)$.
The word Za_iuY which cannot enter any rule of R_2 is eliminated.

After that point, the system works as in the case of the rotation. Consequently, we simulate the application of the rule $u \to va_i$ $(u \to \varepsilon)$ of G.

Obtention of the result

If we have the symbol B at the end of the word in the case (*), then we can apply the rule 1.8:

$(X''a_sw'|BY'', Z''|) \vdash_{1.8} (X''a_sw', Z''BY'' \uparrow)$.
The word $Z''BY''$ is eliminated.

$(X''|a_sw', |Z') \vdash_{2.7} (\mathbf{a_sw'} \uparrow, X''Z' \uparrow)$.
The words $X''Z'$ and a_sw' are eliminated. If the word $a_sw' \in T^*$, the a_sw' belongs to the result of the computation.

It was possible to apply the rule 2.6:

$(X''|a_sw', X|Z) \vdash_{2.6} (Xa_sw' \uparrow, X''Z)$.
The word $X''Z$ is an axiom and Xa_sw' is eliminated.

We observe that, by definition, we have at each step an unlimited number of copies of each word. In the case above both applications are made in parallel and one of these applications produces a result of the computation, while the other one does not produce any new words.

Let us consider now the case (**). We can apply the rule 2.7:

$(X''|a_swY, |Z') \vdash_{2.7} (a_swY, X''Z' \uparrow)$.
The word $X''Z'$ is eliminated.

$(a_sw'|a_tY, Z|Y_t) \vdash_{1.1,\ 1.1'\ ou\ 1.2} (a_sw'Y_t, Za_tY \uparrow),\ 1 \le t \le n$.
The word Za_tY is eliminated.

If $t \neq 1$, then the word $a_sw'Y_t$ is eliminated.

If $t = 1$,

$(a_s w'|Y_1, Z|Y') \vdash_{2.4} (a_s w' Y' \uparrow, ZY_1)$.
The word ZY_1 is an axiom and the word $a_s w' Y'$ is eliminated.

Therefore, this computation does not produce any new word.

Other computations with axioms

We may have the following computations between axioms but they do not lead to new words.

$(Zv|Y_i, Z|Y'_{i-1}) \vdash_{1.3} (ZvY'_{i-1} \uparrow, ZY_i)$, $2 \le i \le n$.

$(X_1|a_1 Z, X'|Z) \vdash_{1.5} (X' a_1 Z \uparrow, X_1 Z)$.

$(X_k|a_k Z, X'_{k-1}|Z) \vdash_{2.2} (X'_{k-1} a_k Z \uparrow, X_k Z)$, $2 \le k \le n$.

$(Zv|Y_1, Z|Y') \vdash_{2.4} (ZvY' \uparrow, ZY_1)$.

As we analysed all cases, our proof is complete.

Final remarks

We note that it was possible to use the word $C_1 Z'$ instead of Z' and the word $Z'' C_2$ instead of Z'' in some places of the computation above. But these applications, introducing C_1 and C_2 in the word that is considered do not change the fact of rejection of that word by the next component.

It is easy to see that by following the flow-chart of the computation we generate all words of $L(G)$ and, as we considered all possible cases, it is clear that the system does not produce other words.

□

4.4 A “small” universal TVDH system

We shall show that TVDH systems with one component are not predictable. In order to do this we show that with one component we can simulate any 2 tag system. Tag systems were used in the literature in order to obtain Turing machines of a small size, see [46], that is why the system that we obtained has a very compact description.

We define an *input* for a TVDH system. An input for the system D is simply a word w over the alphabet of D. The computation of D on the input w is done by adding w to axioms and then by evolving D as usual.

Theorem 4.4.1. *Let $T = (2, V, P)$ be a tag system and $w \in V^*$. Then, there is a TVDH system D of degree 1, $D = (V', V, A, R_1)$, which given the word $LXwY$ as input simulates T on input w,* i.e. *such that:*

1. *for any word w on which T halts producing the result w', the system D produce an unique result w'.*

2. *for any word w on which T does not halt, the system D computes infinitely without producing a result.*

Proof.

Let $V = \{a_1, \dots, a_{n+1}\}$ and $P = \{P_1, \dots, P_n\}$.

In what follows we assume that $1 \le i \le n$ and $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \{a_1, \dots, a_n\}$.

We define D as follows.

The alphabet V' is defined by: $V' = V \cup \{X, Y, L, X', Y', R, Z\}$.

Axioms are defined by: $A = \{LXwY, ZP_iY, X'E, RY'\}$.

Rules of R_1 are defined as follows:

Main rules:

$$1: \begin{array}{c|c} LXa_i\mathbf{b} & \mathbf{c} \\ \hline X' & E \end{array}; \quad 2: \begin{array}{c|c} \mathbf{c}Y & \varepsilon \\ \hline L & Xa_i\mathbf{b}E \end{array}; \quad 3: \begin{array}{c|c} \mathbf{a} & YXa_i \\ \hline Z & P_iY \end{array}; \quad 4: \begin{array}{c|c} X' & \mathbf{a} \\ \hline LX & E \end{array};$$

Additional rules:

$$5: \begin{array}{c|c} LX & a_i\mathbf{b}E \\ \hline LXa_i & \mathbf{b}E \end{array}; \quad 6: \begin{array}{c|c} LX & \mathbf{b}E \\ \hline LX\mathbf{b} & E \end{array}; \quad 7: \begin{array}{c|c} ZP_i & Y \\ \hline R & Y' \end{array}; \quad 8: \begin{array}{c|c} ZP_i & Y' \\ \hline R & Y \end{array};$$

Rules for the result:

$$9: \begin{array}{c|c} LXa_{n+1}\mathbf{b} & \mathbf{c} \\ \hline \varepsilon & X'E \end{array}; \qquad 10: \begin{array}{c|c} \mathbf{c} & Y \\ \hline LXa_{n+1}\mathbf{b}X'E & \varepsilon \end{array};$$

Now we shall show how we simulate the computation of T. In our system we have a word of the form $LXwY$ which corresponds to the working word w in T. First, we cut off the first two symbols of w and we mark by X' the beginning of the word. This gives us two molecules: Xa_ia_jE and $X'w'Y$, where $w = a_ia_jw'$. After that we glue the first molecule to the end of the second one obtaining $X'w'YXa_ia_jE$. Now we can replace the end of this word by a right production which gives us the word $X'w'P_iY$. Finally, we take off the prime from X obtaining $LXw'P_iY$. We can iterate this process that simulates the application of productions of T. We halt when the first symbol is a_{n+1}. In this case we take off the brackets LX and Y and we obtain the result.

In what follows we give more details by considering all possible evolutions.

Step 1.

$$\begin{array}{c|c} LXa_ib & wY \\ \hline X' & E \end{array} \vdash_1 \frac{X'wY}{LXa_ibE},$$

$$\begin{array}{c|c} ZP_i & Y \\ \hline R & Y' \end{array} \vdash_7 \frac{ZP_iY'}{RY}.$$

Step 2.

$$\begin{array}{c|c} X'wY & \varepsilon \\ \hline L & Xa_ibE \end{array} \vdash_2 \frac{X'wYXa_ibE}{L\uparrow},$$

$$\begin{array}{c|c} LX & a_ibE \\ \hline LXa_i & bE \end{array} \vdash_5 \frac{LXbE}{LXa_ia_ibE\uparrow},$$

$$\begin{array}{c|c} ZP_i & Y' \\ \hline R & Y \end{array} \vdash_8 \frac{ZP_iY}{RY'}\,.$$

Step 3.

$$\begin{array}{c|c} X'w & YXa_ibE \\ \hline Z & P_iY \end{array} \vdash_3 \frac{X'wP_iY}{ZYXa_ibE\uparrow}\,,$$

$$\begin{array}{c|c} LX & bE \\ \hline LXb & E \end{array} \vdash_6 \frac{LXE}{LXbbE\uparrow}\,,$$

$$\begin{array}{c|c} ZP_i & Y \\ \hline R & Y' \end{array} \vdash_7 \frac{ZP_iY'}{RY}\,.$$

Step 4.

$$\begin{array}{c|c} X' & wP_iY \\ \hline LX & E \end{array} \vdash_4 \frac{LXwP_iY}{X'E}\,,$$

$$\begin{array}{c|c} ZP_i & Y' \\ \hline R & Y \end{array} \vdash_8 \frac{ZP_iY}{RY'}\,.$$

We note that we have simulated one step of computation in T and that all non-desirable molecules were eliminated. We can continue this process until the first symbol of w became a_{n+1}. Since we need four steps in order to simulate one step in T, this situation arrives when we are at the step number $4k+1$, $k \geq 0$. In this case we can take off the brackets LX and Y obtaining the resulting word:

Step 4k+1.

$$\begin{array}{c|c} LXa_{n+1}b & wY \\ \hline \varepsilon & X'E \end{array} \vdash_9 \frac{wY}{LXa_{n+1}bX'E}\,.$$

Step 4k+2.

$$\begin{array}{c|c} w & Y \\ \hline LXa_{n+1}bX'E & \varepsilon \end{array} \vdash_{10} \frac{w\uparrow}{LXa_{n+1}bX'EY\uparrow}\,.$$

Since we do not have any word of form $LXwY$, the further computation will consist in application of rules 7 and 8 without producing a terminal word.

□

4.5 Correction of existent systems

As we said before in Chapter 2, we used a program which permits to simulate TVDH systems and which was developed during the master thesis of the author. We checked several articles on TVDH systems which were published and we found that some of them contain errors. In this section we present the corresponding systems and we show the errors that they contain as well as corrections that are necessary for a correct functioning. We found that some errors are common to several authors, that is why we hope that our correction will permit to avoid them in the future.

Each system which is considered simulates a type-0 grammar $G=(V,T,S,P)$.

4.5.1 Correction of the system TVDH3

We start by the TVDH system presented in [23]. The Fig. 4.2 contains the definition of this system which we shall call TVDH3.

Consider the system TVDH3 $= (V, T, A, R_1, R_2, R_3)$.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(a_n = B)$ and $B \notin N \cup T$.
Let i, j and k such that $1 \leq i \leq n$, $1 \leq j \leq n-1$, $2 \leq k \leq n$.

The alphabet V is defined by $V = N \cup T \cup \{B\} \cup \{X, Y, Z, X_i, Y_i, X_j', Y_j', X_j'', Y_j'', X', Y', Z', Z''\}$.

Axioms A are defined by $A = \{XSBY\} \cup$ $\{ZvY_i, \exists u : u \to va_i \in P,\ i \in \{1, \ldots, n-1\}\}$ $\cup$ $\{ZY_i, X_j'Z, X'Z,\ X_ia_iZ,\ ZY_j'',\ X_jZ,\ XZ,\ ZY_j',\ ZY',\ X_j''Z,\ ZY, Z', Z''$.

Component R_1:

1.1 : $\begin{array}{c|c} \varepsilon & uY \\ \hline Z & vY_j \end{array}$, $\exists u \to va_j \in P$; 1.2 : $\begin{array}{c|c} \varepsilon & a_iY \\ \hline Z & Y_i \end{array}$; 1.3 : $\begin{array}{c|c} X_k & \varepsilon \\ \hline X_{k-1}' & Z \end{array}$;

1.4 : $\begin{array}{c|c} X_1 & \varepsilon \\ \hline X' & Z \end{array}$; 1.5 : $\begin{array}{c|c} \varepsilon & Y_i'' \\ \hline Z & Y_i \end{array}$; 1.6 : $\begin{array}{c|c} \varepsilon & a_iuY \\ \hline Z & Y_i \end{array}$, $\exists u \to \varepsilon \in P$;

Component R_2:

2.1 : $\begin{array}{c|c} X & \varepsilon \\ \hline X_ia_i & Z \end{array}$; 2.2 : $\begin{array}{c|c} \varepsilon & Y_j' \\ \hline Z & Y_j'' \end{array}$; 2.3 : $\begin{array}{c|c} X_j'' & \varepsilon \\ \hline X_j & Z \end{array}$;

2.4 : $\begin{array}{c|c} X' & \varepsilon \\ \hline X & Z \end{array}$; 2.5 : $\begin{array}{c|c} X' & \varepsilon \\ \hline \varepsilon & Z' \end{array}$;

Component R_3:

3.1 : $\begin{array}{c|c} \varepsilon & Y_k \\ \hline Z & Y_{k-1}' \end{array}$; 3.2 : $\begin{array}{c|c} X_j' & \varepsilon \\ \hline X_j'' & Z \end{array}$; 3.3 : $\begin{array}{c|c} \varepsilon & Y_1 \\ \hline Z & Y' \end{array}$;

3.4 : $\begin{array}{c|c} \varepsilon & Y' \\ \hline Z & Y \end{array}$; 3.5 : $\begin{array}{c|c} \varepsilon & BY' \\ \hline Z'' & \varepsilon \end{array}$;

Note.
Components R_1, R_2 and R_3 contain also the following rules:
$\begin{array}{c|c} \alpha & \varepsilon \\ \hline \alpha & \varepsilon \end{array}$ for each axiom $\alpha \in A$, except $XSBY$.

Figure 4.2: The system TVDH3

The functioning of the system TVDH3 is not correct. The authors did not observe the following situation: the application of the rule 2.5 produce the word $X'Z'$ which is not an axiom and which is not eliminated from the system due to the rule $Z'\#\varepsilon\$Z'\#\varepsilon$. After that, by using the rule 2.4, we obtain XZ'. This word used in the rule 2.1 produces the words X_ia_iZ' and X_ma_iZ', where $m \neq i$. Finally, the rule 2.5 permits to introduce these prefixes in any word. More precisely:

$(X'|wY', |Z') \vdash_{2.5} (X'Z', wY')$.

$X'Z'$ is always present due to the rule $Z'\#\varepsilon\$Z'\#\varepsilon$.

$(X'|Z', X|Z) \vdash_{2.4} (XZ', X'Z)$.

Now, the rule 2.1 is applied to XZ' which is always present and to one of words X_ia_iZ:

$(X|Z', X_ia_i|Z) \vdash_{2.1} (XZ, X_ia_iZ')$.

The words $X_i a_i Z'$ are always present. Starting from them we can obtain the words $X'_{i-1} a_i Z'$:

$(X_i|a_iZ', X'_{i-1}|Z) \vdash_{1.3} (X_iZ, X'_{i-1}a_iZ')$.

If we continue, we can obtain consecutively $X''_{i-1}a_iZ'$, $X_{i-1}a_iZ'$, and finally $X'a_iZ'$ and Xa_iZ'.

Now, let $X'wY'$ be a correct evolution. Then the rule 2.5 of the second component permits to insert a_i in front of w:

$(X'|wY', Xa_i|Z') \vdash_{2.5} (X'Z', Xa_iwY')$,

which is not a correct evolution.

We see that the problem is caused by the word XZ'. In order to solve it, it is enough to replace the rule 2.4 by the following set of rules:

$\dfrac{X' \mid a_i}{X \mid Z}$, $1 \leq i \leq n$.

In this case the word XZ' cannot be produced and the system have a correct behaviour.

We present below another solution to the problem above and we shall use this solution in the future. It concerns words that end by Z', *i.e.* words of form wZ'. We note that the wrong functioning of the system TVDH3 is due to the fact that these words once produced will remain forever because of the rule $Z'\#\varepsilon\$Z'\#\varepsilon$. We can replace this rule by the following set of rules:

$\dfrac{C \mid Z'}{D \mid \varepsilon} \in R_1$ and $\left\{ \dfrac{CZ' \mid \varepsilon}{CZ' \mid \varepsilon}, \dfrac{D \mid \varepsilon}{D \mid \varepsilon} \right\} \in R_1 \cap R_2 \cap R_3$.

We also add CZ' and D to axioms.

In this case, only CZ' is persistent while all other words which end in Z' are eliminated. This solves our problem.

We note that in the system from Theorem 4.3.1 this second approach was used in order to avoid the same problem as the one discussed in this section.

4.5.2 Correction of the system TVDH4

We continue with the system presented in [35] which we shall call TVDH4. Its definition is given in the Fig. 4.3.

The functioning of the system TVDH4 is not correct. There are three errors:

1) An error of the same type as in the previous example: the rule 4.2 permits to obtain the word $X_0Z'_0$ which is not eliminated. After that, we can obtain as above XZ'_0, $X_iv_iZ'_0$, $Xv_iZ'_0$, etc. Let us consider now a correct evolution X_0wY. By using the rule 4.2 we insert v_i in front of w:

$(X_0|wY, Xv_i|Z'_0) \vdash_{4.2} (X_0Z'_0, Xv_iwY)$,

which is not a correct evolution.

In order to correct this problem the rule 4.2 must be replaced by the set of rules $X_0\#\alpha_i\$X\#Z'$, $1 \leq i \leq n$.

2) A repeated application of the rule 4.4 produces $X_kv_jZ'_j$, $k < j$ which become persistent. Now, the rule 4.3 permits to insert X_ka_j, $j \neq k$ in front of a word which leads to an erroneous rotation.

We suppose that G is in Kuroda normal form. Let $N \cup T \cup \{F\} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ $(\alpha_n = F)$ and $F \notin \{N \cup T\}$. Let i, j and s such that $1 \le i \le l$, $m+1 \le j \le l$ et $1 \le s \le m$.
Let $u_k \to v_k \in P$, $n+1 \le k \le m$ be the rules without context of P, and let $u_k \to v_k \in P$, $m+1 \le k \le l$ the other rules of P. We denote $u_k = v_k = \alpha_k$ for $1 \le k \le n$.
Let us consider the system TVDH4 $= (V, T, A, R_1, R_2, R_3, R_4)$.
where
$V = N \cup T \cup \{X, Y, Z, Z', Z_0, Z_0', F, X_0, Y_0, X_i, Y_i, Y_j', Z_i'\}$,
$A = \{XSFY, XZ', ZY, Z_0, Z_0', X_0Z', ZY_0, X_iZ', ZY_i, X_iv_iZ_i', ZY_j'\}$,

$$R_1 = Q \cup \left\{ 1.1 : \frac{\varepsilon \mid Y}{\varepsilon \mid Y},\ 1.2 : \frac{X_0 \mid \varepsilon}{X_0 \mid \varepsilon},\ 1.3 : \frac{\varepsilon \mid Y_i}{\varepsilon \mid Y_i},\ 1.4 : \frac{\varepsilon \mid Y_j'}{\varepsilon \mid Y_j'} \right\},$$

$$R_2 = Q \cup \left\{ 2.1 : \frac{X \mid \varepsilon}{X \mid \varepsilon},\ 2.2 : \frac{\varepsilon \mid Y_0}{\varepsilon \mid Y_0},\ 2.3 : \frac{X_i \mid \varepsilon}{X_i \mid \varepsilon} \right\},$$

$$R_3 = Q \cup \left\{ 3.1 : \frac{\varepsilon \mid Y_0}{Z \mid Y},\ 3.2 : \frac{\varepsilon \mid FY_0}{Z_0 \mid \varepsilon},\ 3.3 : \frac{\varepsilon \mid u_sY}{Z \mid Y_s},\ 3.4 : \frac{\varepsilon \mid Y_i}{Z \mid Y_{i-1}} \right\}$$
$$\cup \left\{ 3.5 : \frac{\varepsilon \mid EY}{Z \mid Y_j'},\ 3.6 : \frac{\varepsilon \mid DY_j'}{Z \mid Y_{j-1}} \text{ où } u_j = DE,\ D, E \in N \right\},$$

$$R_4 = Q \cup \left\{ 4.1 : \frac{X_0 \mid \varepsilon}{X \mid Z'},\ 4.2 : \frac{X_0 \mid \varepsilon}{\varepsilon \mid Z_0'},\ 4.3 : \frac{X \mid \varepsilon}{\varepsilon \mid Z_i'},\ 4.4 : \frac{X_i \mid \varepsilon}{X_{i-1} \mid Z'} \right\},$$

et

$$Q = \left\{ z_1 : \frac{Z \mid \varepsilon}{Z \mid \varepsilon},\ z_2 : \frac{\varepsilon \mid Z'}{\varepsilon \mid Z'},\ z_3 : \frac{Z_0 \mid \varepsilon}{Z_0 \mid \varepsilon},\ z_4 : \frac{\varepsilon \mid Z_0'}{\varepsilon \mid Z_0'},\ z_5 : \frac{\varepsilon \mid Z_i'}{\varepsilon \mid Z_i'} \right\}.$$

Figure 4.3: The system TVDH4

In order to solve this problem it suffices to replace rules 4.3 by $X\#\varepsilon\$X_jv_j\#Z_j'$.

3) In this case we have a problem which is similar to case (1), but at the other end of the word. The rule 3.2 produces the word Z_0FY_0 which exists forever. Once produced this word introduces errors in the computation:

$(X_1w|FY_0, Z_0|FY_0) \vdash_{3.2} (X_1wFY_0, Z_0FY_0)$.

$(X_1|wFY_0, X_0|Z') \vdash_{4.4} (X_0wFY_0, X_1Z')$.

After that X_0wFY_0 arrive without any change in the third component, *i.e.* the erroneous word X_1wFY_0 become X_0wFY_0 which is a correct word.

We did not found a solution to this problem, therefore a complete rewriting of the system is necessary.

4.5.3 Correction of the system TVDH7

We continue with the system presented in [44] which we shall call TVDH7. Its definition is given in the Fig. 4.4.

The functioning of the system TVDH7 is not correct. It has the following errors:

1) It is necessary to add ZY_0, ZY_0' et X_0Z to axioms.

2) The problem of the rule $z.1$. The utilisation of this rule permits to produce erroneous words:

$(X_j\alpha_jZ|, Z|Z) \vdash_{z.1} (X_j\alpha_jZZ, Z)$,

Let $N \cup T = \{\alpha_1, \dots, \alpha_{n-1}\}, n \geq 3$, and $P = \{u_i \to v_i \mid 1 \leq i \leq m\}$. Let $\alpha_n = B$ a new symbol. Let i and j such that $1 \leq i \leq m$ et $1 \leq j \leq n$.
We consider TVDH7 $= (V, T, A, R_1, \dots, R_7)$, where

$$\begin{aligned} V &= N \cup T \cup \{X, Y, Y', Z, B, Y_0, X_0, Y_0', Y_j, Y_j', X_j\}, \\ A &= \{XBSY, ZY, ZY', ZZ, Zv_iY, ZY_j, ZY_j', X_j\alpha_jZ, X_jZ\}, \end{aligned}$$

$R_1 = \{1.1\colon \varepsilon\#u_iY\$Z\#v_iY,\ 1.2\colon \varepsilon\#Y\$Z\#Y,\ 1.3\colon \varepsilon\#Y_j\$Z\#Y_j\}$,
$R_2 = \{2.1\colon \varepsilon\#\alpha_jY\$Z\#Y_j,\ 2.2\colon \varepsilon\#Y\$Z\#Y',\ 2.3\colon \varepsilon\#Y_j\$Z\#Y_j'\}$,
$R_3 = \{3.1\colon X\#\varepsilon\$X_j\alpha_j\#Z,\ 3.2\colon \varepsilon\#Y'\$Z\#Y,\ 3.3\colon \varepsilon\#Y_j'\$Z\#Y_j\}$,
$R_4 = \{4.1\colon \varepsilon\#Y_j\$Z\#Y_{j-1},\ 4.2\colon \varepsilon\#Y\$Z\#Y\}$,
$R_5 = \{5.1\colon X_j\#\varepsilon\$X_{j-1}\#Z,\ 5.2\colon \varepsilon\#Y\$Z\#Y\}$,
$R_6 = \{6.1\colon \varepsilon\#Y_0\$Z\#Y,\ 6.2\colon \varepsilon\#Y_0\$ZZ\#\varepsilon,\ 6.3\colon \varepsilon\#Y\$Z\#Y',\ 6.4\colon \varepsilon\#Y_j\$Z\#Y_j'\}$,
$R_7 = \{7.1\colon X_0\#\varepsilon\$X\#Z,\ 7.2\colon X_0B\#\$\#ZZ,\ 7.3\colon \varepsilon\#Y'\$Z\#Y,\ 7.4\colon \varepsilon\#Y_j'\$Z\#Y_j\}$.

All component contain also the rule $z.1 : Z\#\varepsilon\$Z\#\varepsilon$.

Figure 4.4: The system TVDH7

$(X_j\alpha_jZZ|, Z|Y_k) \vdash_{z.1} (X_j\alpha_jZZY_k, Z)$.

In this way we can obtain all possible words of type $X_i\alpha_jZZY_k$.

Finally, we have the following application which leads to an error:

$(X_0B|wY, X\alpha_i|ZZY_k) \vdash_{7.2} (X\alpha_iwY, X_0BZZY_k)$.

A similar error is caused by the rule 6.2, but at another end of the word.

We can try to solve this problem by replacing ZZ by a new symbol Z', but this creates a new problem, similar to the problem of TVDH3. In this case we produce by the rule 7.2 the word X_0BZ' which is transformed in $X_i\alpha_iBZ'$ which being used with X_0BwY_i in the rule 7.2 leads to an erroneous word. We can solve this second problem by the second method of resolution which was shown for TVDH3. In this case we place corresponding rules in component 6.

4.6 Conclusions

The TVDH systems introduced in this chapter have a lot of interesting properties. The elimination strategy gives them a powerful control which permits to manipulate them easily and which makes them very simple. This is why these systems are important, since their simple structure gives the possibility to simulate them by other systems based on splicing. We shall give in Chapters 6 and 9 several examples of systems that simulate TVDH systems.

The obtained systems were checked by the computer simulator `TVDHsim` which was developed during the author's master thesis. This software permits to simulate a TVDH system step by step. The use of this program permitted to find errors in systems presented in the Section 4.5.

In the next chapter we shall consider systems which are similar to systems that we examined in this chapter and we shall show that it is possible to reuse some ideas in this new framework.

Chapter 5

Enhanced time-varying distributed H systems

Time-varying distributed H systems presented in the previous chapter are based on the following idea: words of the current set are spliced only once and only the result of this splicing forms the next set of words. For the moment this single splicing is almost impossible to realise in practice. It is more natural from a biological point of view to suppose that at each step we can splice words any number of times as it is done in the case of H systems, see Section 3.1. This idea was considered by M. Margenstern et Yu. Rogozhin in [23] where enhanced time-varying distributed H systems, or ETVDH systems are introduced. We give in this chapter a formal definition of these systems and describe their computational power.

5.1 Formal definition

Let $\sigma = (V, R)$ be an H scheme. We define the operation $\tilde{\sigma}$ [23]:

$\tilde{\sigma}(L) = \tilde{\sigma}(L' \cup L'') \overset{\text{def}}{=} \sigma^*(L')$, where

$L' = \{w_1 \in L | \exists w_2 \in L : \exists w, w' \in V^* : \exists r \in R : (w_1, w_2) \vdash_r (w, w') \text{ or } (w_2, w_1) \vdash_r (w, w')\}$, $L'' = L \setminus L'$.

More exactly, to obtain $\tilde{\sigma}(L)$ we take all words from L which can participate in splicing, *i.e.* L', and we apply σ^* to them. As a result, we get all possible iterative splicings of these words as well as initial words from L.

Definition 5.1.1. An *enhanced time-varying distributed H system* of degree n is the following $n + 3$-tuple

$$E = (V, T, A, R_1, R_2, \dots, R_n),$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet, $A \subseteq V^*$ is a finite set of axioms and R_i, $1 \leq i \leq n$ called the components are finite sets of splicing rules.

At each moment $k = n \cdot j + i$, where $j \geq 0,\ 1 \leq i \leq n$, only the rules of the component R_i are used for splicing. More exactly, we define

$$L_1 = A,$$
$$L_{k+1} = \tilde{\sigma_i}(L_k), \text{ for } i \equiv k-1 \pmod{n} + 1,\ k \geq 1, 1 \leq i \leq n,\ \sigma_i = (V, R_i).$$

This means for words that at each step k the component R_i, $i \equiv k-1 \pmod{n} + 1$, firstly applies a filter on the current set of words, *i.e.* it eliminates the words that cannot participate in splicing using rules from R_i, and after that the system works as the corresponding H system.

We say that the component R_i of an ETVDH system rejects the word w if this word cannot enter any rule from R_i. We write in such case $w \uparrow_{\mathrm{R}_i}$. We can omit R_i if the context permits us. In particular, the word w is rejected by R_i if it cannot match any rule from R_i.

The language generated by an ETVDH system E consists of all words over the terminal alphabet produced at some step of computation.

$$L(E) \overset{\text{def}}{=} (\cup_{k \geq 1} L_k) \cap T^*.$$

We denote by $EVDH_n$ the family of languages generated by ETVDH systems of degree at most n.

Example 5.1.1.

We reconsider the example 4.2.1 in the framework of ETVDH systems.
$E = (V, T, A, R)$, where $V = T = \{a, b, c\}$, $A = \{cab\}$, $R = \left\{ r = \begin{array}{c|c} c & a \\ \hline a & b \end{array} \right\}$.
We have $L_1 = \{cab\}$. We can apply the rule r to cab and itself: $(c|ab, ca|b) \vdash_r (cb, caab)$. This gives $L_2 = \{cb, cab, caab\}$. Now we can apply r to $caab$ and itself: $(c|aab, caa|b) \vdash_r (cb, caaaab)$ as well as to cab and $caab$: $(c|ab, caa|b) \vdash_r (cb, caaab)$. This gives $L_2 = \{cb, cab, caab, caaab, caaaab\}$. We can see easily that $L_k = \{cb, cab, \ldots, ca^k b\}$. Therefore the language generated by E is the following: $L(D) = ca^n b,\ n \geq 0$ which is a regular language.

5.2 The generative power of ETVDH systems

In this section we shall examine the generative power of systems introduced above. Now one component is not sufficient to produce all recursively enumerable languages. More exactly, we have the following result:

Theorem 5.2.1. *$EVDH_1 = REG$.*

Proof. Let $E = (V, T, A, R)$ be an ETVDH system. We have by definition that $L_1 = A$. Let A' be the set of words that can enter a splicing rule from R. In this case $L_2 = \sigma^*(A')$. Let L_2' be the words from L_2 that can enter a rule from R. Then $L_3 = \sigma^*(L_2')$. It is easy to observe that $L_2 = L_3$. Indeed, L_2 is the smallest closure of A' with respect to the splicing operation. As $A' \subseteq L_2' \subseteq L_2 = \sigma^*(A')$, we obtain that $\sigma^*(L_2') = L_2$. Similarly we obtain that $L_k = L_2$, $k > 2$. Therefore, the language generated by E is $L(E) = \sigma^*(A') \cap T^*$. As $\sigma^*(A')$ is a regular language and the family of regular languages is closed with respect to intersection we obtain that $L(E) \subseteq REG$. The inverse inclusion also holds, see Theorem 4.1.1. $\square$

We observe that ETVDH systems with one component are very similar to extended H systems. The following example shows the difference between these two models. It consists in the application of the filtering rule.

Example 5.2.1.

Let us consider the ETVDH system $E = (V, T, A, R)$, where $V = T = \{c, a, b\}$, $A = \{cab, caab\}$ and $R = \left\{ \frac{c \mid aa}{aa \mid b}, \frac{c \mid ab}{aaa \mid b} \right\}$. Let us consider also the extended H system $EH = (V, T, A, R)$. It is easy to see that $L(EH) = ca^n b$, $n \geq 0, n \neq 3$. In the case of the system E, the word cab is eliminated firstly as it cannot participate in a splicing. Therefore, the language produced by this system is $L(E) = \sigma^*(\{caab\}) = ca^{2n}b$, $n \geq 0$.

Now we show that ETVDH systems have the same power as Turing machines.

Theorem 5.2.2. *For any type-0 grammar $G = (N, T, P, S)$ there is an ETVDH system $E_G = (V, T, A, R_1, R_2, R_3)$ of degree 3 which simulates G and $L(G) = L(E_G)$.*

Proof.

Formal definition of the system

We define $E_G = (V, T, A, R_1, R_2, R_3)$ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(B = a_n)$ and $B \notin N \cup T$.

We assume below that $1 \leq i \leq n$, $1 \leq j \leq n-1$, $2 \leq l \leq n$, $1 \leq k \leq 5, \mathbf{a} \in N \cup T \cup \{B\}$.

The alphabet V is defined by $V = N \cup T \cup \{B\} \cup \{X, Y, X_i, Y_i, X_i', Y_i', X_j'', Y_j'', X', Y', X'', Y'', X''', Y''', Y^{IV}, Z, Z_E, C, C_1, C_2, D, D_1, D_2, Z_E^k\}$.

The terminal alphabet T is the same as for the grammar G.

Axioms are defined by: $A = \{XSBY, X_i'Z_E^3, Z_E^4Y_i', ZY_i, X''Z, ZY^{IV}, X_ia_iZ, ZY_j'', X'Z, ZY', X_jZ, ZY, ZY''', XZ, X_j''Z, ZY'', X'''Z, ZY, CZ_E^k, CZ_E, Z_E^kD, Z_ED, C_2Z'', D_2, Z'C_1, D_1\} \cup \{ZvY : \exists u \to v \in P\}$.

We define the components of the system as follows.

Component R_1:

1.1 : $\frac{\varepsilon \mid uY}{Z \mid vY}$, $\exists u \to v \in P$; 1.2 : $\frac{\varepsilon \mid a_iY}{Z \mid Y_i}$; 1.3 : $\frac{X_i \mid \mathbf{a}}{X_i' \mid Z_E}$;

1.4 : $\frac{\mathbf{a} \mid Y_j''}{Z \mid Y_j}$; 1.5 : $\frac{X' \mid \mathbf{a}}{X'' \mid Z}$; 1.6 : $\frac{\mathbf{a} \mid Y'''}{Z \mid Y^{IV}}$;

1.7 : $\frac{X_i' \mid Z_E}{C \mid Z_E^1}$; 1.8 : $\frac{X_i' \mid Z_E^3}{C \mid Z_E^4}$; 1.9 : $\frac{Z_E^1 \mid Y_i'}{Z_E^2 \mid D}$;

1.10 : $\frac{Z_E^4 \mid Y_i'}{Z_E^5 \mid D}$; 1.11 : $\frac{Z' \mid C_1}{D_1 \mid \varepsilon}$;

Component R_2:

$$2.1: \begin{array}{c|c} X & \mathbf{a} \\ \hline X_i a_i & Z \end{array}; \quad 2.2: \begin{array}{c|c} \mathbf{a} & Y'_l \\ \hline Z & Y''_{l-1} \end{array}; \quad 2.3: \begin{array}{c|c} X'_1 & \mathbf{a} \\ \hline X' & Z \end{array}; \quad 2.4: \begin{array}{c|c} \mathbf{a} & Y'_1 \\ \hline Z & Y' \end{array};$$

$$2.5: \begin{array}{c|c} X''_j & \mathbf{a} \\ \hline X_j & Z \end{array}; \quad 2.6: \begin{array}{c|c} \mathbf{a} & Y'' \\ \hline Z & Y''' \end{array}; \quad 2.7: \begin{array}{c|c} \mathbf{a} & BY'' \\ \hline Z' & \varepsilon \end{array}; \quad 2.8: \begin{array}{c|c} X''' & \mathbf{a} \\ \hline X & Z \end{array};$$

$$2.9: \begin{array}{c|c} X'_i & Z^1_E \\ \hline C & Z^2_E \end{array}; \quad 2.10: \begin{array}{c|c} X'_i & Z^4_E \\ \hline C & Z^5_E \end{array};$$

$$2.11: \begin{array}{c|c} Z^2_E & Y'_i \\ \hline Z^3_E & D \end{array}; \quad 2.12: \begin{array}{c|c} Z^5_E & Y'_i \\ \hline Z_E & D \end{array}; \quad 2.13: \begin{array}{c|c} C_2 & Z'' \\ \hline \varepsilon & D_2 \end{array};$$

Component R_3:

$$3.1: \begin{array}{c|c} \mathbf{a} & Y_i \\ \hline Z_E & Y'_i \end{array}; \quad 3.2: \begin{array}{c|c} X'_l & \mathbf{a} \\ \hline X''_{l-1} & Z \end{array}; \quad 3.3: \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z & Y'' \end{array}; \quad 3.4: \begin{array}{c|c} X'' & \mathbf{a} \\ \hline X''' & Z \end{array};$$

$$3.5: \begin{array}{c|c} X'' & \mathbf{a} \\ \hline \varepsilon & Z'' \end{array}; \quad 3.6: \begin{array}{c|c} \mathbf{a} & Y^{IV} \\ \hline Z & Y \end{array};$$

$$3.7: \begin{array}{c|c} X'_i & Z^2_E \\ \hline C & Z^3_E \end{array}; \quad 3.8: \begin{array}{c|c} X'_i & Z^5_E \\ \hline C & Z_E \end{array}; \quad 3.9: \begin{array}{c|c} Z_E & Y'_i \\ \hline Z^1_E & D \end{array}; \quad 3.10: \begin{array}{c|c} Z^3_E & Y'_i \\ \hline Z^4_E & D \end{array};$$

The components R_1, R_2 and R_3 contain also the following rules:
$\begin{array}{c|c} \alpha & \varepsilon \\ \hline \alpha & \varepsilon \end{array}$ for any axiom $\alpha \in A$, except $XSBY$, $X_iZ^3_E$ et $Z^4_EY_i$.

We affirm that $L(E_G) = L(G)$.

We shall prove this assertion in the following way. Firstly we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(E_G)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. Consequently, our assertion will be proved.

Notations

We shall use the following notation:

$$\begin{array}{c|c} w_1 & w_2 \\ \hline w'_1 & w'_2 \end{array} \quad \vdash_r \quad \frac{w_1 w_2'^{\,*}}{w'_1 w_2}, \quad w_1 w_2$$

where $*$ indicates a possible occurrence of $\uparrow$. The application of the rule r on w_1w_2 and $w'_1w'_2$ is shown in the left side of the formula. The right side contains resulting words $w_1w'_2$ and w'_1w_2. We shall also use the following convention: the upper part of both sides will contain the words in which we are interested, while the lower part will contain either an axiom, or a word containing Z with indices, which does not alter the computation. The symbol $\uparrow$ indicates the rejection of the considered molecule by the next component. The optional term of the right side in the formula, w_1w_2, is omitted if that molecule, which in principle enters the next component, is rejected by the next component. In the opposite case it is written.

Our simulation is based on the "rotate-and-simulate" method, see Section 4.1.2. We use a technique similar to the one used in [23, 40, 44], see also Theorem 4.3.1. We start with the word Xwa_iY in the component R_1. After the application of rule 1.2 component R_2 receives XwY_i. In R_2, the rule 2.1 is applied, and R_3 receives words $X_ja_jwY_i$ $(1 \leq j \leq n)$. After that point the system works in a cycle where indices i and j decrease simultaneously. The words for which $j \neq i$ are eliminated and only words of type $X_1a_iwY_1$ remain. After that we obtain the word Xa_iwY, therefore we rotated the word Xwa_iY. If $a_iw = a_iw'B$ and $a_iw' \in T^*$, this word which belongs to the result is also produced.

The system is constructed in a such way that the words $X_i'Z_E$ appear in the first component only during an even step of computation, *i.e.* if we have these molecules in the first component, the next time when we will be in this component (after 3 steps) these molecules $(X_i'Z_E)$ will not exist. This is a very important property of our system that permits a correct simulation to be performed.

This property is implemented in the following way. We have in the system a word $X_i'Z_E^p$ and at each step we increment p modulo 5 by using rules 1.7, 1.8, 2.9, 2.10, 3.7 and 3.8. When $p = 0$, we obtain $X_i'Z_E$. We start with $X_i'Z_E^3$ and this permits to have the word $X_i'Z_E$ in the first tube only during even steps of computation.

A similar thing happens to the words Z_EY_i'. They appear in the third component only during an odd step of computation.

The word Z' is produced in the first component and it appears in the second component only. The word Z'' is produced in the second component and it appears in the third component only.

The motivation of above properties is the same as for the method of directing molecules and it can be found in Chapter 6 where we describe it more systematically.

The flow-chart of the computation

The computation follows the flow-chart shown in the Fig. 5.1. The vertices of the flow-chart show a configuration of molecules during the computation. We enumerate all configurations and their numbers are in the upper right corner. Inside configurations, the symbol $\mathbf{w}$ is treated as a variable, and it may have different values in different configurations. E.g., if in configuration 14 the symbol $\mathbf{w}$ is equal to $w'a_i$ then in configuration 15 it may have the value w'. We shall show that the computation follows the flow-chart from the Fig. 5.1, *i.e.* all molecules produced in one configuration will be eliminated except these from the next configuration.

As the length of both cycles is an even number, the parity of the step number is the same for all words in a particular configuration. Moreover, it is easy to verify that words from one configuration arrive always in the same component. Therefore we can say that each configuration has a component number and a parity of computing step associated to it.

We see that in the lower cycle we decrement simultaneously both indices and that in the upper cycle we make the rotation of letters.

Because we deal with an ETVDH system, there are two types of molecules which

$$\begin{tikzcd}
 & \overset{12}{X'''\mathbf{w}Y^{IV}} \arrow[d] \arrow[dr, bend left] & \overset{11}{X'''\mathbf{w}Y'''} \arrow[l] & \overset{10}{X''\mathbf{w}Y'''} \arrow[l] & \mathbf{w} \\
 & \overset{13}{X\mathbf{w}Y^{IV}} \arrow[d] & X'''\mathbf{w}Y \arrow[d] & \overset{9}{X''\mathbf{w}Y''} \arrow[u] \arrow[r] & X''\mathbf{w} \arrow[u] \\
\text{Start} \arrow[r, Rightarrow] & \overset{14}{X\mathbf{w}Y} \arrow[d] & X'''\mathbf{w}Y_i \arrow[dl, bend left] & \overset{8}{X'\mathbf{w}Y''} \arrow[u] & \\
 & \overset{15}{X\mathbf{w}Y_i} \arrow[d] & & \overset{7}{X'\mathbf{w}Y'} \arrow[u] & \\
 & \overset{1}{X_j\mathbf{w}Y_i} \arrow[r] & \overset{2}{X_j\mathbf{w}Y_i'} \arrow[r] & \overset{3}{X_j'\mathbf{w}Y_i'} \arrow[u] \arrow[d] & \\
 & \overset{6}{X''_{j-1}\mathbf{w}Y_{i-1}} \arrow[u] & \overset{5}{X''_{j-1}\mathbf{w}Y''_{i-1}} \arrow[l] & \overset{4}{X_j'\mathbf{w}Y''_{i-1}} \arrow[l] &
\end{tikzcd}$$

Figure 5.1: The flow-chart of the computation

pass to the next step: the generated molecules and the molecules which generated them, as these last ones belong to σ^*. We show that the last ones will be eliminated during the next step, including other molecules that may be produced from them during this next step, and we will discuss in detail other cases in order to show that we perform a correct simulation.

We will show several steps of the computation and after that we will discuss each configuration in detail.

We start with the word $XwY = Xw'a_iY$, where $1 \leq i \leq n$.

$$\frac{Xw' \mid a_iY}{Z \mid Y_i} \vdash_{1.2} \frac{Xw'Y_i}{Za_iY}, \quad Xw'a_iY\,.$$

The words $Xw'a_iY$ and $Xw'Y_i$ go to the next component.

Let us consider the evolution of $Xw'a_iY$:

$$\frac{X \mid w'a_iY}{X_ka_k \mid Z} \vdash_{2.1} \frac{X_ka_kw'a_iY \uparrow}{XZ}, \quad k \in \{1,\dots,n\}.$$

We see that we do not produce any new word. For $Xw'Y_i$ we have:

$$\frac{X \mid w'Y_i}{X_ka_k \mid Z} \vdash_{2.1} \frac{X_ka_kw'Y_i}{XZ}, \quad Xw'Y_i\,, \quad k \in \{1,\dots,n\}.$$

The words $Xw'Y_i$ and $X_ka_kw'Y_i$ go to the next component.

We are at an odd step. So, the words Z_EY_i' exist and we can apply the following rule:

$$\frac{Xw' \mid Y_i}{Z_E \mid Y_i'} \vdash_{3.1} \frac{Xw'Y_i' \uparrow}{Z_EY_i}\,.$$

The word $Xw'Y_i$ is eliminated. For $X_ka_kw'Y_i$ we have:

$$\frac{X_ka_kw' \mid Y_i}{Z_E \mid Y_i'} \vdash_{3.1} \frac{X_ka_kw'Y_i'}{Z_EY_i}\,.$$

The words $X_k a_k w' Y_i$ and $X_k a_k w' Y_i'$ go to the next component.

And we can continue in a similar manner.

Now we shall discuss each configuration in details. We shall group configurations having a similar behaviour.

Group 1 Configurations $4, 5, 6, 7, 11, 13, 15$ which have a simple behaviour.

We shall discuss configuration 5 in details.

$$\begin{array}{c|c} X''_{j-1}w & Y''_{i-1} \\ \hline Z & Y_{i-1} \end{array} \quad \vdash_{1.4} \quad \frac{X''_{j-1}wY_{i-1}}{ZY''_{i-1}}, \quad X''_{j-1}wY''_{i-1}\,, \quad 2 \le i, j \le n.$$

The word $X''_{j-1}wY_{i-1}$ is in configuration 6.

The word $X''_{j-1}wY''_{i-1}$ is eliminated during the next step:

$$\begin{array}{c|c} X''_{j-1} & wY''_{i-1} \\ \hline X_{j-1} & Z \end{array} \quad \vdash_{2.5} \quad \frac{X_{j-1}wY''_{i-1} \uparrow}{X''_{j-1}Z}\,.$$

A computation similar to the one shown above permits us to advance in the flow-chart and applies for all configurations, except configuration 3. We examine below other possibilities of computation that exist for remaining configurations.

Group 2 Configuration 14, related to the application of the rule $u \to v$ of the grammar.

We can apply the rule 1.1 in addition to computations similar to group 1. This permits us to simulate the application of the rule $u \to v$ from G.

$$\begin{array}{c|c} Xw & uY \\ \hline Z & vY \end{array} \quad \vdash_{1.1} \quad \frac{XwvY}{ZuY}, \quad XwuY\,.$$

The word $XwvY$ can be further involved in a splicing in the same component for a computation similar to group 1.

Group 3 Configurations 8 and 10 where we can splice with Z' or Z''.

We can apply the rule 2.7, or 3.5 for configuration 10, in addition to computations similar to group 1, but this do not produces any new words. Below we give the description of this computation for configuration 10.

$$\begin{array}{c|c} X'' & wY''' \\ \hline \varepsilon & Z'' \end{array} \quad \vdash_{3.5} \quad \frac{wY'''}{X''Z''}, \quad X''wY'''\,,$$

$$\begin{array}{c|c} X''w & Y''' \\ \hline Z & Y^{IV} \end{array} \quad \vdash_{1.6} \quad \frac{X''wY^{IV} \uparrow}{ZY'''}\,,$$

$$\begin{array}{c|c} w & Y''' \\ \hline Z & Y^{IV} \end{array} \quad \vdash_{1.6} \quad \frac{wY^{IV} \uparrow}{ZY'''}\,.$$

Group 4 Configuration 9 where it is possible to obtain the result.

We can apply rules 2.7 and 3.5 consecutively in addition to computations similar to group 1 and 3. In this case, if $w \in T^*$, then we add w to the result.

$$\begin{array}{c|c} X''w'' & BY'' \\ \hline Z' & \varepsilon \end{array} \vdash_{2.7} \frac{X''w''}{Z'BY''}, \quad X''w''BY'',$$

$$\begin{array}{c|c} X'' & w \\ \hline \varepsilon & Z'' \end{array} \vdash_{3.5} \frac{\mathbf{w}\uparrow}{X''Z''}.$$

Group 5 Configurations 1 and 2.

We can have the following computation in addition to computations similar to group 1 and 3. We examine the case of configuration 1:

$$\begin{array}{c|c} X_1w & Y_i \\ \hline Z_E & Y_i' \end{array} \vdash_{3.1} \frac{X_1wY_i'}{Z_EY_i}, \quad X_1wY_i, \quad 1 \le i \le n.$$

The word X_1wY_i' is in configuration 2.

$$\begin{array}{c|c} X_1 & wY_i \\ \hline X_1' & Z_E \end{array} \vdash_{1.3} \frac{X_1'wY_i}{X_1Z_E}.$$

This is one of cases when a generating molecule produces a word which is not eliminated immediately.

$$\begin{array}{c|c} X_1' & wY_i \\ \hline X' & Z \end{array} \vdash_{2.3} \frac{X'wY_i\uparrow}{X_1'Z}.$$

The words $X_1'wY_i$ and $X'wY_i$ are rejected by the next component because the next step is an even step and the molecules Z_EY_i' do not exist during this step. This is why we cannot apply the rule 3.1.

For configuration 2 we have the following computation:

$$\begin{array}{c|c} X_k & wY_1' \\ \hline X_k' & Z_E \end{array} \vdash_{1.3} \frac{X_k'wY_1'}{X_kZ_E}, \quad X_kwY_1', \quad 1 \le k \le n.$$

The word $X_k'wY_1'$ is in configuration 3.

$$\begin{array}{c|c} X_kw & Y_1' \\ \hline Z & Y' \end{array} \vdash_{2.4} \frac{X_kwY'}{ZY_1'},$$

$$\begin{array}{c|c} X_kw & Y' \\ \hline Z & Y'' \end{array} \vdash_{3.3} \frac{X_kwY''\uparrow}{ZY'}.$$

The words X_kwY' and X_kwY'' are rejected by the next component because the next step is an odd step and the molecules $X_i'Z_E$ do not exist during this step. This is why we cannot apply the rule 1.3.

Group 6 Configuration 3 where we check if we are at the end of the rotation. There are 4 cases with respect to i and j:

a) $X_j'wY_1'$, b) $X_1'wY_i'$, c) $X_1'wY_1'$, d) $X_j'wY_i'$, $2 \le i, j \le n$

a) Case $\mathbf{X_j'wY_1'}$.

$$\begin{array}{c|c} X_j'w & Y_1' \\ \hline Z & Y' \end{array} \vdash_{2.4} \frac{X_j'wY'}{ZY_1'}, \quad X_j'wY_1',$$

$$\begin{array}{c|c} X_j' & wY_1' \\ \hline X_{j-1}'' & Z \end{array} \vdash_{3.2} \frac{X_{j-1}''wY_1'\uparrow}{X_j'Z}.$$

There are two rules which may be applied to X'_jwY': 3.2 and 3.3. We obtain:

$$\begin{array}{c|c} X'_jw & Y' \\ \hline Z & Y'' \end{array} \vdash_{3.3} \frac{X'_jwY'' \uparrow}{ZY''},$$

$$\begin{array}{c|c} X'_j & wY' \\ \hline X''_{j-1} & Z \end{array} \vdash_{3.2} \frac{X''_{j-1}wY' \uparrow}{X'_jZ}.$$

We can apply once more rules 3.3 and 3.2 to words above, which gives us:

$$\begin{array}{c|c} X''_{j-1}w & Y' \\ \hline Z & Y'' \end{array} \vdash_{3.3} \frac{X''_{j-1}wY'' \uparrow}{ZY'}.$$

Therefore this computation do not produce new words.

b) Case $\mathbf{X'_1wY'_i}$.

There are two rules which may be applied to $X'_1wY'_i$: 2.2 and 2.3. We obtain:

$$\begin{array}{c|c} X'_1w & Y'_i \\ \hline Z & Y''_{i-1} \end{array} \vdash_{2.2} \frac{X'_1wY''_{i-1} \uparrow}{ZY'_i},$$

$$\begin{array}{c|c} X'_1 & wY'_i \\ \hline X' & Z \end{array} \vdash_{2.3} \frac{X'wY'_i \uparrow}{X'_1Z}.$$

We can apply once more rules 2.3 and 2.2 to words above, which gives us:

$$\begin{array}{c|c} X'w & Y'_i \\ \hline Z & Y''_{i-1} \end{array} \vdash_{2.2} \frac{X'wY''_{i-1} \uparrow}{ZY'_i}.$$

Therefore this computation do not produce new words.

c) Case $\mathbf{X'_1wY'_1}$.

There are two rules which may be applied to $X'_1wY'_1$: 2.4 and 2.3. We obtain:

$$\begin{array}{c|c} X'_1w & Y'_1 \\ \hline Z & Y' \end{array} \vdash_{2.4} \frac{X'_1wY'}{ZY'_1},$$

$$\begin{array}{c|c} X'_1 & wY'_1 \\ \hline X' & Z \end{array} \vdash_{2.3} \frac{X'wY'_1 \uparrow}{X'_1Z}.$$

We can apply once more rules 2.3 and 2.4 to words above, which gives us:

$$\begin{array}{c|c} X'_1 & wY' \\ \hline X' & Z \end{array} \vdash_{2.3} \frac{\mathbf{X'wY'}}{X'_1Z}, \quad X'_1wY'.$$

The word $X'wY'$ is in configuration 7.

$$\begin{array}{c|c} X'_1w & Y' \\ \hline Z & Y'' \end{array} \vdash_{3.3} \frac{X'_1wY'' \uparrow}{ZY'}.$$

d) Case $\mathbf{X'_jwY'_i}$, $\mathbf{i,j > 1}$.

$$\begin{array}{c|c} X'_kw & Y'_i \\ \hline Z & Y''_{i-1} \end{array} \vdash_{2.2} \frac{\mathbf{X'_kwY''_{i-1}}}{ZY'_i}, \quad X'_kwY'_i.$$

The word $X'_kwY''_{i-1}$ is in configuration 4.

$$\begin{array}{c|c} X'_k & wY'_i \\ \hline X''_{k-1} & Z \end{array} \vdash_{3.2} \frac{X''_{k-1}wY'_i \uparrow}{X'_kZ}.$$

Group 7 Configuration 12, which permits to start a parallel branch.

$$\begin{array}{c|c} X''' & wY^{IV} \\ \hline X & Z \end{array} \vdash_{2.8} \frac{XwY^{IV}}{X'''Z}, \quad X'''wY^{IV}.$$

The word XwY^{IV} is in configuration 13.

$$\begin{array}{c|c} X'''w & Y^{IV} \\ \hline Z & Y \end{array} \quad \vdash_{3.6} \quad \frac{X'''wY}{ZY^{IV}}\ ,$$

$$\begin{array}{c|c} X'''w' & a_iY \\ \hline Z & Y_i \end{array} \quad \vdash_{1.2} \quad \frac{X'''w'Y_i}{Za_iY}, \quad X'''w'a_iY\ , \quad 1 \leq i \leq n,$$

$$\begin{array}{c|c} X''' & wY \\ \hline X & Z \end{array} \quad \vdash_{2.8} \quad \frac{XwY}{X'''Z}\ ,$$

$$\begin{array}{c|c} X''' & w'Y_i \\ \hline X & Z \end{array} \quad \vdash_{2.8} \quad \frac{Xw'Y_i}{X'''Z}, \quad X'''w'Y_i\ .$$

The obtained words XwY and $Xw'Y_i$ are the same as the words that are obtained by following the flow-chart of the computation. Therefore they exist already during this step.

$$\begin{array}{c|c} X'''w' & Y_i \\ \hline Z_E & Y_i' \end{array} \quad \vdash_{3.1} \quad \frac{X'''w'Y_i' \uparrow}{Z_EY_i}\ .$$

Final remarks

We note that it was possible to use the word C_1Z' instead of Z' and the word $Z''C_2$ instead of Z'' in some places of the computation above. Meanwhile these applications, which introduce C_1 and C_2 in the concerned word, do not annulate the rejection of that word by the next component.

It is easy to see that by following the flow-chart of the computation we obtain all words of $L(G)$ and, as we considered all possible cases, it is clear that the system does not produce other words. $\square$

We remark that the same result may be obtained with two components and we present this proof in Chapter 6, see Theorem 6.2.1.

5.3 Conclusions

In spite of complexity of ETVDH systems, we succeeded to direct the computation and to produce all recursively enumerable languages with only three components. The proof of this result is based on the same principles as the proof in the previous chapter, but it is much more complicated. Therefore it would be very difficult to use the same approach in the future. In the next chapter we present a different approach, which permits to decrease the number of components up to two being very simple in the same time.

Chapter 6

The method of directing molecules

The proof technique used in previous two chapters reached its limits and it cannot be further used to decrease the number of components in considered systems. In this chapter we analyse this technique and we show a new method that will permit us to decrease the number of components in the systems considered before. This new technique, the method of directing molecules, is based on a reorganisation of the computation flow. It also requires a good synchronisation between different parts of the computation flow. Moreover, this method is very generic and it can be applied to a variety of systems based on splicing, see also Chapters 7, 8, and 9.

6.1 Description of the method

First, we analyse the computation in systems presented in Theorems 4.3.1 and 5.2.2. The main idea of these computations is the following: the word that codes a sentential form of the grammar is modified at one of its ends by splicing it with an axiom. We shall say that this axiom is associated to the corresponding rule. We are interested only in one result of this splicing and rules are distributed over components in a way that permits the elimination of the second word as well as of the initial word in the case of ETVDH systems. To preserve the axiom, we use special rules that propagate axioms.

Correct simulation of the grammar is performed by using this method with judicious choice of rule placement in the components. Since necessary axioms are present all the time, any rule of a component can be used each time we arrive in that component. But, in fact, each time we need to apply only one rule. Therefore other rules may pose problems for correct simulation.

This is why we propose a new method of simulation which corrects some of defects of the method above. Now we do not permit any more to axioms to be present all the time, but only during steps when they are really needed. In order to do this, we mark them with a number, the *state*, and we increase it modulo k at each step. When we arrive to zero, the right word is recreated and we can use it for a splicing. We shall call these words *directing molecules*. So, the apparition of a directing molecule at some step of the computation depends on its *period* k and its initial state.

Now let us see what are the changes in the computation which result from the introduction of directing molecules. The application of a rule is controlled now not only by its position, but also by the period of the directing molecule associated to that rule. Therefore at each step only necessary rules may be applied, because other rules of the same component are not applicable because the second word participating in splicing is not present. This gives us the possibility to refine the control imposed by components and, by rearranging rules, to decrease the number of components necessary for a simulation of a grammar.

We present below two examples of this technique where we decrease the number of components needed to simulate a type-0 grammar by TVDH and ETVDH systems.

6.2 ETVDH systems of degree 2

In this section we show how it is possible to apply the reflections above. We consider ETVDH systems and we show that two components are enough in order to simulate an arbitrary grammar.

Theorem 6.2.1. *Let $G = (N, T, P, S)$ be a type-0 grammar. Then, there is an ETVDH system $E_G = (V, T, A, R_1, R_2)$ of degree 2 which simulates G and $L(G) = L(E_G)$.*

Proof. We shall prove this assertion in the following way. Firstly we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(E_G)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. Consequently, our assertion will be proved.

Our simulation is based on the "rotate-and-simulate" method, see Section 4.1.2. We already described how to adapt this method to ETVDH systems during the proof of Theorem 5.2.2.

At first we give the flow-chart of the computation and the description of the method that we use. After that we give the formal definition of the system.

The flow-chart of the computation

The computation in E_G follows the flow-chart shown in the Fig. 6.1. The vertices of the flow-chart show a configuration of molecules during the computation. We enumerate all configurations and their numbers are in the upper right corner. The flow-chart is composed from two parts: the upper part and the lower part. Configurations having the same number in both parts occur in the same time. Is is easy to see that in the lower part we decrease simultaneously indices of X and Y and that we continue to the upper part when both indices are equal to one. The upper part completes the rotation, simulates rules of the grammar, and permits to produce a resulting word. It is easy to verify that words from one particular configuration always arrive in a component with the same number. Therefore, we can say that each configuration has an associated component. In configurations, the symbol $\mathbf{w}$ is treated as a variable, and it may have different values in different configurations. For example, if in configuration 8 the symbol $\mathbf{w}$ is equal to $w'a_i$ then in configuration 1 it may have the value w'. We will show that the computation follows the flow-chart from the Fig. 6.1, *i.e.*, all molecules produced in one configuration will be eliminated except molecules from the next configuration.

$$\begin{tikzcd}
 & & \overset{5}{X''\mathbf{w}Y'''} \arrow[dl] & & \mathbf{w} \\
 & \overset{6}{X\mathbf{w}Y'''} \arrow[d] & & \overset{4}{X''\mathbf{w}Y''} \arrow[ul] \arrow[r] & X''\mathbf{w} \arrow[u] \\
\text{Start} \arrow[r, Rightarrow] & \overset{7}{X\mathbf{w}Y} \arrow[d] & & \overset{3}{X'\mathbf{w}Y''} \arrow[u] & \\
 & \overset{8}{X_j\mathbf{w}Y} \arrow[dr] & & \overset{2}{X'\mathbf{w}Y'} \arrow[u] & \\
 & & \overset{1}{X_j\mathbf{w}Y_i} \arrow[ur] \arrow[dr] & & \\
 & \overset{8}{X'''_{j-1}\mathbf{w}Y_{i-1}} \arrow[ur] & & \overset{2}{X_j\mathbf{w}Y'_{i-1}} \arrow[d] & \\
 & \overset{7}{X'''_{j-1}\mathbf{w}Y'''_{i-1}} \arrow[u] & & \overset{3}{X'_{j-1}\mathbf{w}Y'_{i-1}} \arrow[d] & \\
 & \overset{6}{X''_{j-1}\mathbf{w}Y'''_{i-1}} \arrow[u] & & \overset{4}{X'_{j-1}\mathbf{w}Y''_{i-1}} \arrow[dl] & \\
 & & \overset{5}{X''_{j-1}\mathbf{w}Y''_{i-1}} \arrow[ul] & &
\end{tikzcd}$$

Figure 6.1: The flow-chart of the computation

Directing molecules

The transition between two configurations is done by making changes at one of the ends of the word. For this, the word is spliced with directing molecules of form Z_kT_k or T_mZ_m which make changes at the right end, respectively left end, of the word. The method of directing molecules permits us to avoid erroneous branches of the computation.

Directing molecules, see also Section 6.1, are created only at the moment when they are really necessary. For example, directing molecules for configuration 5 (Z_4Y_j''', XZ_{14}, C_1Z_{15}) are created during the previous step and they will not exist in configuration 8. Therefore, each molecule can be in one of eight states and at each step the state of the molecule is incremented modulo 8. When the state of the molecule is zero, in this case it is omitted, we obtain the directing molecule which can be further used. For example, we are in configuration 5 and we have the directing molecule XZ_{14}. We mark it with 1 by the rule $1.z.5$: $(X|Z_{14}, Z|Z_{14}^1) \vdash_{1.z.5} (XZ_{14}^1, ZZ_{14})$. During the next step, the molecule XZ_{14}^1 changes its state in 2 after the application of the rule $2.z.1$. We continue in a similar manner and we obtain the following computation: $XZ_{14} \Rightarrow XZ_{14}^1 \Rightarrow XZ_{14}^2 \Rightarrow \ldots \Rightarrow XZ_{14}^7 \Rightarrow XZ_{14}$. Thus, the molecule XZ_{14} appears with a period equal to 8 and it is present in configurations 4, 5 and 6, but it can be used in configuration 5 only, see also page 72.

So, this technique permits to refine the control of the application of rules of the same component and it permits to divide them in subsets which are activated by specific directing molecules.

Formal definition of the system

Now we give the formal definition of the system.

We construct $E_G = (V, T, A, R_1, R_2)$ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(B = a_n)$ and $B \notin V \cup T$.

In what follows we assume that:
$1 \leq i \leq n,\ 1 \leq j \leq n-1,\ 2 \leq l \leq n,\ 1 \leq k \leq 11, 12 \leq m \leq 20,\ 1 \leq s \leq 7,$
$\mathbf{a}, \mathbf{b} \in N \cup T \cup \{B\}$.

The alphabet V is defined by:
$V = N \cup T \cup \{B\} \cup \{X, Y, X', Y'X'', Y'', Y''', X_i, Y_i, X_j', Y_j', X_j'', Y_j'', X_j''', Y_j''', Z_k, Z_k^s, Z_m, Z_m^s, Z, C_1, C_2\}$.

The terminal alphabet T is the same as for the grammar G.

Axioms are defined by:
$A = \{XBSY\} \cup \{X_ia_iZ_{16}, Z_5Y_j, X_j'''Z_{19}^1, Z_9^1Y, XZ_{14}^2, C_1Z_{15}^2, Z_4^2Y_j''', X_j''Z_{18}^3, Z_7^3Y''', Z_8^3C_2, X''Z_{13}^4, Z_3^4Y_j'', X_j'Z_{17}^5, Z_6^5Y'', Z_1^6Y_j', Z_2^6Y', X'Z_{12}^6, X_jZ_{20}^7, Z_{10}^7Y_i\} \cup \{Z_kZ, Z_k^sZ, ZZ_m, ZZ_m^s,\} \cup \{Z_{11}^7vY : \exists u \rightarrow v \in P\}$.

Now we shall give the definition of components of the system. The numbering of rules not having z in their names follows the following conventions:

- The first number shows the component that this rule belongs to.
- The second number indicates in which configuration this rule may be applied.
- The third number shows the position on the flow-chart of computation of the molecule to which this rule may be applied: if it is equal to 1, then the rule is applied to a molecule being in a configuration from a lower part of the flow-chart, while if it is greater than 1, then the rule is applied to a molecule being in a configuration from the upper part of the flow-chart.

The numbering of rules having z in their names follows the following conventions:

- The first number shows the component to which this rule belongs to.
- The second number which is just after z indicates the parity of the step when the directing molecule (to which this rule is applied) is used. If this number is between 1 and 8, then this rule is applied to a directing molecule which is utilisable during an odd step; if the number is between 9 and 16, then the rule is applied to a directing molecule which is utilisable during an even step.

Component R_1:

Rules for transition between configurations:

$1.1.1 : \begin{array}{c|c} \mathbf{a} & Y_l \\ \hline Z_1 & Y'_{l-1} \end{array}$; $1.1.2 : \begin{array}{c|c} \mathbf{a} & Y_1 \\ \hline Z_2 & Y' \end{array}$; $1.1.3 : \begin{array}{c|c} X_1 & \mathbf{a} \\ \hline X' & Z_{12} \end{array}$;

$1.3.1 : \begin{array}{c|c} \mathbf{a} & Y'_j \\ \hline Z_3 & Y''_j \end{array}$; $1.3.2 : \begin{array}{c|c} X' & \mathbf{a} \\ \hline X'' & Z_{13} \end{array}$; $1.5.1 : \begin{array}{c|c} \mathbf{a} & Y''_j \\ \hline Z_4 & Y'''_j \end{array}$;

$1.5.2 : \begin{array}{c|c} X'' & \mathbf{a} \\ \hline X & Z_{14} \end{array}$; $1.5.3 : \begin{array}{c|c} X'' & \mathbf{a} \\ \hline \varepsilon & C_1 Z_{15} \end{array}$; $1.7.1 : \begin{array}{c|c} \mathbf{a} & Y'''_j \\ \hline Z_5 & Y_j \end{array}$; $1.7.2 : \begin{array}{c|c} X & \mathbf{a} \\ \hline X_i a_i & Z_{16} \end{array}$;

Rules for creation of directing molecules:

Directing molecules for odd steps:

$1.z.1 : \begin{array}{c|c} Z_{k_1} & T_{k_1} \\ \hline Z^1_{k_1} & Z \end{array}$; $1.z.2 : \begin{array}{c|c} Z^2_{k_1} & T_{k_1} \\ \hline Z^3_{k_1} & Z \end{array}$; $1.z.3 : \begin{array}{c|c} Z^4_{k_1} & T_{k_1} \\ \hline Z^5_{k_1} & Z \end{array}$; $1.z.4 : \begin{array}{c|c} Z^6_{k_1} & T_{k_1} \\ \hline Z^7_{k_1} & Z \end{array}$;

$1.z.5 : \begin{array}{c|c} T_{m_1} & Z_{m_1} \\ \hline Z & Z^1_{m_1} \end{array}$; $1.z.6 : \begin{array}{c|c} T_{m_1} & Z^2_{m_1} \\ \hline Z & Z^3_{m_1} \end{array}$; $1.z.7 : \begin{array}{c|c} T_{m_1} & Z^4_{m_1} \\ \hline Z & Z^5_{m_1} \end{array}$; $1.z.8 : \begin{array}{c|c} T_{m_1} & Z^6_{m_1} \\ \hline Z & Z^7_{m_1} \end{array}$;

Directing molecules for even steps:

$1.z.9 : \begin{array}{c|c} Z^1_{k_2} & T_{k_2} \\ \hline Z^2_{k_2} & Z \end{array}$; $1.z.10 : \begin{array}{c|c} Z^3_{k_2} & T_{k_2} \\ \hline Z^4_{k_2} & Z \end{array}$; $1.z.11 : \begin{array}{c|c} Z^5_{k_2} & T_{k_2} \\ \hline Z^6_{k_2} & Z \end{array}$; $1.z.12 : \begin{array}{c|c} Z^7_{k_2} & T_{k_2} \\ \hline Z_{k_2} & Z \end{array}$;

$1.z.13 : \begin{array}{c|c} T_{m_2} & Z^1_{m_2} \\ \hline Z & Z^2_{m_2} \end{array}$; $1.z.14 : \begin{array}{c|c} T_{m_2} & Z^3_{m_2} \\ \hline Z & Z^4_{m_2} \end{array}$; $1.z.15 : \begin{array}{c|c} T_{m_2} & Z^5_{m_2} \\ \hline Z & Z^6_{m_2} \end{array}$; $1.z.16 : \begin{array}{c|c} T_{m_2} & Z^7_{m_2} \\ \hline Z & Z_{m_2} \end{array}$;

Component R_2:

Rules for transition between configurations:

$2.2.1 : \begin{array}{c|c} X_l & \mathbf{a} \\ \hline X'_{l-1} & Z_{17} \end{array}$; $2.2.2 : \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z_6 & Y'' \end{array}$; $2.4.1 : \begin{array}{c|c} X'_j & \mathbf{a} \\ \hline X''_j & Z_{18} \end{array}$; $2.4.2 : \begin{array}{c|c} \mathbf{ab} & Y'' \\ \hline Z_7 & Y''' \end{array}$;

$2.4.3 : \begin{array}{c|c} \mathbf{a} & BY'' \\ \hline Z_8 C_2 & \varepsilon \end{array}$; $2.6.1 : \begin{array}{c|c} X''_j & \mathbf{a} \\ \hline X'''_j & Z_{19} \end{array}$; $2.6.2 : \begin{array}{c|c} \mathbf{a} & Y''' \\ \hline Z_9 & Y \end{array}$; $2.8.1 : \begin{array}{c|c} X'''_j & \mathbf{a} \\ \hline X_j & Z_{20} \end{array}$;

$2.8.2 : \begin{array}{c|c} \mathbf{a} & a_i Y \\ \hline Z_{10} & Y_i \end{array}$; $2.8.3 : \begin{array}{c|c} \mathbf{a} & uY \\ \hline Z_{11} & vY \end{array}$;

Rules for creation of directing molecules:

Directing molecules for odd steps:

$2.z.1 : \begin{array}{c|c} Z^1_{k_1} & T_{k_1} \\ \hline Z^2_{k_1} & Z \end{array}$; $2.z.2 : \begin{array}{c|c} Z^3_{k_1} & T_{k_1} \\ \hline Z^4_{k_1} & Z \end{array}$; $2.z.3 : \begin{array}{c|c} Z^5_{k_1} & T_{k_1} \\ \hline Z^6_{k_1} & Z \end{array}$; $2.z.4 : \begin{array}{c|c} Z^7_{k_1} & T_{k_1} \\ \hline Z_{k_1} & Z \end{array}$;

$2.z.5 : \begin{array}{c|c} T_{m_1} & Z^1_{m_1} \\ \hline Z & Z^2_{m_1} \end{array}$; $2.z.6 : \begin{array}{c|c} T_{m_1} & Z^3_{m_1} \\ \hline Z & Z^4_{m_1} \end{array}$; $2.z.7 : \begin{array}{c|c} T_{m_1} & Z^5_{m_1} \\ \hline Z & Z^6_{m_1} \end{array}$; $2.z.8 : \begin{array}{c|c} T_{m_1} & Z^7_{m_1} \\ \hline Z & Z_{m_1} \end{array}$;

Directing molecules for even steps:

$$2.z.9 : \frac{Z_{k_2} \mid T_{k_2}}{Z^1_{k_2} \mid Z}\ ; \quad 2.z.10 : \frac{Z^2_{k_2} \mid T_{k_2}}{Z^3_{k_2} \mid Z}\ ; \quad 2.z.11 : \frac{Z^4_{k_2} \mid T_{k_2}}{Z^5_{k_2} \mid Z}\ ; \quad 2.z.12 : \frac{Z^6_{k_2} \mid T_{k_2}}{Z^7_{k_2} \mid Z}\ ;$$

$$2.z.13 : \frac{T_{m_2} \mid Z_{m_2}}{Z \mid Z^1_{m_2}}\ ; \quad 2.z.14 : \frac{T_{m_2} \mid Z^2_{m_2}}{Z \mid Z^3_{m_2}}\ ; \quad 2.z.15 : \frac{T_{m_2} \mid Z^4_{m_2}}{Z \mid Z^5_{m_2}}\ ; \quad 2.z.16 : \frac{T_{m_2} \mid Z^6_{m_2}}{Z \mid Z^7_{m_2}}\ ;$$

where $1 \le k_1 \le 5$, $6 \le k_2 \le 11$, $12 \le m_1 \le 16$, $17 \le m_2 \le 20$.
The symbols T_k and T_m are given by the following tables:

k	1	2	3	4	5	6	7	8	9	10	11
T_k	Y'_j	Y'	Y''_j	Y'''_j	Y_j	Y''	Y'''	C_2	Y	Y_i	vY

m	12	13	14	15	16	17	18	19	20
T_m	X'	X''	X	C_1	$X_i a_i$	X'_j	X''_j	X'''_j	X_j

Components R_1 and R_2 contain also the following rules:
$\dfrac{\alpha \mid \varepsilon}{\alpha \mid \varepsilon}$ where $\alpha \in \{Z_k Z, Z^s_k Z, Z Z_m, Z Z^s_m\}$.

Notations

We shall use the same notation as we used in Theorem 5.2.2:

$$\frac{w_1 \mid w_2}{w'_1 \mid w'_2} \quad \vdash_r \quad \frac{w_1 {w'_2}^*}{w'_1 w_2}, \quad w_1 w_2.$$

The meaning of this notation may be found at page 60.

Description of the computation

It is easy to see that in each configuration n we have three types of directing molecules that exist:

- Directing molecules for the previous configuration, $n-1$, because they participated in a splicing during the previous step and being by definition part of σ^* they are present during this step.

- Directing molecules for the current configuration, n, because they were created during the previous step.

- Directing molecules for the next configuration, $n+1$, because they are created during this step and by definition of ETVDH systems they can be further involved in a splicing during this step.

The behaviour of all configurations is similar. We discuss now the behaviour of configurations 5 and 1, see Fig. 6.1.

Configuration 5. We have the following molecules: $X''wY'''$, $X''_{j-1}wY''_{i-1}$ $(i,j>1)$ and the following directing molecules: XZ_{14}, C_1Z_{15}, $Z_4Y'''_j$, X''_jZ_{18}, Z_7Y''', Z_8C_2, X'''_jZ_{19}, Z_9Y. The first three molecules are directing molecules for configuration 5. We are in the first component and we have the following computation:

$$\begin{array}{c|c} X'' & wY''' \\ \hline X & Z_{14} \end{array} \quad \vdash_{1.5.2} \quad \frac{XwY'''}{X''Z_{14}}, \quad X''wY''' ,$$

$$\begin{array}{c|c} X'' & wY''' \\ \hline \varepsilon & C_1Z_{15} \end{array} \quad \vdash_{1.5.3} \quad \frac{wY'''}{X''C_1Z_{15}}, \quad X''wY''' ,$$

$$\begin{array}{c|c} X''_{j-1}w & Y''_{i-1} \\ \hline Z_4 & Y'''_{i-1} \end{array} \quad \vdash_{1.5.1} \quad \frac{X''_{j-1}wY'''_{i-1}}{Z_4Y''_{i-1}}, \quad X''_{j-1}wY''_{i-1} .$$

The words XwY''' et $X''_{j-1}wY'''_{i-1}$ are in configuration 6. The molecules $X''wY'''$, wY''' and $X''_{j-1}wY''_{i-1}$ are eliminated during the next step:

$$\begin{array}{c|c} X''w & Y''' \\ \hline Z_9 & Y \end{array} \quad \vdash_{2.6.2} \quad \frac{X''wY \uparrow}{Z_9Y'''} ,$$

$$\begin{array}{c|c} w & Y''' \\ \hline Z_9 & Y \end{array} \quad \vdash_{2.6.2} \quad \frac{wY \uparrow}{Z_9Y'''} ,$$

$$\begin{array}{c|c} X''_{j-1} & wY''_{i-1} \\ \hline X'''_{j-1} & Z_{19} \end{array} \quad \vdash_{2.6.1} \quad \frac{X'''_{j-1}wY''_{i-1} \uparrow}{X''_{j-1}Z_{19}} .$$

Thus, we showed that we obtain the molecules XwY''' and $X''_{j-1}wY'''_{i-1}$ while all other molecules are eliminated, *i.e.*, we followed the flow-chart.

Configuration 1. We have the following molecules: X_jwY_i, as well as the following directing molecules: $Z_1Y'_j$, Z_2Y', $X'Z_{12}$, $Z_{10}Y_i$, X_jZ_{20}, X'_jZ_{17}, Z_6Y''.

We shall distinguish 4 cases:

a) $i,j>1$

$$\begin{array}{c|c} X_jw & Y_i \\ \hline Z_1 & Y'_{i-1} \end{array} \quad \vdash_{1.1.1} \quad \frac{X_jwY'_{i-1}}{Z_1Y_i}, \quad X_jwY_i .$$

The molecule $X_jwY'_{i-1}$ is in configuration 2. The molecule X_jwY_i is eliminated during the next step:

$$\begin{array}{c|c} X_j & wY_i \\ \hline X'_{j-1} & Z_{17} \end{array} \quad \vdash_{2.2.1} \quad \frac{X'_{j-1}wY_i \uparrow}{X_jZ_{17}} .$$

b) $i,j=1$

We can apply two rules: 1.1.2 or 1.1.3:

$$\begin{array}{c|c} X_1w & Y_1 \\ \hline Z_2 & Y' \end{array} \quad \vdash_{1.1.2} \quad \frac{X_1wY'}{Z_2Y_1} ,$$

$$\begin{array}{c|c} X_1 & wY_1 \\ \hline X' & Z_{12} \end{array} \quad \vdash_{1.1.3} \quad \frac{X'wY_1 \uparrow}{X_1Z_{12}} .$$

We can apply once more the rule 1.1.3 or 1.1.2 to the result of the previous application:

$$\begin{array}{c|c} X_1 & wY' \\ \hline X' & Z_{12} \end{array} \quad \vdash_{1.1.3} \quad \frac{X'wY'}{X_1Z_{12}}, \quad X_1wY' .$$

The molecule $X'wY'$ is in configuration 2. The molecule X_1wY' will be eliminated during the next step:

$$\begin{array}{c|c} X_1w & Y' \\ \hline Z_6 & Y'' \end{array} \vdash_{2.2.2} \frac{X_1wY''\uparrow}{Z_6Y'}\ .$$

c) $i>1,\ j=1$

$$\begin{array}{c|c} X_1w & Y_i \\ \hline Z_1 & Y'_{i-1} \end{array} \vdash_{1.1.1} \frac{X_1wY'_{i-1}\uparrow}{Z_1Y_i}\ ,$$

$$\begin{array}{c|c} X_1 & wY_i \\ \hline X' & Z_{12} \end{array} \vdash_{1.1.3} \frac{X'wY_i\uparrow}{X_1Z_{12}}\ .$$

We can apply one more time the rule 1.1.1 or 1.1.3 to molecules obtained above:

$$\begin{array}{c|c} X'w & Y_i \\ \hline Z_1 & Y'_{i-1} \end{array} \vdash_{1.1.1} \frac{X'wY'_{i-1}\uparrow}{Z_1Y_i}\ .$$

d) $i=1,\ j>1$

$$\begin{array}{c|c} X_jw & Y_1 \\ \hline Z_2 & Y' \end{array} \vdash_{1.1.2} \frac{X_jwY'}{Z_2Y_1},\quad X_jwY_1\ .$$

The molecules X_jwY_1 et X_jwY' are eliminated during the next step:

$$\begin{array}{c|c} X_j & wY_1 \\ \hline X'_{j-1} & Z_{17} \end{array} \vdash_{2.2.1} \frac{X'_{j-1}wY_1\uparrow}{X_jZ_{17}}\ ,$$

$$\begin{array}{c|c} X_j & wY' \\ \hline X'_{j-1} & Z_{17} \end{array} \vdash_{2.2.1} \frac{X'_{j-1}wY'\uparrow}{X_jZ_{17}}\ ,$$

$$\begin{array}{c|c} X_jw & Y' \\ \hline Z_6 & Y'' \end{array} \vdash_{2.2.2} \frac{X_jwY''\uparrow}{Z_6Y'}\ .$$

We can apply one more time the rule 2.2.1 or 2.2.2 to molecules obtained above:

$$\begin{array}{c|c} X_j & wY'' \\ \hline X'_{j-1} & Z_{17} \end{array} \vdash_{2.2.1} \frac{X'_{j-1}wY''\uparrow}{X_jZ_{17}}\ .$$

So, we obtained the molecules $X_jwY'_{i-1}$ and $X'wY'$. In the same time all other molecules were eliminated, *i.e.* we followed the flow-chart.

Final remarks

As it is described above, we can pass from one configuration to another one, thus following the flow-chart of computation from Fig. 6.1. The simulation of the rule $u \to v$ of the grammar is made in configuration 8 by the rule 2.8.3. As we can see, we implemented the "rotate-and-simulate" method which permits to simulate the grammar.

In order to obtain the result we apply the rule 2.4.3 in configuration 4 and after that the rule 1.5.3:

$$\begin{array}{c|c} X''w & BY'' \\ \hline Z_8C_2 & \varepsilon \end{array} \vdash_{2.4.3} \frac{X''w}{Z_8C_2BY''},\quad X''wBY''\ .$$

$$\begin{array}{c|c} X'' & w \\ \hline \varepsilon & C_1Z_{15} \end{array} \vdash_{1.5.3} \frac{\mathbf{w}\uparrow}{X''C_1Z_{15}}\ .$$

$$\begin{array}{c|c} X'' & w \\ \hline X & Z_{14} \end{array} \vdash_{1.5.2} \frac{Xw\uparrow}{X''Z_{14}}\ .$$

If $w \in T^*$, then it belongs to the result. We note that molecules Z_8C_2BY'' and $X''C_1Z_{15}$ persist forever being produced once, but they cannot produce new words.

It is easy to see that by following the flow-chart of computation we generate all words of $L(G)$, see also Section 4.1.2, and it is clear after the discussion above that the system does not produce other words.

□

6.3 TVDH systems of degree 1

In this section we show another example of application of the method of directing molecules. We consider TVDH systems and we show that one component is sufficient to simulate an arbitrary grammar.

Theorem 6.3.1. *Let $G = (N, T, P, S)$ be a type-0 grammar. Then, there is a TVDH system $D_G = (V, T, A, R)$ of degree 1 which simulates G and $L(G) = L(D_G)$.*

This theorem is a corollary of Theorem 4.3.1 and of the following lemma.

Lemma 6.3.2. *For any TVDH system D_G^2 constructed in Theorem 4.3.1 there is a TVDH system of degree 1, $D_G = (V, T, A, R)$, which simulates D_G^2 and $L(D_G) = L(D_G^2)$.*

Proof. In order to prove this lemma we transform, for technical reasons, the TVDH system D_G^2 from Theorem 4.3.1 into a form which is more suitable for our simulation. In order to do this, some letters are numbered and some rules have bigger sites, but these transformations do not alter neither the behaviour of the system nor the generated language. We do not make any more the distinction between this transformed system and the system from Theorem 4.3.1 and we shall refer below to it as D_G^2, see Fig. 6.2. This system has the following properties:

- It has two components.
- It has two independent subsets of axioms which persist during the computation. One of these subsets is used for splicing in the first component only, the other one is used for splicing in the second component only .
- The rules are constructed in a such way that molecules that code different stages of the grammar simulation can be spliced only with axioms from the two said subsets.

We shall use the method of directing molecules in order to simulate D_G^2. At first we place rules 1.1 to 1.9 from R_1^2 and rules 2.1 to 2.8 from R_2^2 in one component. We shall call these rules main rules. We can easily see that main rules permit to implement the "rotate-and-simulate" method, while all other rules are necessary to propagate axioms. Each axiom from A_1^2 or A_2^2 of the initial system, *i.e.*, associated to a main rule, become a directing molecule with period of 2. Since we have only one state, we replace it by a prime. We say that a directing molecule is in the passive state if it is primed, otherwise we say that it is in the active state.

$D^2_G = (V^2, T^2, A^2, R^2_1, R^2_2)$.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \dots, a_n\}$ $(a_n = B)$ and $B \notin N \cup T$.

In what follows we assume that $1 \leq \mathbf{i} \leq n$, $1 \leq \mathbf{j} \leq n-1$, $2 \leq \mathbf{k} \leq n$, $\mathbf{a}, \mathbf{b} \in N \cup T \cup \{B\}$.

The alphabet V^2 is defined by:
$V^2 = N \cup T \cup \{B\} \cup \{X, Y, X_{\mathbf{i}}, Y_{\mathbf{i}}, X'_{\mathbf{j}}, Y'_{\mathbf{j}}, X''_{\mathbf{j}}, Y''_{\mathbf{j}}, X', Y', X'', Y'', C_1, C_2, D_1, D_2, Z_m\} (1 \leq m \leq 18)$.

The terminal alphabet T^2 is the same as for the formal grammar G.
The axioms are defined by: $A^2 = \{XSBY\} \cup A^2_1 \cup A^2_2 \cup \{C_1 Z_{16}, Z_7 C_2, D_1, D_2\}$, where $A^2_1 = \{Z_1 v Y_{\mathbf{j}}, \exists u \to v a_{\mathbf{j}} \in P\} \cup \{Z_1 Y_{\mathbf{i}}, Z_2 Y_{\mathbf{i}}, Z_3 Y_{\mathbf{j}}, Z_4 Y'_{\mathbf{j}}, Z_5 Y''_{\mathbf{j}}, Z_6 Y, X' Z_8, X'' Z_9\}$ and $A^2_2 = \{X_{\mathbf{i}} a_{\mathbf{i}} Z_{11}, X'_{\mathbf{j}} Z_{13}, X''_{\mathbf{j}} Z_{14}, Z_{17} Y', Z_{18} Y'', X_{15} Z, X_{\mathbf{j}} Z_{12}\}$.

The components of the system are defined as follows.
Component R^2_1:

$$1.1 : \begin{array}{c|c} \mathbf{a} & uY \\ \hline Z_1 & vY_{\mathbf{j}} \end{array}, {\scriptstyle \exists u \to v a_{\mathbf{j}} \in P}; \quad 1.1' : \begin{array}{c|c} \mathbf{a} & a_{\mathbf{i}} uY \\ \hline Z_1 & Y_{\mathbf{i}} \end{array}, {\scriptstyle \exists u \to \varepsilon \in P}; \quad 1.2 : \begin{array}{c|c} \mathbf{a} & a_{\mathbf{i}} Y \\ \hline Z_2 & Y_{\mathbf{i}} \end{array};$$

$$1.3 : \begin{array}{c|c} \mathbf{ab} & Y_{\mathbf{k}} \\ \hline Z_4 & Y'_{\mathbf{k}-1} \end{array}; \quad 1.4 : \begin{array}{c|c} \mathbf{a} & Y'_{\mathbf{j}} \\ \hline Z_5 & Y''_{\mathbf{j}} \end{array}; \quad 1.5 : \begin{array}{c|c} X_1 & \mathbf{ab} \\ \hline X' & Z_8 \end{array}; \quad 1.6 : \begin{array}{c|c} X' & \mathbf{a} \\ \hline X'' & Z_9 \end{array};$$

$$1.7 : \begin{array}{c|c} \mathbf{a} & Y'' \\ \hline Z_6 & Y \end{array}; \quad 1.8 : \begin{array}{c|c} \mathbf{a} & BY'' \\ \hline Z_7 & \varepsilon \end{array}; \quad 1.9 : \begin{array}{c|c} \mathbf{a} & Y''_{\mathbf{j}} \\ \hline Z_3 & Y_{\mathbf{j}} \end{array}; \quad 1.10 : \begin{array}{c|c} C_1 & Z_{16} \\ \hline \varepsilon & D_1 \end{array};$$

Component R^2_2:

$$2.1 : \begin{array}{c|c} X & \mathbf{a} \\ \hline X_{\mathbf{i}} a_{\mathbf{i}} & Z_{11} \end{array}; \quad 2.2 : \begin{array}{c|c} X_{\mathbf{k}} & \mathbf{ab} \\ \hline X'_{\mathbf{k}-1} & Z_{13} \end{array}; \quad 2.3 : \begin{array}{c|c} X'_{\mathbf{j}} & \mathbf{a} \\ \hline X''_{\mathbf{j}} & Z_{14} \end{array}; \quad 2.4 : \begin{array}{c|c} \mathbf{a} & Y_1 \\ \hline Z_{17} & Y' \end{array};$$

$$2.5 : \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z_{18} & Y'' \end{array}; \quad 2.6 : \begin{array}{c|c} X'' & \mathbf{a} \\ \hline X & Z_{15} \end{array}; \quad 2.7 : \begin{array}{c|c} X'' & \mathbf{a} \\ \hline \varepsilon & Z_{16} \end{array}; \quad 2.8 : \begin{array}{c|c} X''_{\mathbf{j}} & \mathbf{a} \\ \hline X_{\mathbf{j}} & Z_{12} \end{array}; \quad 2.9 : \begin{array}{c|c} Z_7 & C_2 \\ \hline D_2 & \varepsilon \end{array};$$

Components R^2_1 and R^2_2 contain also the following rules:

$$\begin{array}{c|c} \alpha & \varepsilon \\ \hline \alpha & \varepsilon \end{array} \quad \text{for each axiom } \alpha \in A^2, \text{ except } XSBY.$$

Figure 6.2: The system D^2_G

Now we organise the computation in a way to have directing molecules from A^2_1 in the active state during an odd step and in the passive state during an even step. Similarly, we make directing molecules from A^2_2 to appear in the passive state during an odd step and in the active state during an even step. We note that directing molecules which are in the passive state match only rules which permit to make them active. Directing molecules which are in the active state may enter rules which permit to make them passive as well as rules which correspond to the main rules from D^2_G. We also note that molecules, which are complementary to directing molecules with respect to rules that change their state, are already present in the system.

We see that by this process we simulate the behaviour of the first component during an odd step and the behaviour of the second component during an even step, see Fig. 6.3 and 6.5. This leads to a correct simulation of the system D_G^2. The transformation of the system D_G^2 into D_G is shown at Fig. 6.4.

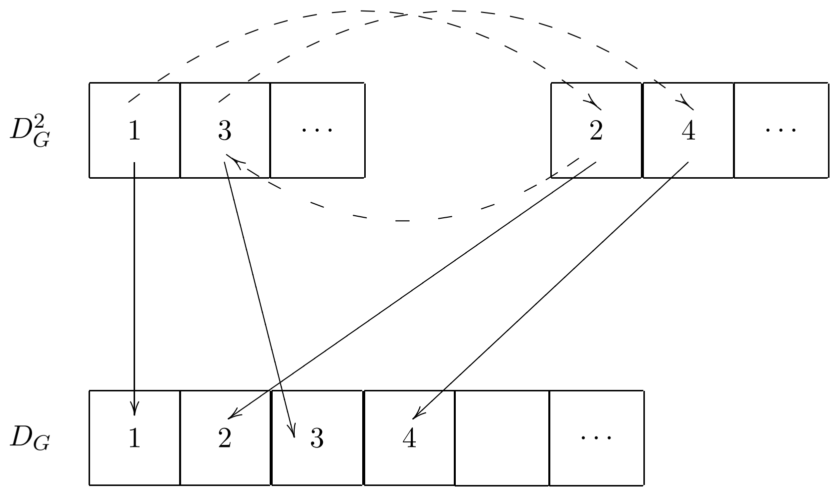


Figure 6.3: Simulation of components of the system D_G^2. The step number is indicated in the boxes.

For example, we have the molecule Z_2Y_i during an odd step. It may be used in rule 2 for a splicing and we simulate in this case the rule 1.2 of the system D_G^2. We can also deactivate this molecule by the rule $z2$: $(Z_2Y_i, Z_2'R_1) \vdash_{z2} (Z_2'Y_i, Z_2R_1)$. During the next step, which is even, the molecule Z_2Y_i does not exist any more, therefore the rule 2 cannot be applied. From the other side, we can activate the molecule $Z_2'R_1$ by using the rule $z2'$, because its complementary molecule Z_2R_1 was produced during the previous step. So, we obtain again the molecule Z_2R_i and we are at an odd step. The Fig. 6.5 contains an illustration of it.

Similar reasoning applies to all molecules having Z with indices.

Rules 1.8, 1.10, 2.7 and 2.9 are used in D_G^2 in order to obtain a resulting terminal word. In D_G, for technical reasons, they are simulated in a special way by rules 7, 16, x.1, x.2, and r.1 to r.16.

Formal description of the system

Now we will give a more formal definition of the system.

We define $D_G = (V, T, A, R_1)$ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \dots, a_n\}$ $(B = a_n)$ and $B \notin N \cup T$.

In what follows we assume that:

$1 \le i \le n,\ 1 \le j \le n-1,\ 2 \le k \le n,\ s \in \{0,2\},\ \mathbf{a}, \mathbf{b} \in N \cup T \cup \{B\}$.

The alphabet V is defined by $V = \{V^2 \setminus \{C_1, C_2, D_1, D_2\}\} \cup \{R_1, R_2, Z'_m\ (1 \le m \le 18)\} \cup \{C_1^s, C_2^s, R, Z_7^p, Z_{16}^p, Z_D^p, Z_E^p\ (1 \le p \le 3), D_1^s, E_2^s, Z_D, Z_E\}$.

TVDH2 system D_G^2

Component I

Rule 1.2: $\begin{array}{c|c} \mathbf{a} & a_iY \\ \hline Z_2 & Y_i \end{array}$

Axiom: Z_2Y_i

Rule for axiom: (present in both components) $\begin{array}{c|c} Z_2Y_i & \varepsilon \\ \hline Z_2Y_i & \varepsilon \end{array}$

Component II

Rule 2.2: $\begin{array}{c|c} X_l & \mathbf{ab} \\ \hline X'_{l-1} & Z_{13} \end{array}$

Axiom: X'_jZ_{13}

Rule for axiom: (present in both components) $\begin{array}{c|c} X'_jZ_{13} & \varepsilon \\ \hline X'_jZ_{13} & \varepsilon \end{array}$

TVDH1 system D_G (1 component)

Rule 2: $\begin{array}{c|c} \mathbf{a} & a_iY \\ \hline Z_2 & Y_i \end{array}$

Axioms: $Z_2Y_i,\ Z'_2R_1$

Rules for axioms: $z2 : \begin{array}{c|c} Z_2 & Y_i \\ \hline Z'_2 & R_1 \end{array}$ $z2' : \begin{array}{c|c} Z'_2 & Y_i \\ \hline Z_2 & R_1 \end{array}$

Rule 13: $\begin{array}{c|c} X_l & \mathbf{ab} \\ \hline X'_{l-1} & Z_{13} \end{array}$

Axioms: $X'_jZ_{13},\ R_2Z_{13}$

Rules for axioms: $z13 : \begin{array}{c|c} X'_j & Z_{13} \\ \hline R_2 & Z'_{13} \end{array}$ $z13' : \begin{array}{c|c} X'_j & Z'_{13} \\ \hline R_2 & Z_{13} \end{array}$

Figure 6.4: Transformation of TVDH2 system D_G^2 into a TVDH1 system.

The terminal alphabet T is the same as the one of D_G^2 ($T = T^2$), *i.e.*, the terminal alphabet of G.

Axioms are defined by: $A = \{XSBY\} \cup A_1 \cup A_2 \cup A_R$, where

$A_1 = \{Z_1vY_i : \exists u \to va_i \in P\} \cup \{Z_1Y_i : \exists u \to \epsilon \in P\} \cup$
$\{Z_2Y_i, Z_3Y_j, Z_4Y'_j, Z_5Y''_j, Z_6Y, X'Z_8, X''Z_9\} \cup \{Z'_mR_1 (1 \le m \le 7), R_1Z'_8, R_1Z'_9\}$.

$A_2 = \{X_ia_iZ'_{11}, X_jZ'_{12}, X'_jZ'_{13}, X''_jZ'_{14}, XZ'_{15}, Z'_{17}Y', Z'_{18}Y''\} \cup$
$\{Z_{17}R_2, Z_{18}R_2, R_2Z_{m(11\le m\le 16)}\}$.

$A_R = \{C_1^0Z_{16}, C_1^2Z_{16}^2, RZ_{16}^1, RZ_{16}^3, Z_7^1C_2^0,$
$Z_7^3C_2^2, Z_7^2R, Z_7R, D_1^0Z_D^1, D_1^2Z_D^3, RZ_D^2, RZ_D, Z_EE_2^0, Z_E^2E_2^2, Z_E^1R, Z_E^3R\}$.

Axioms from A_1 and A_2 are used to simulate the behaviour of the first and, respectively, second component. Axioms from A_R are used to obtain the result.

Simulation of rules of the first component of the original system (the original

The evolution of $X_3a_3wY_3$

TVDH2 system D_G^2

Odd step (we are in the component 1)

We have Z_4Y_2' and $X_2'Z_{13}$.

$$\begin{array}{c|c} X_3a_3w & Y_3 \\ \hline Z_4 & Y_2' \end{array} \vdash \frac{X_3a_3wY_2'}{Z_4Y_3}$$

$$\begin{array}{c|c} Z_4Y_2' & \varepsilon \\ \hline Z_4Y_2' & \varepsilon \end{array} \vdash \frac{Z_4Y_2'}{Z_4Y_2'}$$

$$\begin{array}{c|c} X_2'Z_{13} & \varepsilon \\ \hline X_2'Z_{13} & \varepsilon \end{array} \vdash \frac{X_2'Z_{13}}{X_2'Z_{13}}$$

Even step (we are in the component 2)

$$\begin{array}{c|c} X_3 & a_3wY_2' \\ \hline X_2' & Z_{13} \end{array} \vdash \frac{X_2'a_3wY_2'}{X_3Z_{13}}$$

$$\begin{array}{c|c} Z_4Y_2' & \varepsilon \\ \hline Z_4Y_2' & \varepsilon \end{array} \vdash \frac{Z_4Y_2'}{Z_4Y_2'}$$

$$\begin{array}{c|c} X_2'Z_{13} & \varepsilon \\ \hline X_2'Z_{13} & \varepsilon \end{array} \vdash \frac{X_2'Z_{13}}{X_2'Z_{13}}$$

TVDH1 system D_G

Odd step

We have Z_4Y_2', $Z_4'R_1$, $X_2'Z_{13}'$, R_2Z_{13}.

$$\begin{array}{c|c} X_3a_3w & Y_3 \\ \hline Z_4 & Y_2' \end{array} \vdash \frac{X_3a_3wY_2'}{Z_4Y_3}$$

$$\begin{array}{c|c} Z_4 & Y_2' \\ \hline Z_4' & R_1 \end{array} \vdash \frac{Z_4'Y_2'}{Z_4R_1}$$

$$\begin{array}{c|c} X_2' & Z_{13}' \\ \hline R_2 & Z_{13} \end{array} \vdash \frac{X_2'Z_{13}}{R_2Z_{13}'}$$

Even step

$$\begin{array}{c|c} X_3 & a_3wY_2' \\ \hline X_2' & Z_{13} \end{array} \vdash \frac{X_2'a_3wY_2'}{X_3Z_{13}}$$

$$\begin{array}{c|c} Z_4' & Y_2' \\ \hline Z_4 & R_1 \end{array} \vdash \frac{Z_4Y_2'}{Z_4'R_1}$$

$$\begin{array}{c|c} X_2' & Z_{13} \\ \hline R_2 & Z_{13}' \end{array} \vdash \frac{X_2'Z_{13}'}{R_2Z_{13}}$$

Figure 6.5: The evolution of $X_3a_3wY_3$ in both systems.

number is given in parenthesis):

$$1(1.1) : \begin{array}{c|c} \mathbf{a} & uY \\ \hline Z_1 & vY_i \end{array} \ {\scriptstyle (\exists u\to va_i\in P)};\quad 1'(1.1') : \begin{array}{c|c} \mathbf{a} & a_iuY \\ \hline Z_1 & Y_i \end{array} \ {\scriptstyle (\exists u\to \epsilon\in P)};\quad 2(1.2) : \begin{array}{c|c} \mathbf{a} & a_iY \\ \hline Z_2 & Y_i \end{array};$$

$$3(1.9) : \begin{array}{c|c} \mathbf{a} & Y_j'' \\ \hline Z_3 & Y_j \end{array};\quad 4(1.3) : \begin{array}{c|c} \mathbf{ab} & Y_k \\ \hline Z_4 & Y_{k-1}' \end{array};\quad 5(1.4) : \begin{array}{c|c} \mathbf{a} & Y_j' \\ \hline Z_5 & Y_j'' \end{array};$$

$$6(1.7) : \begin{array}{c|c} \mathbf{a} & Y'' \\ \hline Z_6 & Y \end{array};\quad 7(1.8) : \begin{array}{c|c} \mathbf{a} & BY'' \\ \hline Z_7 & C_2^s \end{array};\quad 8(1.5) : \begin{array}{c|c} X_1 & \mathbf{ab} \\ \hline X' & Z_8 \end{array};\quad 9(1.6) : \begin{array}{c|c} X' & \mathbf{a} \\ \hline X'' & Z_9 \end{array};$$

Simulation of rules of the second component of the original system (the original number is given in parenthesis):

$$11(2.1) : \begin{array}{c|c} X & \mathbf{a} \\ \hline X_ia_i & Z_{11} \end{array};\qquad 12(2.8) : \begin{array}{c|c} X_j'' & \mathbf{a} \\ \hline X_j & Z_{12} \end{array};\qquad 13(2.2) : \begin{array}{c|c} X_k & \mathbf{ab} \\ \hline X_{k-1}' & Z_{13} \end{array};$$

$$14(2.3) : \begin{array}{c|c} X_j' & \mathbf{a} \\ \hline X_j'' & Z_{14} \end{array};\qquad 15(2.6) : \begin{array}{c|c} X'' & \mathbf{a} \\ \hline X & Z_{15} \end{array};\qquad 16(2.7) : \begin{array}{c|c} X'' & \mathbf{a} \\ \hline C_1^s & Z_{16} \end{array};$$

$$17(2.4) : \begin{array}{c|c} \mathbf{ab} & Y_1 \\ \hline Z_{17} & Y' \end{array};\qquad 18(2.5) : \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z_{18} & Y'' \end{array};$$

Creation of directing molecules from A_1^2:
Deactivation rules:

$$z1 : \begin{array}{c|c} Z_1 & vY_i \\ \hline Z_1' & R_1 \end{array} ; \quad z2 : \begin{array}{c|c} Z_2 & Y_i \\ \hline Z_2' & R_1 \end{array} ; \quad z3 : \begin{array}{c|c} Z_3 & Y_j \\ \hline Z_3' & R_1 \end{array} ; \quad z4 : \begin{array}{c|c} Z_4 & Y_j' \\ \hline Z_4' & R_1 \end{array} ;$$

$$z5 : \begin{array}{c|c} Z_5 & Y_j'' \\ \hline Z_5' & R_1 \end{array} ; \quad z6 : \begin{array}{c|c} Z_6 & Y \\ \hline Z_6' & R_1 \end{array} ; \quad z8 : \begin{array}{c|c} X' & Z_8 \\ \hline R_1 & Z_8' \end{array} ; \quad z9 : \begin{array}{c|c} X'' & Z_9 \\ \hline R_1 & Z_9' \end{array} ;$$

Activation rules:

$$z1' : \begin{array}{c|c} Z_1' & vY_i \\ \hline Z_1 & R_1 \end{array} ; \quad z2' : \begin{array}{c|c} Z_2' & Y_i \\ \hline Z_2 & R_1 \end{array} ; \quad z3' : \begin{array}{c|c} Z_3' & Y_j \\ \hline Z_3 & R_1 \end{array} ; \quad z4' : \begin{array}{c|c} Z_4' & Y_j' \\ \hline Z_4 & R_1 \end{array} ;$$

$$z5' : \begin{array}{c|c} Z_5' & Y_j'' \\ \hline Z_5 & R_1 \end{array} ; \quad z6' : \begin{array}{c|c} Z_6' & Y \\ \hline Z_6 & R_1 \end{array} ; \quad z8' : \begin{array}{c|c} X' & Z_8' \\ \hline R_1 & Z_8 \end{array} ; \quad z9' : \begin{array}{c|c} X'' & Z_9' \\ \hline R_1 & Z_9 \end{array} ;$$

Creation of directing molecules from A_2^2:
Deactivation rules:

$$z11' : \begin{array}{c|c} X_i a_i & Z_{11}' \\ \hline R_2 & Z_{11} \end{array} ; \quad z12' : \begin{array}{c|c} X_j & Z_{12}' \\ \hline R_2 & Z_{12} \end{array} ; \quad z13' : \begin{array}{c|c} X_j' & Z_{13}' \\ \hline R_2 & Z_{13} \end{array} ; \quad z14' : \begin{array}{c|c} X_j'' & Z_{14}' \\ \hline R_2 & Z_{14} \end{array} ;$$

$$z15' : \begin{array}{c|c} X & Z_{15}' \\ \hline R_2 & Z_{15} \end{array} ; \quad z17' : \begin{array}{c|c} Z_{17}' & Y' \\ \hline Z_{17} & R_2 \end{array} ; \quad z18' : \begin{array}{c|c} Z_{18}' & Y'' \\ \hline Z_{18} & R_2 \end{array} ;$$

Activation rules:

$$z11 : \begin{array}{c|c} X_i a_i & Z_{11} \\ \hline R_2 & Z_{11}' \end{array} ; \quad z12 : \begin{array}{c|c} X_j & Z_{12} \\ \hline R_2 & Z_{12}' \end{array} ; \quad z13 : \begin{array}{c|c} X_j' & Z_{13} \\ \hline R_2 & Z_{13}' \end{array} ; \quad z14 : \begin{array}{c|c} X_j'' & Z_{14} \\ \hline R_2 & Z_{14}' \end{array} ;$$

$$z15 : \begin{array}{c|c} X & Z_{15} \\ \hline R_2 & Z_{15}' \end{array} ; \quad z17 : \begin{array}{c|c} Z_{17} & Y' \\ \hline Z_{17}' & R_2 \end{array} ; \quad z18 : \begin{array}{c|c} Z_{18} & Y'' \\ \hline Z_{18}' & R_2 \end{array} ;$$

Rules for the result:

$$x.1 : \begin{array}{c|c} \mathbf{a} & C_2^s \\ \hline Z_E E_2^s & \varepsilon \end{array} ; \quad x.2 : \begin{array}{c|c} C_1^s & \mathbf{a} \\ \hline \varepsilon & D_1^s Z_D \end{array} ;$$

$$r.1 : \begin{array}{c|c} C_1^s & Z_{16} \\ \hline R & Z_{16}^1 \end{array} ; \quad r.2 : \begin{array}{c|c} C_1^s & Z_{16}^1 \\ \hline R & Z_{16}^2 \end{array} ; \quad r.3 : \begin{array}{c|c} C_1^s & Z_{16}^2 \\ \hline R & Z_{16}^3 \end{array} ; \quad r.4 : \begin{array}{c|c} C_1^s & Z_{16}^3 \\ \hline R & Z_{16} \end{array} ;$$

$$r.5 : \begin{array}{c|c} D_1^s & Z_D \\ \hline R & Z_D^1 \end{array} ; \quad r.6 : \begin{array}{c|c} D_1^s & Z_D^1 \\ \hline R & Z_D^2 \end{array} ; \quad r.7 : \begin{array}{c|c} D_1^s & Z_D^2 \\ \hline R & Z_D^3 \end{array} ; \quad r.8 : \begin{array}{c|c} D_1^s & Z_D^3 \\ \hline R & Z_D \end{array} ;$$

$$r.9 : \begin{array}{c|c} Z_7 & C_2^s \\ \hline Z_7^1 & R \end{array} ; \quad r.10 : \begin{array}{c|c} Z_7^1 & C_2^s \\ \hline Z_7^2 & R \end{array} ; \quad r.11 : \begin{array}{c|c} Z_7^2 & C_2^s \\ \hline Z_7^3 & R \end{array} ; \quad r.12 : \begin{array}{c|c} Z_7^3 & C_2^s \\ \hline Z_7 & R \end{array} ;$$

$$r.13 : \begin{array}{c|c} Z_E & E_2^s \\ \hline Z_E^1 & R \end{array} ; \quad r.14 : \begin{array}{c|c} Z_E^1 & E_2^s \\ \hline Z_E^2 & R \end{array} ; \quad r.15 : \begin{array}{c|c} Z_E^2 & E_2^s \\ \hline Z_E^3 & R \end{array} ; \quad r.16 : \begin{array}{c|c} Z_E^3 & E_2^s \\ \hline Z_E & R \end{array} ;$$

The functioning and the result

It is easy to see that each main rule of D_G^2 is directly simulated by a rule of D_G. Now we shall concentrate on the obtention of the result in our system. In the initial system the molecules Z' and Z'' were used to delete the brackets X and Y. One of these molecules was appearing during an even step, while another was appearing during an odd step. It is intuitively clear that we need directing molecules with the period of four in order to simulate this behaviour. At first we change the brackets X and Y to C_1^s and C_2^s in order to assure that the corresponding molecule will participate only in the production of the result. After that we use directing molecules $C_1^s Z_{16}$ and $Z_7 C_2^s$ with the period of four permitting to eliminate the brackets C_1^s and C_2^s. The index s which represents the step number modulo 4 is used in order to decrease the number of molecules present in the system. This reduction of the number of molecules permitted to check the constructed system by the TVDH systems simulator `TVDHsim`, see [49], developed by the author during his master thesis. We also note that this software simulator was widely used for the construction of this system.

Final remarks

It is clear that by simulating D_G^2 in a way described above we obtain all words of $L(D_G^2)$. Is is also easy to see that we do not produce other words, because in order to produce a word in our system we need to use rules 7 and 16, and this corresponds to the production of a word in D_G^2.

Now we shall speak about the complexity of the simulation. Each rule of D_G^2 except 1.8, 1.10, 2.7 and 2.9 is simulated in one step of computation. Therefore we use one step in our system in order to simulate one step of D_G^2. Rules 1.8, 1.10, 2.7 and 2.9 are used in D_G^2 to obtain the resulting terminal string. In D_G, we need two more steps to obtain the same result.

We would also like to note that at each step the set of current words is different from the set of previous words, *i.e.*, $L_k \cap L_{k+1} = \emptyset$. This is an important property of the system that will be used later.

$\square$

6.4 Conclusions

We introduced in this chapter the method of directing molecules. We also gave two examples of application of this method by showing how it is possible to decrease the number of components in TVDH and ETVDH systems. Now we shall present in an informal way an abstraction of the method of directing molecules.

Let S be a system based on splicing which is distributed, *i.e.* it contains several components. Let us also suppose that S permits the elimination of molecules in the following sense. In order to eliminate a molecule, we either eliminate it from the system, or we send it out of the scope of rules of its component. For example, the elimination of a molecule in the case of TVDH systems is done directly, while in the case of test tube systems which will be introduced in the next chapter we eliminate a molecule from a component by sending it to another component. Let $R_1, \dots, R_n$ be the rules of the system distributed over n components. Let $A_1, \dots, A_n$ be the axioms which correspond to components. Now let $R'_1 \subseteq R_1, \dots, R'_n \subseteq R_n$ and $A'_1 \subseteq A_1, \dots, A'_n \subseteq A_n$ such that for each axiom $x = x_1u_3u_4x_2$ of A'_i, there is a rule r in R'_i such that x matches r: $r = u_1\#u_2\$u_3\#u_4$ and there is also a rule $x\#\varepsilon\$x\#\varepsilon$ in $R_i \setminus R'_i$. By using the method of directing molecules, we can refine the control of the system S. In order to do this, we transform axioms from A'_i into directing molecules which appear with a certain period. The creation of these molecules during some step of the computation must be done by one of the following:

- Creation by enumeration and elimination.
- Creation directly by the control.
- Creation by nesting.

The first possibility is realisable if it is possible to eliminate certain molecules at certain moments of the computation. The directing molecule has a number, its state, and this state is incremented, at each step, modulo k. In order to do this, we must be able to increment the state of the molecule as well as to eliminate the molecule in the previous state. The proof of Theorem 6.2.1 is an example of usage of this technique.

The second possibility permits to implement the apparition of directing molecules during certain steps of the computation directly by the control of the system. In the next chapter we give several proofs which use this technique.

We can use the third possibility if we deal with models where it is possible to have several iterative applications of splicing rules during the same step. We can have in this case two levels of directing molecules. Directing molecules of the first level are created by the mean of directing molecules of the second level. The last ones may be created by one of the techniques above. Theorem 7.2.2 in the next chapter is an example which uses this technique.

If the creation of directing molecules related to a step of computation may be implemented, then the resulting system will have an additional control related to periods of directing molecules. This control will permit the application of certain rules at specific moments. Examples of application of this technique to different types of distributed systems based on splicing may be found in the following chapters.

Chapter 7

Test tube systems with alternating filters

In this chapter we introduce another distributed model of computation based on splicing: test tube systems. This is one of the first distributed models based on splicing which was considered and which was heavily studied. Test tube systems are inspired by H systems and by formal distributed grammars. In fact, these systems are composed from a finite number of tubes which are H systems with some additional filters. The functioning of a such system consists from two steps: a step of computation and a step of communication which are synchronised. The distribution of results of the computation during the step of communication is made by specific rules inspired from similar rules of distributed grammars. Test tube systems have a complex behaviour and a lot of results concerning their computational power are known. For example, three tubes suffice to generate all recursively enumerable languages. From the other side, the computational power of test tube systems with two tubes is still open. This problem stimulated modifications of test tube systems. These modifications which are made in the communication process permit to obtain the computational power of a Turing machine with two tubes only. We also propose a new variant of test tube systems: test tube systems with alternating filters, or TTF systems. In this variant the process of filtering is changed and this permits to obtain a big computational power with two tubes only. Moreover, it is possible to place rules in the first tube in a way that permits to have no more rules in the second tube which is used only as a garbage collector. We also show how it is possible to apply the method of directing molecules to these systems in spite of the fact that they do not have elimination of molecules.

7.1 Test tube systems

Definition 7.1.1. [5] A *test tube system*, or a communicating distributed H system, with n tubes is the following construction:

$$\Delta = (V, T, (A_1, R_1, F_1), \dots, (A_n, R_n, F_n)),$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet, $A_i \subseteq V^*$ are finite sets of axioms, R_i, are finite sets of splicing rules and $F_i \subseteq V$, are the filters which are finite sets of letters, $F_i \subseteq V$, $1 \leq i \leq n$.

Each triplet (A_i, R_i, F_i), $1 \leq i \leq n$ is called a *test tube*, or simply tube, of Δ. We can also call it *component*.

A *configuration* of the system is the n-tuple $(L_1, \ldots, L_n)$, $L_i \subseteq V^*$, $1 \leq i \leq n$; L_i is also called the *contents* of the i^{th} tube. We define the transition between two configurations $(L_1, \ldots, L_n)$ et $(L'_1, \ldots, L'_n)$ as follows:

$(L_1, \ldots, L_n) \Rightarrow (L'_1, \ldots, L'_n)$ iff

$$L'_i = \left(\bigcup_{j=1}^{n} \sigma_j^*(L_j) \cap F_i^* \right) \cup \left(\sigma_i^*(L_i) \cap \left(V^* - \bigcup_{k=1}^{n} F_k^* \right) \right)$$

where $\sigma_i = (V, R_i)$ is the splicing scheme associated to the i^{th} tube of the system.

In words, this means that the computation consists from two sub-steps which are repeated iteratively. During the first sub-step, the one of computing, each tube evolves as an ordinary H system. During the second sub-step, the one of communication, molecules which are present in each tube are sent to all tubes. Molecules which can pass a filter of a tube, *i.e.* which are composed only from the letters which form the set of the filter, remain in the corresponding tube and form the initial language for the next step of computation. If a molecule can pass several filters, each of corresponding tubes receives a copy of this molecule. The molecules which cannot pass any filter remain in the tube where they were produced. This protocol is inspired from the theory of formal distributed grammars and it may change when we consider different variants of test tube systems.

The language generated by a test tube system Δ consists of all words over the terminal alphabet produced by the system at some step and which are in the first tube.

$L(\Delta) = \{w \in T^* | w \in L_1, \exists L_1, \ldots, L_n \subseteq V^* : (A_1, \ldots, A_n) \overset{*}{\Rightarrow} (L_1, \ldots, L_n)\}$.

We note that by definition test tube systems do not have elimination of molecules, *i.e.* once produced they exist forever. But a molecule may leave the tube in which it was produced if it can pass a filter of another tube and in the same time it cannot pass the filter of the tube in which it was produced.

We denote by TT_n the family of languages produced by test tube systems having at most n tubes.

It is easy to see that test tube systems with one tube are extended H systems. So, we obtain the following result:

Theorem 7.1.1. *$TT_1 = REG$.*

If we consider more tubes, then we obtain a bigger computational power. Three tubes are sufficient in order to generate all recursively enumerable languages, see [34].

Theorem 7.1.2. *[34] $TT_3 = RE$.*

As for test tube systems with two components, the problem of their computational power is still open, but as shows the following example they can generate non-regular languages.

Example 7.1.1.

We take the example given in [5] et [44].
Consider the following system.

$$\begin{aligned}
&\Gamma = (\{a,b,c,d,e\}, \{a,b,c\}, (A_1,R_1,V_1), (A_2,R_2,V_2)),\\
&A_1 = \{cabc, ebd, dae\},\\
&R_1 = \{b\#c\$e\#bd,\ da\#e\$c\#a\},\\
&V_1 = \{a,b,c\},\\
&A_2 = \{ec, ce\},\\
&R_2 = \{b\#d\$e\#c,\ c\#e\$d\#a\},\\
&V_2 = \{a,b,d\}.
\end{aligned}$$

The only possible splicings in the first tube are the following:

$$\begin{aligned}
&(x_1b|c, e|bd) \vdash_1 (x_1bbd, ec), \quad \text{pour } x_1 \in \{a,b,c,d,e\}^*,\\
&(da|e, c|ax_2) \vdash_2 (daax_2, ce), \quad \text{pour } x_2 \in \{a,b,c,d,e\}^*.
\end{aligned}$$

We see that these application permit to add an a, respectively a b, to words of form cax_2, respectively x_1bc, and they replace in the same time c by d. Now let ca^nb^mc, $n,m \geq 1$ be a word in the first component (initially $n = m = 1$). By using rules of the first tube we can produce the word $da^{n+1}b^{m+1}d$ which is sent to the second tube. We note that both rules of R_1 must be used, otherwise the word will contain the symbol c and it cannot pass the filter of the second tube.

In the second tube we have the following applications:

$$\begin{aligned}
&(\alpha a^ib^j|d, e|c) \vdash_1 (\alpha a^ib^jc, ed), \quad \alpha \in \{c,d\},\\
&(c|e, d|a^ib^j\alpha) \vdash_2 (ca^ib^j\alpha, de), \quad \alpha \in \{c,d\}.
\end{aligned}$$

Only the word ca^ib^jc is sent to the first tube.
The process above can be iterated and, by consequent, we obtain:

$$L(\Gamma) = \{ca^nb^nc \mid n \geq 1\},$$

which is a context-free language.

Therefore, we have the following result:

Proposition 7.1.3. *$TT_2 \setminus REG \neq \emptyset$.*

The authors in [5] and [44] suppose that the family of languages generated by test tube systems with two tubes is included in the family of context-free languages.

7.2 Test tube systems with alternating filters

Definition 7.2.1. A *test tube system with m alternating filters* and n tubes is the following $n+2$-tuple:

$$\Gamma = \left(V, T, \left(A_1, R_1, F_1^{(1)}, \ldots, F_1^{(m)}\right), \ldots, \left(A_n, R_n, F_n^{(1)}, \ldots, F_n^{(m)}\right)\right),$$

where V, T, A_i and R_i are defined as in the case of test tube systems. Now instead of a filter F_i for each tube i, we have an m-tuple of filters $F_i^{(r)}$, $1 \leq r \leq m$, where each filter $F_i^{(r)}$ is defined as in the case above.

At each step $k \geq 1$, only the filter $F_i^{(r)}$, $r = (k-1) \pmod{m} + 1$, is active, *i.e.* to be used, for the tube i.

We define the transition between two configurations as follows:

$$\left(L_1^{(1)}, \ldots, L_n^{(1)}\right) = (A_1, \ldots, A_n)$$

$$L_i^{(k+1)} = \left(\bigcup_{j=1}^{n} \sigma_j^* \left(L_j^{(k)}\right) \cap F_i^{(r)*}\right) \cup \left(\sigma_i^* \left(L_i^{(k)}\right) \cap \left(V^* - \bigcup_{k=1}^{n} F_k^{(r)*}\right)\right)$$

where $\sigma_i = (V, R_i)$ is the splicing scheme associated to the i^{th} tube of the system. The obtained systems are very similar to test tube systems. The only difference is in the fact that instead of an unique filter F_i we have an m-tuple of filters $F_i^{(1)} \ldots F_i^{(m)}$ and the filter to be used depend on the number of steps which were done.

The language generated by Γ is the following:
$L(\Gamma) = \{w \in T^* | \exists k : w \in L_1^{(k)}\}$.

We say that Γ generates the empty language if for all k, the set $L_1^{(k)}$ does not contain any terminal word.

We denote by $TTF_{n,m}$ the family of languages generated by test tube systems with alternating filters having at most n tubes and m filters.

7.2.1 The computational power of TTF$_{2,2}$

In this section we shall present a TTF system with two components and two filters and which simulates a type-0 grammar. Moreover, both filters of the first component coincide.

Theorem 7.2.1. *Let $G = (N, T, P, S)$ be a type-0 grammar. Then, there is a test tube system with alternating filters having two components and two filters, Γ, defined by $\Gamma = \left(V, T, \left(A_1, R_1, F_1^{(1)}, F_1^{(2)}\right), \left(A_2, R_2, F_2^{(1)}, F_2^{(2)}\right)\right)$, which simulates G and $L(\Gamma) = L(G)$. Moreover, $F_1^{(1)} = F_1^{(2)}$.*

Proof. We construct Γ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(B = a_n)$ and $B \notin N \cup T$.

In what follows we assume that:
$1 \leq i \leq n, \mathbf{a} \in N \cup T \cup \{B\}, \mathbf{b} \in N \cup T \cup \{B\} \cup \{\diamond\}, \gamma \in \{\alpha, \beta\}$.

The alphabet V is defined by:
$V = N \cup T \cup \{B\} \cup \{\alpha, \beta\} \cup \{\diamond\} \cup \{X, Y, X_\alpha, X_\beta, X'_\alpha, Y_\alpha, Y'_\alpha, Y_\beta, Z, Z'\}$.

The terminal alphabet T is the same as for the grammar G.

The components of the system are defined as follows.

Tube I:

Rules of R_1:

$$1.1: \begin{array}{c|c} \varepsilon & uY \\ \hline Z & vY \end{array}; \qquad 1.2: \begin{array}{c|c} \mathbf{a} & a_iY \\ \hline Z & \beta\alpha^{i-1}Y'_\alpha \end{array}; \qquad 1.3: \begin{array}{c|c} \mathbf{a} & BY \\ \hline Z & \diamond Y_\beta \end{array};$$

$$1.4: \begin{array}{c|c} X & \mathbf{a} \\ \hline X_\alpha\beta & Z \end{array}; \qquad 1.5: \begin{array}{c|c} X & \mathbf{a} \\ \hline X_\beta\beta\beta & Z \end{array}; \qquad 1.6: \begin{array}{c|c} \mathbf{b} & \beta Y_a \\ \hline Z & Y_\beta \end{array};$$

$$1.7: \begin{array}{c|c} X'_\alpha & \alpha \\ \hline X_\alpha & Z \end{array}; \qquad 1.8: \begin{array}{c|c} X'_\alpha & \alpha \\ \hline X_\beta\beta & Z \end{array}; \qquad 1.9: \begin{array}{c|c} \gamma & \alpha Y_\alpha \\ \hline Z & Y'_\alpha \end{array};$$

Filters:
$F_1 = F_1^{(1)} = F_1^{(2)} = N \cup T \cup \{B\} \cup \{\alpha, \beta\} \cup \{\diamond\} \cup \{X, Y, X'_\alpha, Y_\alpha\}$.

Axioms of A_1:
$\{XBSY, Z\beta\alpha^iY'_\alpha, Z \diamond Y_\beta, X_\alpha\beta Z, X_\beta\beta\beta Z, ZY_\beta, X_\alpha Z, X_\beta\beta Z, ZY'_\alpha\} \cup$
$\{ZvY : \exists u \to v \in P\}$.

Tube II:

Rules of R_2:

$$2.1: \begin{array}{c|c} \gamma & Y'_\alpha \\ \hline Z & Y_\alpha \end{array}; \qquad 2.2: \begin{array}{c|c} X_\alpha & \gamma \\ \hline X'_\alpha\alpha & Z \end{array}; \qquad 2.3: \begin{array}{c|c} a_i & Y_\beta \\ \hline Z & Y \end{array};$$

$$2.4: \begin{array}{c|c} X_\beta\beta\alpha^i\beta & \mathbf{a} \\ \hline Xa_i & Z \end{array}; \qquad 2.5: \begin{array}{c|c} X_\beta\beta\beta & \mathbf{a} \\ \hline \varepsilon & Z' \end{array}; \qquad 2.6: \begin{array}{c|c} \mathbf{a} & \diamond Y_\beta \\ \hline Z' & \varepsilon \end{array};$$

Filters:
$F_2^{(1)} = N \cup T \cup \{B\} \cup \{\alpha, \beta\} \cup \{\diamond\} \cup \{X_\alpha, Y'_\alpha\}$.
$F_2^{(2)} = N \cup T \cup \{B\} \cup \{\alpha, \beta\} \cup \{\diamond\} \cup \{X_\beta, Y_\beta\}$.

Axioms of A_2:
$\{ZY_\alpha, X'_\alpha\alpha Z, ZY, Xa_iZ, Z'\}$.

We affirm that $L(\Gamma) = L(G)$.

We shall prove this assertion in the following way. First we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(\Gamma)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. By consequent our assertion will be proved.

Our simulation is based on the "rotate-and-simulate" method, see Section 4.1.2. We use a technique similar to the one used in [34, 42, 44], see also Theorems 4.3.1 and 5.2.2. We rotate the word Xwa_iY in three steps. First we encode a_i into $\beta\alpha^i$ and we obtain $X_\alpha\beta w\beta\alpha^{i-1}Y'_\alpha$. After that we transfer α from the right end of the word to its left end. Finally we decode $\beta\alpha^i\beta$ into a_i and we obtain Xa_iwY. More exactly, we start with the word Xwa_iY in the first tube. The second tube receives $X_\alpha\beta w\beta\alpha^{i-1}Y'_\alpha$ from the first component. From this moment the system works in a cycle where one α is inserted at the left end of the word and in the same time one α is erased from its right end. The cycle stops when there is no more α at the right end. After this we obtain the word $X_\beta\beta\alpha^i\beta wY_\beta$ in the second component and we can decode a_i obtaining Xa_iwY.

If we have the word $XwBY$, then we can take off symbols X, Y and B producing w which will be a part of the result if it is terminal.

Notations

We note that the bottom part of each rule represent a molecule which is already present in the corresponding tube. Similarly, one of results of the splicing will contain Z and will not participate any more to the production of the result. Therefore we can omit these molecules and write:

$Xwa_iY \underset{1.2}{\longrightarrow} Xw\beta\alpha^{i-1}Y'_\alpha$ instead of

$(Xw|a_iY, Z|\beta\alpha^{i-1}Y'_\alpha) \vdash_{1.2} (Xw\beta\alpha^{i-1}Y'_\alpha, Za_iY)$

where by $|$ we highlighted the splicing sites. In what follows we mark molecules which can evolve in the same tube with ⓗ and we mark molecules which must be send into the first, respectively second, tube with ①, respectively ②. We also write $m\uparrow$ if the molecule m cannot enter any rule of the tube in which it is situated and if in the same time it cannot be sent to another tube.

We shall show the evolution of words of form Xwa_iY. We indicate the splicing rules which are used as well as the resulting words. We note that, due to the parallelism, in the same time in our system there can be several molecules of this form or intermediate forms which evolve in parallel. These molecules do not interact with each other, so we shall concentrate only on a single evolution of this type.

Rotation

We show how to rotate the word Xwa_iY which is in the first tube.

Step 1.

Tube I.

Xwa_iY ⓗ $\underset{1.2}{\longrightarrow} Xw\beta\alpha^{i-1}Y'_\alpha$ ⓗ $\underset{1.4}{\longrightarrow} X_\alpha\beta w\beta\alpha^{i-1}Y'_\alpha$ ②.

We can use these two rules in opposite order which gives the same result. We can also use rule 1.5 instead of the last application. This gives us:

$Xw\beta\alpha^{i-1}Y'_\alpha$ ⓗ $\underset{1.5}{\longrightarrow} X_\beta\beta\beta w\beta\alpha^{i-1}Y'_\alpha\uparrow$.

We can also have the following application:

Xwa_iY ⓗ $\underset{1.5}{\longrightarrow} X_\beta\beta\beta wa_iY$ ⓗ.

The molecule $X_\alpha\beta w\beta\alpha^{i-1}Y'_\alpha$ is sent to the second tube because it can pass the filter $F_2^{(1)}$; all other molecules cannot pass any filter of the second tube.

As we noted before there are other applications of splicing rules on other molecules but they are not relevant for our evolution. We shall not mention this fact in the future.

Tube II.

As we noted before there are other applications of splicing rules on other molecules but they are not relevant for our evolution. We shall not mention this fact in the future.

`Step 2.`

Tube II.

$X_\alpha \beta w \beta \alpha^{i-1} Y'_\alpha \textcircled{h} \xrightarrow[2.1]{} X_\alpha \beta w \beta \alpha^{i-1} Y_\alpha \textcircled{h} \xrightarrow[2.2]{} X'_\alpha \alpha \beta w \beta \alpha^{i-1} Y_\alpha \textcircled{1}.$

Only molecule $X'_\alpha \alpha \beta w \beta \alpha^{i-1} Y_\alpha$ can pass filter F_1.

In this way we transferred one α from the right end to the left end.

`Step 3.`

Tube I.

$X'_\alpha \alpha \beta w \beta \alpha^{i-1} Y_\alpha \textcircled{h} \xrightarrow[1.9]{} X'_\alpha \alpha \beta w \beta \alpha^{i-2} Y'_\alpha \textcircled{h} \xrightarrow[1.7]{} X_\alpha \alpha \beta w \beta \alpha^{i-2} Y'_\alpha \textcircled{2}.$

We can also have the following applications:

$X'_\alpha \alpha \beta w \beta \alpha^{i-1} Y_\alpha \textcircled{h} \xrightarrow[1.8]{} X_\beta \beta \alpha \beta w \beta \alpha^{i-1} Y_\alpha \textcircled{h} \xrightarrow[1.9]{} X_\beta \beta \alpha \beta w \beta \alpha^{i-2} Y'_\alpha \uparrow.$

Only the word $X_\alpha \alpha \beta w \beta \alpha^{i-1} Y'_\alpha$ can pass the filter $F_2^{(1)}$.

`Step 4.`

Tube II.

$X_\alpha \alpha \beta w \beta \alpha^{i-2} Y'_\alpha \textcircled{h} \xrightarrow[2.1]{} X_\alpha \alpha \beta w \beta \alpha^{i-2} Y_\alpha \textcircled{h} \xrightarrow[2.2]{} X'_\alpha \alpha \alpha \beta w \beta \alpha^{i-2} Y_\alpha \textcircled{1}.$

Only the molecule $X'_\alpha \alpha \alpha \beta w \beta \alpha^{i-2} Y_\alpha$ can pass the filter F_1.

In this way we transferred one more α from the right end to the left end. We can continue in this way until we obtain for the first time $X'_\alpha \alpha^i \beta w \beta Y_\alpha$ in the second tube at some step `p`. This molecule is communicated to the first tube where we can use the rule 1.6 and we complete the rotation of a_i.

`Step p+1.`

Tube I.

$X'_\alpha \alpha^i \beta w \beta Y_\alpha \textcircled{h} \xrightarrow[1.6]{} X'_\alpha \alpha^i \beta w Y_\beta \textcircled{h} \xrightarrow[1.8]{} X_\beta \beta \alpha^i \beta w Y_\beta \textcircled{2}.$

We can also have the following applications:

$X'_\alpha \alpha^i \beta w \beta Y_\alpha \textcircled{h} \xrightarrow[1.7]{} X_\alpha \alpha^i \beta w \beta Y_\alpha \textcircled{h} \xrightarrow[1.6]{} X_\alpha \alpha^i \beta w Y_\beta \uparrow.$

Only $X_\beta \beta \alpha^i \beta w Y_\beta$ can pass the filter $F_2^{(2)}$. We also note that `p+1` is an odd number. Therefore, the molecule $X_\beta \beta \alpha^i \beta w Y_\beta$ will remain for one more step in the first component, because the filter used in the second component during odd steps is $F_2^{(1)}$. During the next step, `p+2`, this molecule will not change and it will be communicated to the second tube because the new filter $F_2^{(2)}$ permit its passage.

`Step p+3.`

Tube II.

$X_\beta \beta \alpha^i \beta w Y_\beta \textcircled{h} \xrightarrow[2.4]{} X a_i w Y_\beta \textcircled{h} \xrightarrow[2.3]{} X a_i w Y \textcircled{1}.$

The word $X a_i w Y$ is sent to the first tube.

In this way we competed the rotation of a_i.

Simulation of grammar productions

If we have a molecule $XwuY$ in the first tube and there is a rule $u \to v \in P$ then we can apply rule 1.1 in order to simulate the corresponding production of the grammar.

Obtention of the result

Now we shall show how we can obtain a result. If we have a molecule of type $XwBY$, then we can perform a rotation of B as described above. We can also take off X and Y. In this way, if $w \in T^*$, then w will be a part of the result. Now we shall show how to do this. Suppose that $XwBY$ appears for the first time in tube 1 at some step q (it is easy to check that q is even).

Step q.
Tube 1.
$XwBY \textcircled{h} \underset{1.3}{\longrightarrow} Xw \diamond Y_\beta \textcircled{h} \underset{1.5}{\longrightarrow} X_\beta\beta\beta w \diamond Y_\beta \textcircled{2}$.
We can also have the following application:
$Xw \diamond Y_\beta \textcircled{h} \underset{1.4}{\longrightarrow} X_\alpha\beta w \diamond Y_\beta \uparrow$.
The word $X_\beta\beta\beta w \diamond Y_\beta$ can pass the filter $F_2^{(2)}$.
Step q+1.
Tube II.
$X_\beta\beta\beta w \diamond Y_\beta \textcircled{h} \underset{2.5}{\longrightarrow} w \diamond Y_\beta \textcircled{h} \underset{2.6}{\longrightarrow} w \textcircled{1}$.
If $w \in T^*$, then it will be a part of the result. □

7.2.2 TTF$_{2,4}$ with garbage collection

In this section we shall present a TTF system having two components and four filters. Moreover, this system contains no rules in the second component, therefore it is used only as a garbage collector. We also add that all filters in the second tube are the same.

Theorem 7.2.2. *Let $G = (N, T, P, S)$ be a type-0 grammar. Then, there is a test tube system with alternating filters having two components and four filters, Γ, defined by $\Gamma = \Big(V, T, \Big(A_1, R_1, F_1^{(1)}, \ldots, F_1^{(4)}\Big), \Big(A_2, R_2, F_2^{(1)}, \ldots, F_2^{(4)}\Big)\Big)$ which simulates G and $L(\Gamma) = L(G)$. Moreover, the second component of Γ has no rules, i.e. $R_2 = \emptyset$, and also $F_2^{(1)} = \cdots = F_2^{(4)}$.*

Proof. We construct Γ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(B = a_n)$ and $B \notin N \cup T$.

In what follows we assume that:
$1 \le i \le n, 1 \le j \le 4, \mathbf{a} \in N \cup T \cup \{B\}, \mathbf{b} \in N \cup T \cup \{B\} \cup \{\diamond\}, \gamma \in \{\alpha, \beta\}$.
Also let $\mathcal{V} = N \cup T \cup \{B\} \cup \{\alpha, \beta\} \cup \{\diamond\}$.

The alphabet V is defined by:
$V = \mathcal{V} \cup \{X, Y, X_\alpha, X_\beta, X'_\alpha, Y_\alpha, Y'_\alpha, Y_\beta, Z, Z', Z_j, R_j, L_j\}$.

The terminal alphabet T is the same as for the formal grammar G.

The components of the system are defined as follows.

Tube I:

Rules of R_1:

$1.1.1 : \begin{array}{c|c} \varepsilon & uY \\ \hline R_1 & vY \end{array}$; $\quad 1.1.2 : \begin{array}{c|c} X & \mathbf{a} \\ \hline X_\alpha\beta & L_1 \end{array}$; $\quad 1.1.3 : \begin{array}{c|c} X & \mathbf{a} \\ \hline X_\beta\beta\beta & L_1 \end{array}$;

$1.1.4 : \begin{array}{c|c} \mathbf{a} & a_iY \\ \hline R_1 & \beta\alpha^{i-1}Y'_\alpha \end{array}$; $\quad 1.1.5 : \begin{array}{c|c} \mathbf{a} & BY \\ \hline R_1 & \diamond Y_\beta \end{array}$;

$1.2.1 : \begin{array}{c|c} \gamma & Y'_\alpha \\ \hline R_2 & Y_\alpha \end{array}$; $\quad 1.2.2 : \begin{array}{c|c} X_\alpha & \gamma \\ \hline X'_\alpha\alpha & L_2 \end{array}$;

$1.3.1 : \begin{array}{c|c} X'_\alpha & \alpha \\ \hline X_\alpha & L_3 \end{array}$; $\quad 1.3.2 : \begin{array}{c|c} X'_\alpha & \alpha \\ \hline X_\beta\beta & L_3 \end{array}$; $\quad 1.3.3 : \begin{array}{c|c} \gamma & \alpha Y_\alpha \\ \hline R_3 & Y'_\alpha \end{array}$;

$1.3.4 : \begin{array}{c|c} \mathbf{b} & \beta Y_a \\ \hline R_3 & Y_\beta \end{array}$;

$1.4.1 : \begin{array}{c|c} a_i & Y_\beta \\ \hline R_4 & Y \end{array}$; $\quad 1.4.2 : \begin{array}{c|c} X_\beta\beta\alpha^i\beta & \mathbf{a} \\ \hline Xa_i & L_4 \end{array}$; $\quad 1.4.3 : \begin{array}{c|c} X_\beta\beta\beta & \mathbf{a} \\ \hline \varepsilon & Z'L_4 \end{array}$;

$1.4.4 : \begin{array}{c|c} \mathbf{a} & \diamond Y_\beta \\ \hline Z'L_4 & \varepsilon \end{array}$;

$1.1.1' : \begin{array}{c|c} Z_1 & vY \\ \hline R_1 & L_1 \end{array}$; $\quad 1.1.2' : \begin{array}{c|c} X_\alpha\beta & Z_1 \\ \hline R_1 & L_1 \end{array}$; $\quad 1.1.3' : \begin{array}{c|c} X_\beta\beta\beta & Z_1 \\ \hline R_1 & L_1 \end{array}$;

$1.1.4' : \begin{array}{c|c} Z_1 & \beta\alpha^iY'_\alpha \\ \hline R_1 & L_1 \end{array}$; $\quad 1.1.5' : \begin{array}{c|c} Z_1 & \diamond Y_\beta \\ \hline R_1 & L_1 \end{array}$;

$1.2.1' : \begin{array}{c|c} Z_2 & Y_\alpha \\ \hline R_2 & L_2 \end{array}$; $\quad 1.2.2' : \begin{array}{c|c} X'_\alpha\alpha & Z_2 \\ \hline R_2 & L_2 \end{array}$;

$1.3.1' : \begin{array}{c|c} X_\alpha & Z_3 \\ \hline R_3 & L_3 \end{array}$; $\quad 1.3.2' : \begin{array}{c|c} X_\beta\beta & Z_3 \\ \hline R_3 & L_3 \end{array}$; $\quad 1.3.3' : \begin{array}{c|c} Z_3 & Y'_\alpha \\ \hline R_3 & L_3 \end{array}$;

$1.3.4' : \begin{array}{c|c} Z_3 & Y_\beta \\ \hline R_3 & L_3 \end{array}$;

$1.4.1' : \begin{array}{c|c} Z_4 & Y \\ \hline R_4 & L_4 \end{array}$; $\quad 1.4.2' : \begin{array}{c|c} Xa_i & Z_4 \\ \hline R_4 & L_4 \end{array}$; $\quad 1.4.3' : \begin{array}{c|c} Z' & Z_4 \\ \hline R_4 & L_4 \end{array}$;

Filters:
$F_1^{(1)} = \mathcal{V} \cup \{X_\alpha, Y'_\alpha, R_2, L_2\}$,
$F_1^{(2)} = \mathcal{V} \cup \{X'_\alpha, Y_\alpha, R_3, L_3\}$,
$F_1^{(3)} = \mathcal{V} \cup \{X_\beta, Y_\beta, R_4, L_4\}$,
$F_1^{(4)} = \mathcal{V} \cup \{X, Y, R_1, L_1\}$.

Axioms of A_1:
$\{XBSY, X_\alpha\beta Z_1, X_\beta\beta\beta Z_1, Z_1\beta\alpha^iY'_\alpha, Z \diamond Y_\beta, Z_2Y_\alpha, X'_\alpha\alpha Z_2,$
$X_\alpha Z_3, X_\beta\beta Z_3, Z_3Y'_\alpha, ZY_\beta, Z_4Y, Xa_iZ_4, Z'Z_4, R_1L_1\} \cup$
$\{ZvY : \exists u \to v \in P\}$.

Tube II:

No rules: $R_2 = \emptyset$.

Filters:

$F_2^{(j)} = \mathcal{V} \cup \{X, Y, X_\alpha, X_\beta, X'_\alpha, Y_\alpha, Y'_\alpha, Y_\beta, Z', R_j, L_j\}$.

Axioms of A_2:

$\{R_2L_2, R_3L_3, R_4L_4\}$

The system is similar to the system constructed in the previous section. It contains the same rules which are grouped in the first tube. The rules are split into four subsets, 1.1.x to 1.4.x, which cannot be used all in the same time. In order to achieve this we use the method of directing molecules, see Chapter 6 and, in particular, Section 6.4. We use directing molecules R_jL_j which travel from one tube to another. When a molecule R_jL_j appears in the first tube it triggers the subset j of rules which can be further applied. More precisely, the bottom part of each rule is made in a special way: the site corresponds to a directing molecule which is produced only if the molecule R_jL_j corresponding to the subset is present in the first tube. For example, the rule 1.1.3 can be used only if molecule $X_\beta\beta\beta L_1$ is present. But this molecule is present only if the word R_1L_1 is present in the first tube and it is produced by applying the rule 1.1.3′ to the axiom $X_\beta\beta\beta Z_1$ and to the directing molecule R_1L_1. At the next step the word $X_\beta\beta\beta L_1$ is sent to the second tube and it is no more present in the first one.

So we have a nesting of directing molecules. Directing molecules of the first level which correspond to sites of rules are created by the mean of directing molecules of the second level, R_jL_j, which travel from one tube to another and their apparition is directed by alternating filters. The filters also act like selectors for correct molecules and the computation is made in the following way: the molecules are sent into the second component from which correct molecules are extracted into the first one where they are further transformed.

□

We observe that operations done in the first and in the third group are independent and that molecules which are used have a different form. Therefore, we can join the first and the third group as well as corresponding filters 2 and 4. Thus we obtain:

Theorem 7.2.3. *Let $G = (N, T, P, S)$ be a type-0 grammar. Then, there is a test tube system with alternating filters having two components and three filters, Γ, defined by $\Gamma = \Big(V, T, \big(A_1, R_1, F_1^{(1)}, \dots, F_1^{(3)}\big), \big(A_2, R_2, F_2^{(1)}, \dots, F_2^{(3)}\big)\Big)$ which simulates G and $L(\Gamma) = L(G)$. Moreover, the second component of Γ has no rules, i.e. $R_2 = \emptyset$, and also $F_2^{(1)} = \dots = F_2^{(3)}$.*

7.2.3 TTF$_{2,2}$ with garbage collection

In this section we shall present a TTF system with two components and two filters and which has no rules in the second component. Moreover, the two filters of each component differ only in one letter. Also the proof of this result is different: we simulate a Turing machine instead of a Chomsky grammar.

We define the coding function ϕ as follows. For any configuration w_1qw_2 of a Turing machine M we define $\phi(w_1qw_2) = Xw_1Sqw_2Y$, where X, Y and S are three symbols which are not in T.

We also define the notion of an *input* for a TTF system. An input word for a system Γ is simply a word w over the alphabet of Γ. The computation of Γ on input w is obtained by adding w to axioms of the first tube and after that by making Γ to evolve as usual.

Lemma 7.2.4. *Let $M = (Q, T, a_0, s_0, F, \delta)$ be a Turing machine and w an input. Then, there is a test tube system with alternating filters having two components and two filters $\Gamma = \left(V, T_\Gamma, \left(A_1, R_1, F_1^{(1)}, F_1^{(2)}\right), \left(A_2, R_2, F_2^{(1)}, F_2^{(2)}\right)\right)$ which, given the input $\phi(w)$, simulates M on input w,* i.e. *such that:*

1. *for any word w on which M halts in configuration w_1qw_2, the system Γ produce a unique result $\phi(w_1qw_2)$.*

2. *for any word w, on which M does not halt, the system Γ generates the empty language.*

Proof.

Let $T = \{a_0, \dots, a_{m-1}\}$, $Q = \{q_0, \dots, q_{n-1}\}$, $\mathbf{a} \in T \cup \{X\}$, $\mathbf{b}, \mathbf{d}, \mathbf{e} \in T$, $\mathbf{c} \in T \cup \{Y\}$, $\mathbf{q} \in Q$, where a_0 is the blank symbol.

We may assume without loss of generality that w_1w_2 does not contain the symbol a_0 and that M is not stationary.

We construct Γ as follows:

The alphabet V is defined by:
$V = T \cup Q \cup \{X, Y, S, S', R, L, R', L', R^R, Z_1^R, Z_X, Z_Y, R_1^L, Z_1^L, R_1', Z_1'\}$.

The terminal alphabet T is defined by:
$T_\Gamma = T \setminus \{a_0\} \cup \{X, Y, S\} \cup \{q : q \in F\}$.

The components are defined as follows:

Tube I:

Rules de R_1:

For any rule $q_ia_kRa_lq_j \in \delta$ we have the following group of 4 rules:

1.1.1.1. $\dfrac{\mathbf{a}Sq_ia_k \mid Y}{Z_Y \mid a_0Y}$, 1.1.1.2. $\dfrac{\mathbf{a} \mid Sq_ia_k\mathbf{b}}{R \mid L}$,

1.1.1.3. $\dfrac{RSq_ia_k \mid \mathbf{b}}{R_1^Ra_lS'q_j \mid Z_1^R}$, 1.1.1.4. $\dfrac{\mathbf{a} \mid L}{R_1^R \mid \mathbf{b}S'\mathbf{qd}}$, For any rule $q_ia_kLa_lq_j \in \delta$ we have the following group of 5 rules:

1.1.2.1. $\begin{array}{c|c} X & Sq_ia_k \\ \hline Xa_0 & Z_X \end{array}$, 1.1.2.2. $\begin{array}{c|c} \mathbf{b} & \mathbf{d}Sq_ia_k \\ \hline R & L \end{array}$, 1.1.2.3. $\begin{array}{c|c} R\mathbf{b}Sq_ia_k & \mathbf{c} \\ \hline R_1^LS'q_j\mathbf{b}a_l & Z_1^L \end{array}$,

1.1.2.1′. $\begin{array}{c|c} X & \mathbf{b}Sq_ia_k \\ \hline Xa_0 & Z_X \end{array}$ 1.1.2.4. $\begin{array}{c|c} \mathbf{b} & L \\ \hline R_1^L & S'\mathbf{qdec} \end{array}$,

We have also the following group of 3 rules:

1.2.1. $\begin{array}{c|c} \mathbf{ab} & S'\mathbf{qd} \\ \hline R' & L' \end{array}$, 1.2.2. $\begin{array}{c|c} R'S' & \mathbf{qb} \\ \hline R_1'S & Z_1' \end{array}$, 1.2.3. $\begin{array}{c|c} \mathbf{b} & L' \\ \hline R_1' & S\mathbf{qd} \end{array}$,

Finally, we have the following group of 3 rules:

1.3.1. $\begin{array}{c|c} \mathbf{b} & a_0Y \\ \hline Z_Y & Y \end{array}$, 1.3.2. $\begin{array}{c|c} Xa_0 & \mathbf{b} \\ \hline X & Z_X \end{array}$, 1.3.3. $\begin{array}{c|c} Xa_0 & S \\ \hline X & Z_X \end{array}$.

Filters:
$F_1^{(1)} = \{R', L, L'\}$,
$F_1^{(2)} = \{R, L, L'\} = (F_1^1 \setminus \{R'\}) \cup \{R\}$.
Axioms of A_1:
$\{R_Ya_0Y, Xa_0Z_X, R_1'SZ_1', RL\} \cup \{R_1^Ra_lS'q_jZ_1^R : \exists q_ia_kRa_lq_j \in \delta\} \cup \{R_1^LS'q_j\mathbf{b}a_lZ_1^L : \exists q_ia_kLa_lq_j \in \delta\}$.

Tube II:
No rules: R_2 is empty.
Filters:
$F_2^{(1)} = Q \cup T \cup \{X, Y, S, R, L, R', L', R_1^L, R_1^R, R_1'\}$,
$F_2^{(2)} = Q \cup T \cup \{X, Y, S', R, L, R', L', R_1^L, R_1^R, R_1'\} = (F_2^1 - \{S\}) \cup \{S'\}$.
Axioms of A_2:
$\{R'L'\}$.

The part of tape of M which contain the information is encoded in the following way: for any configuration w_1qw_2 of M we have a configuration Xw_1Sqw_2Y in system Γ. So, the tape is enclosed by X and Y and we have a marker Sq_i which marks on the tape the head position and its current state.

The system Γ simulates each step of M in 2 stages. During the first stage it simulates a step of the Turing machine and it primes the symbol S by using rules 1.1.x.2 to 1.1.x.4. We say that these rules are associated to the corresponding rule of the machine M. During the second stage the system takes off the prime from S' by using rules 1.2.x. If necessary, the tape can be extended by rules 1.1.x.1 or reduced by rules 1.3.x. The system produces a result only if M halts on the initial string w.

In order to achieve the separation in 2 stages we use two subsets of rules in the first tube as well as directing molecules RL and $R'L'$ which appear alternatively in the first and the second test tube by triggering the first (1.1.x) or the second (1.2.x) subset of rules, see also Section 7.2.2 and Chapter 6. All extra molecules which are of the wrong form are sent to the second test tube which is considered as a garbage collector because only the directing molecules RL and $R'L'$ may go from that tube to the first one.

The molecules R_Ya_0Y, $R_1^Ra_lS'q_jZ_1^R$, Xa_0Z_X, $R_1^LS'q_j\mathbf{b}a_lZ_1^L$ and $R_1'SZ_1'$ are present all the time in the tube 1. Also, all molecules containing the symbol Z with indices remain in the first tube.

We start with $Xw_1Sq_0w_2Y$ in the first tube, where $w_1q_0w_2$ is the initial configuration of M.

The simulation

We shall describe the simulation of the application of rule $q_ia_kRa_lq_j \in \delta$ to configuration $Xw_1Sq_ia_kw_2$Y. We shall omit from our description the molecules R_Ya_0Y, $R_1^Ra_lS'q_jZ_1^R$, Xa_0Z_X, $R_1^LS'q_j\mathbf{b}a_lZ_1^L$ and $R_1'SZ_1'$ which are alway present in the first tube. Similarly, molecules containing Z with indices and which do not alter the computation will be omitted from the description after one step of the computation.

We also note that the second tube may contain other molecules, obtained during the previous steps, as it works as a garbage collector. We shall omit these molecules as they do not alter our simulation.

Step 1.

Splicing:

Tube 1:

We have: $Xw_1Sq_ia_kw_2Y, RL$.

We apply the following rules which are all from group 1.1.1.x:

$(Xw_1|Sq_ia_kw_2Y, R|L) \vdash_{1.1.1.2} (Xw_1L, RSq_ia_kw_2Y)$.

$(RSq_ia_k|w_2Y, R_1^Ra_lS'q_j|Z_1^R) \vdash_{1.1.1.3} (R_1^Ra_lS'q_jw_2Y, RSq_ia_kZ_1^R)$.

$(Xw_1|L, R_1^R|a_lS'q_jw_2Y) \vdash_{1.1.1.4} (Xw_1a_lS'q_jw_2Y, R_1^RL)$.

Tube 2:

We have: $R'L'$. As it was said above we can have here other molecules obtained during previous steps of computation and which do not alter the computation. We omit them and we do not mention this in the future.

No application of rules.

Communication:

Current filters are:

$F_1^{(1)} = \{R', L, L'\}$ et $F_2^{(1)} = Q \cup T \cup \{X, Y, S, R, L, R', L', R_1^L, R_1^R, R_1'\}$.

Molecules sent from tube 1 to tube 2:

$RL, Xw_1Sq_ia_kw_2Y, Xw_1L, RSq_ia_kw_2Y, R_1^RL$.

Molecules remaining in tube 1:

$R_1^Ra_lS'q_jw_2Y, RSq_ia_kZ_1^R, Xw_1a_lS'q_jw_2Y$.

Molecules sent from tube 2 to tube 1:

$R'L'$.

Molecules remaining in tube 2:

None.

The molecule $RSq_ia_kZ_1^R$ cannot evolve further, so we shall omit it in the future. Also, the molecule $R_1^Ra_lS'q_jw_2Y$ will be sent from from the first tube to the second tube during the next step.

Step 2.

Splicing:

Tube 1:

We have: $R_1^Ra_lS'q_jw_2Y, Xw_1a_lS'q_jw_2Y, R'L'$.

We apply the following rules which are all from group 1.2.x:

$(Xw_1a_l|S'q_jw_2Y, R'|L') \vdash_{1.2.1} (Xw_1a_lL', R'S'q_jw_2Y)$.

$(R'S'|q_jw_2Y, R_1'S|Z_1') \vdash_{1.2.2} (R_1'Sq_jw_2Y, R'S'Z_1')$.

$(Xw_1a_l|L', R_1'|Sq_jw_2Y) \vdash_{1.2.3} (Xw_1a_lSq_jw_2Y, R_1'L')$.

Tube 2:

We have: $Xw_1Sq_ia_kw_2Y, RL, Xw_1L, RSq_ia_kw_2Y, R_1'L'$.

No application of rules.

Communication:

Current filters are:

$F_1^{(1)} = \{R', L, L'\}$ et $F_2^{(2)} = Q \cup T \cup \{X, Y, S', R, L, R', L', R_1^L, R_1^R, R_1'\}$.

Molecules sent from tube 1 to tube 2:

$R'L', R_1^Ra_lS'q_jw_2Y, Xw_1a_lS'q_jw_2Y, Xw_1a_lL', R'S'q_jw_2Y, R_1'L'$.

Molecules remaining in tube 1:

$R'S'Z_1, R_1'Sq_jw_2Y, Xw_1a_lSq_jw_2Y$.

Molecules sent from tube 2 to tube 1:

RL.

Molecules remaining in tube 2:

$R'L', R_1^Ra_lS'q_jw_2Y, Xw_1a_lS'q_jw_2Y, Xw_1a_lL', R'S'q_jw_2Y, R_1'L', Xw_1Sq_ia_kw_2Y, Xw_1L, RSq_ia_kw_2Y, R_1^RL$.

Similarly, the molecule $R_1'Sq_jw_2Y$ will be sent from the first tube to the second tube during the next step.

Thus we simulated the application of the rule $q_ia_kRa_lq_j$ of the machine M. It is easy to observe that if $w_2 = \varepsilon$, then we first apply the rule 1.1.1.1 in order to extend the tape to the right and after that $w_2 = a_0$. We note that the application of rules of extension or restriction of the tape produce several copies of the main molecule which codes the state of the Turing machine. These copies differ in the number of symbols a_0 situated at their ends and they do not alter the computation, even if the rules 1.1.x.4 are applied to parts issued from different molecules. We also remark that there is a only one molecule which does not contain any symbol a_0 at its ends. It is also easy to check that produced molecules which do not have a correct form ($Xw_1Sq_ia_kw_2Y$, Xw_1L, $RSq_ia_kw_2Y$, R_1^RL, $R_1^Ra_lS'q_jw_2Y$, $RSq_ia_kZ_1^R$, $Xw_1a_lS'q_jw_2Y$, $R'S'q_jw_2Y$, $R'S'Z_1$, $R_1'Sq_jw_2Y$, $Xw_1a_lS'q_jw_2Y$, Xw_1a_lL') either go to the second test tube or do not alter the computation.

For the left shift rules we proceed in a similar manner, except that we ensure that there are at least 2 symbols at the left of S.

We see that we model step by step the application of rules of M. It is easy to observe that this is the only possibility for the computation to go on as the molecule which encodes the configuration of M triggers the application of rules of Γ.

$\square$

Theorem 7.2.5. *Let $\mathcal{L} \subseteq T^*$ be a recursively enumerable language. Then, there is a test tube system with alternating filters having two components and two filters Γ, defined by $\Gamma = \Big(V, T, \Big(A_1, R_1, F_1^{(1)}, F_1^{(2)}\Big), \Big(A_2, R_2, F_2^{(1)}, F_2^{(2)}\Big)\Big)$, which generates $\mathcal{L}$, i.e. $\mathcal{L} = L(\Gamma)$. Moreover, $R_2 = \emptyset$.*

Proof. For the proof we use ideas similar to the ones used in [24] and [26].

Let $\mathcal{L} = \{w_0, w_1, \dots\}$. Then there is a Turing machine $T_{\mathcal{L}}$ which computes $w_i 0 \dots 0$ starting from 01^{i+1} where 0 is the blank symbol.

It is possible to construct such a machine in the following way. Let G be a grammar which generates $\mathcal{L}$. Then we can construct a Turing machine M which will simulate derivations in G, *i.e.*, we shall obtain all sentential forms of G. We shall simulate bounded derivations in order to deal with a possible recursion in a derivation. After deriving a new word the machine checks if this word is a terminal word and if this is the case it checks if this word is the i-th terminal word which is obtained. If the last condition holds then the machine erases everything except the word.

Moreover, this machine is not stationary, *i.e.* the head has to move at each step, and it never visits cells to the left of the 0 from the initial configuration, *i.e.* the tape is semi-infinite to the right. In the book [18] it is possible to find ideas how to satisfy these conditions. So, the machine $T_{\mathcal{L}}$ transforms configuration $q_0 0 1^{k+1}$ into $q_f w_k 0 \dots 0$.

Now starting from the machine $T_{\mathcal{L}}$, it is possible to construct a machine $T'_{\mathcal{L}}$ which computes $01^{k+2} M q'_f w_k 0 \dots 0$ starting from $q_0 0 1^{k+1}$. After that, by using Γ we cut off w_k and we continue the computation from the configuration $q_0 0 1^{k+2}$. In this way we will generate $\mathcal{L}$ word by word.

In order to simplify the proof we consider the machine $T''_{\mathcal{L}}$ which do the same thing as $T'_{\mathcal{L}}$, but at the end move its head to the right until the first zero, *i.e.* it will produce $01^{k+2} M w_k q''_f 0 \dots 0$ from $q_0 0 1^{k+1}$. We also use two additional Turing machines T_1 and T_2 having the tape alphabet of $T''_{\mathcal{L}}$ extended by the set $\{X, Y, S\}$. The first machine, T_1, moves to the left until it reaches M and then stops in the state q_f^1. The initial state of T_1 is q_s^1. The second machine, T_2, moves to the left until it reaches 0 and then stops in the state q_0. The initial state of this machine is q_s^2.

Now let us consider Γ which is very similar to the system constructed in the previous lemma. In fact, we take machine $T''_{\mathcal{L}}$ and we construct Γ for this machine as it was done above. After that we take the rules of T_1 and T_2 and we add to Γ all associated splicing rules as well as the axioms which correspond to them. Finally we add the following splicing rules:

$$x.1.\ \frac{\mathbf{a} \mid \mathbf{bd}Sq_f''0}{Z_1 \mid Sq_s^1\mathbf{bd}},\qquad x.2.\ \frac{\mathbf{e} \mid 11Sq_f^1M}{X_2 \mid \mathbf{e}Sq_s^2 11Y},\qquad x.2'.\ \frac{X_2 11Sq_f^1M \mid \mathbf{c}}{\varepsilon \mid Z'}$$

where $\mathbf{e} = \{0, 1\}$.

We also add $Xq_0 01Y$, as well as the words $Z_1Sq_s^1\mathbf{bd}$, $X_2\mathbf{e}Sq_s^2 11Y$ and Z' to axioms of the first tube and we consider T as terminal alphabet.

We shall show that we succeeded to implement the mechanism of generation of words which we presented before.

We claim that Γ generates $\mathcal{L}$. First of all this system simulates the machine $T_{\mathcal{L}}''$, so it permits to pass from the configuration $XSq_0 01^{k+1}Y$ to the configuration $X01^{k+2}Mw_kSq_f''0\dots0Y$. Now we can use the rule $x.1$. This rule cuts off the word $0\dots0Y$ which is sent to the second tube and only the molecule $X01^{k+2}Mw_k'Sq_s^1ab$ ($w_k = w_k'ab$) remains. We note that now we can simulate the work of T_1, because the only condition which must be satisfied is the presence of one symbol at the right from the head position. Therefore, starting from this moment the system Γ will simulate the work of T_1 and we will arrive to the following molecule: $X01^{k+2}Sq_f^1Mw_k$. Now the rules $x.2$ and $x.2'$ cut off w_k which will represent the result and we will obtain the molecule $X01^kSq_s^2 11Y$. In a similar way we can simulate the work of T_2 and we will arrive to configuration $XSq_0 01^{k+2}Y$. Now we can restart the computation above and to produce w_{k+1}, by consequent, we shall produce all words from $\mathcal{L}$.

□

7.3 Conclusions

In this chapter we have seen the power of elimination. Indeed, as it was shown by examples in this chapter, it suffices to add a possibility of elimination to extended H systems in order to let them to acquire a big computational power. The directing molecules also play an important role. We have seen that by using them it is possible to refine the control of a TTF system with two tubes to a such extent that we do not need to have rules in the second tube any more.

The control of test tube systems with alternating filters is very powerful and it permits to apply easily the method of directing molecules. In the next chapter we show another example of utilisation of this method for systems which have a much more delicate control.

Chapter 8

Modified test tube systems

In this chapter we continue the study of test tube systems and we introduce another variant of them. As before we make changes at the level of the protocol of communication, but this time we do not add any additional ingredient. The modification which we make concerns molecules which may go out from a tube. In this case we do not permit any more to these molecules to remain in the initial tube, even if they can pass its filter. By consequent, we can control to a certain extent the elimination of these molecules. The obtained control is much more delicate than the control by alternating filters and it requires a careful manipulation. We show that two tubes suffice in order to produce all recursively enumerable languages. The method of directing molecules play an important role in the proof of this result which is very difficult to obtain in another way. Another interesting particularity of the obtained system is that the process of elimination is not static as in the case of test tube systems with alternating filters where we send incorrect molecules to another tube where they cannot evolve any more. In our case this process is dynamic and the eliminated molecule travels from one tube to another without having the possibility to participate in a splicing because during the corresponding step it cannot enter any active rule.

8.1 Modified test tube systems

Definition 8.1.1. A *modified test tube system* with n tubes is the following construction:

$$\Gamma = (V, T, (A_1, R_1, F_1), \dots, (A_n, R_n, F_n)),$$

where V, T, A_i, R_i et F_i are defined like in the case of a test tube system.

Modified test tube systems and the systems described in the previous chapter have a different communication protocol. More exactly, we define the transition between two configurations in the following way:

$(L_1, \dots, L_n) \Rightarrow (L'_1, \dots, L'_n)$ iff

$$L'_i = \left(\bigcup_{\substack{j=1 \\ j\neq i}}^{n} \sigma_j^*(L_j) \cap F_i^* \right) \cup \left(\sigma_i^*(L_i) \cap \left(V^* - \bigcup_{k=1}^{n} F_k^* \right) \right).$$

We see that in the union in the first parenthesis the case $j = i$ is excluded.

In words this means that if a molecule can pass a filter of a tube different from the tube where it was produced, then this molecule do not remain in the initial tube, even if it can pass its filter. This modification permits to organise an elimination of molecules and, consequently, to apply the method of directing molecules.

This definition is applicable in the case were we have at least two tubes. When $n = 1$, we consider, by definition, that $L'_1 = \sigma_1^*(A_1)$.

We denote by TTM_n the family of languages generated by modified test tubes systems having at most n tubes.

We remark the following property of the systems introduced above. Let Γ be a modified test tube system having 2 tubes. Let M be a molecule which can path filters of both tubes. Suppose also that initially it is situated in tube 1. It is easy to see that after the first step of computation M is sent to the second tube and no copy of M remains in the first tube. At the next step M returns to the first tube because it can pass its filter and no copy of M remains in the second tube. Therefore, at each step M changes the component, *i.e.* it appears in each tube with a period equal to two. So, in this way it is possible to create directing molecules with a period equal to 2 and whose apparition is directed by the control, see also Section 6.4.

Theorem 8.1.1. *Let $G = (N, T, P, S)$ be a type-0 grammar. Then, there is a modified test tube system having two tubes, $\Gamma = (V, T, (A_1, R_1, F_1), (A_2, R_2, F_2))$ which simulates G and $L(\Gamma) = L(G)$.*

Proof. We construct Γ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(B = a_n)$ and $B \notin N \cup T$.

In what follows we assume that:
$\mathbf{a} \in V \cup T \cup \{B\}$, $\mathbf{b} \in V \cup T \cup \{B\} \cup \{\alpha\}$, $\mathbf{c} \in V_\delta$, $\gamma \in \{\alpha, \beta\}$.

Consider $V_\delta = V \cup T \cup \{B\} \cup \{\alpha, \beta\}$.

Consider also $V_{AB} = \{A_x, B_x\}, x \in \{X'_\beta\beta, Y'_\beta, X'_\alpha, Y'_\alpha, X, Y, X_\alpha\alpha, Y_\alpha, X', Y'\}$ and $V_{CD} = \{D_x, C_y\}$, where $x \in \{X, X'_\beta\beta, X'_\alpha, X_\alpha\alpha, X'\}$ and $y \in \{Y'_\alpha, Y, Y'_\beta, Y_\alpha, Y'\}$.

The alphabet V is defined by:
$V = V_\delta \cup V_{AB} \cup V_{CD} \cup \{Z_X, Z_{BY}, Z_v, Z_{Y_i}, Z_{X_i}\} \cup$
$\{X, Y, X', Y', X_\alpha, Y_\alpha, X_\beta, Y_\beta, X'_\alpha, Y'_\alpha, X'_\beta, Y'_\beta\}$.

The tubes of the system are defined as follows.

Tube I:

Axioms of A_1:
$\{XBSY\} \cup \{Z_v vY : u \to v \in P\} \cup \{C_{Y'_\alpha}Y'_\alpha, XD_X, C_Y Y, X'_\beta\beta D_{X'_\beta\beta}, C_{Y'_\beta}Y'_\beta, X'_\alpha D_{X'_\alpha}\} \cup$
$\{Z_{Y_i}\beta\alpha^i Y, Xa_i Z_{X_i}, Z_{Y_\beta}Y_\beta\} \cup \{A_{Y'_\alpha}B_{Y'_\alpha}, B_X A_X\} \cup \{B_{X_\alpha}A_{X_\alpha}, A_{Y_\alpha}B_{Y_\alpha}, A_{Y'}B_{Y'}\}$.

Rules of R_1:

$$1.3.1: \begin{array}{c|c} \varepsilon & uY \\ \hline Z_v & vY \end{array} \quad 1.3.2: \begin{array}{c|c} \mathbf{a} & a_iY \\ \hline Z_{Y_i} & \beta\alpha^iY \end{array} \quad 1.3.3: \begin{array}{c|c} X\beta\alpha^i & \mathbf{a} \\ \hline Xa_i & Z_{X_i} \end{array} \quad 1.3.4: \begin{array}{c|c} \mathbf{a} & \beta Y \\ \hline Z_{Y_\beta} & Y_\beta \end{array}$$

$$1.1.1: \begin{array}{c|c} \gamma & Y_\alpha \\ \hline A_{Y'_\alpha} & Y'_\alpha \end{array} \quad 1.1.2: \begin{array}{c|c} X' & \gamma \\ \hline X & A_X \end{array}$$

$$1.2.1: \begin{array}{c|c} X & \alpha \\ \hline X'_\beta\beta & A_{X'_\beta\beta} \end{array} \quad 1.2.2: \begin{array}{c|c} \mathbf{a} & Y_\beta \\ \hline A_{Y'_\beta} & Y'_\beta \end{array} \quad 1.2.3: \begin{array}{c|c} X_\alpha & \mathbf{b} \\ \hline X'_\alpha & A_{X'_\alpha} \end{array} \quad 1.2.4: \begin{array}{c|c} \mathbf{c} & Y' \\ \hline A_Y & Y \end{array}$$

$$BA.x: \begin{array}{c|c} x & D_x \\ \hline B_x & A_x \end{array}, \quad \forall xD_x \in A_1, \qquad AB.y: \begin{array}{c|c} C_y & y \\ \hline A_y & B_y \end{array}, \quad \forall C_yy \in A_1$$

Filter F_1:
$V_\delta \cup V_{AB} \cup \{X_\alpha, Y_\alpha, X', Y'\}$.

Tube I:

Axioms of A_2:
$\{X_\alpha\alpha D_{X_\alpha\alpha}, C_{Y_\alpha}Y_\alpha, C_{Y'}Y', X'D_{X'}\} \cup \{Z_X, Z_{BY}\} \cup \{B_{X'}A_{X'}\} \cup$
$\{B_{X'_\beta}A_{X'_\beta}, A_{Y'_\beta}B_{Y'_\beta}, B_{X'_\alpha}A_{X'_\alpha}, A_YB_Y\}$.

Rules of R_2:

$$2.3.1: \begin{array}{c|c} X & \mathbf{a} \\ \hline \varepsilon & Z_X \end{array} \quad 2.3.2: \begin{array}{c|c} \varepsilon & BY \\ \hline Z_{BY} & \varepsilon \end{array}$$

$$2.1.1: \begin{array}{c|c} X & \mathbf{b} \\ \hline X_\alpha\alpha & A_{X_\alpha\alpha} \end{array} \quad 2.1.2: \begin{array}{c|c} \gamma & \alpha Y \\ \hline A_{Y_\alpha} & Y_\alpha \end{array} \quad 2.1.3: \begin{array}{c|c} \gamma & Y'_\alpha \\ \hline A_{Y'} & Y' \end{array} \quad 2.1.4: \begin{array}{c|c} \mathbf{a} & Y'_\beta \\ \hline A_{Y'} & Y' \end{array}$$

$$2.2.1: \begin{array}{c|c} X'_\alpha & \mathbf{b} \\ \hline X' & A_{X'} \end{array} \quad 2.2.2: \begin{array}{c|c} X'_\beta & \beta \\ \hline X' & A_{X'} \end{array}$$

$$BA.x: \begin{array}{c|c} x & D_x \\ \hline B_x & A_x \end{array}, \quad \forall xD_x \in A_2, \qquad AB.y: \begin{array}{c|c} C_y & y \\ \hline A_y & B_y \end{array}, \quad \forall C_yy \in A_2$$

Filter F_2:
$V_\delta \cup V_{AB} \cup \{X, Y, X'_\beta, Y'_\beta, X'_\alpha, Y'_\alpha, X_\alpha, Y'\}$.

We claim that $L(\Gamma) = L(G)$.

We shall prove this assertion in the following way. First we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(\Gamma)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. By consequent our assertion will be proved.

Our simulation is based on the "rotate-and-simulate" method. We use the same ideas as the ones used during the proof of Theorem 7.2.1. More precisely, we rotate the word Xwa_iY in three stages. First, we encode a_i by $\beta\alpha^i$ and we obtain $Xw\beta\alpha^iY$. After that we transfer α from the right end of the word to its left end and we obtain $X\beta\alpha^iwY$. Finally we decode $\beta\alpha^i$ into a_i and we obtain Xa_iwY.

In order to implement the above procedure we use the method of directing molecules, see Chapter 6. We consider two types of directing molecules. Directing molecules of the first type that direct the application of rules are of the form

xD_x or $C_y y$. They are created by nesting by the mean of directing molecules of the second type. Directing molecules of the second type are of form $A_y B_y$ or $B_x A_x$ and they are created by the control in the following way. Filters of both tubes permit the passage of these molecules, so they will appear in each tube with a period equal to two, see also the considerations presented before. Therefore, we obtain that both types of directing molecules appear with a period equal to two.

By using these directing molecules we separate the rules of the system in three groups.

1. Rules of the first tube which are applicable only during an odd step and rules of the second tube which are applicable only during an even step.
2. Rules of the first tube which are applicable only during an even step and rules of the second tube which are applicable only during an odd step.
3. Rules of both tubes which are applicable at any moment.

We say that a rule is activated during a step when it is applicable and deactivated during other steps.

The numbering of rules permits to know in which group the corresponding rule is placed. More exactly, the first number of a rule indicates to which tube that rule belongs to, while the second number indicates the group to which that rule belongs to.

Indeed, the separation of rules in these groups permits to obtain the following property: if a rule is applied to a molecule and if the result of this application is sent immediately to the other tube where it is processed by some rule and after that it is sent again in the first tube, then both rules which were used will be from the same group.

Now we show in details how we activate rules of groups (1) and (2). We take as example the rule 1.2.1: $\dfrac{X \mid \alpha}{X'_\beta\beta \mid A_{X'_\beta\beta}}$.

In Γ there are the following objets which are associated to this rule:

In the tube I:

The axiom: $X'_\beta\beta D_{X'_\beta\beta}$.

The rules:

$1.2.1: \dfrac{X \mid \alpha}{X_\beta\beta \mid A_{X'_\beta\beta}}$ and $BA.X'_\beta: \dfrac{X_\beta\beta \mid D_{X'_\beta\beta}}{B_{X'_\beta\beta} \mid A_{X'_\beta\beta}}$.

A part of the filter F_1: $B_{X'_\beta\beta}, A_{X'_\beta\beta}$.

In the tube II:

The axiom: $B_{X'_\beta\beta}A_{X'_\beta\beta}$.

None rule.

A part of the filter F_2: $X'_\beta, B_{X'_\beta\beta}, A_{X'_\beta\beta}$.

The directing molecule $B_{X'_\beta\beta}A_{X'_\beta\beta}$ travels from one tube to another and it appears in the first tube only during an even step. By consequent, the directing molecules of the first level $X'_\beta\beta A_{X'_\beta\beta}$ appears in the first tube only during an even step and after its usage it is immediately sent to the second tube. This means that the rule 1.2.1 is utilisable only during an odd step, because only the directing molecule $X'_\beta \beta A_{X'_\beta \beta}$ matches its lower site. Therefore, we obtain that the rule 1.2.1 belongs to the group (2).

All other rules from groups (1) and (2) are defined in a similar way.

We note that rules of the third group are utilisable all the time, because their lower site correspond to a molecule which is already present in the corresponding tube and which cannot pass the filter of the other tube because it contains the symbol Z with indices.

Notations

We note that the bottom site of each rule correspond either to a directing molecule, or to a molecule which is already present in the corresponding tube. Similarly, one of results of the splicing either contain the symbol Z with indices, so it cannot match any rule, either it is sent to the other tube and it remains there forever, or it travels at each step between two tubes and it does not alter the computation. This is why we omit these molecules and we write:

$Xwa_iY \underset{1.3.2}{\longrightarrow} Xw\beta\alpha^iY$ instead of

$(Xw|a_iY, Z_{Y_i}|\beta\alpha^iY) \vdash_{1.3.2} (Xw\beta\alpha^iY, Z_{Y_i}a_iY)$

where by | we highlighted the splicing sites. In what follows we mark molecules which can evolve in the same tube with ⓗ and we mark molecules which must be send into the first, respectively second, tube with ①, respectively ②. We also write $m \uparrow$ if the molecule m cannot enter any rule of the tube in which it is situated and if in the same time it cannot be sent to another tube.

We shall show the evolution of words of form Xwa_iY. We indicate the splicing rules which were used as well as the resulting words. We note that, due to the parallelism, in the same time in our system there can be several molecules of this form or intermediate forms which evolve in parallel. These molecules do not interact with each other, so we shall concentrate only on a single evolution of this type.

Rotation

We show how to rotate the word Xwa_iY which is in the first tube.

Step 1.

Tube 1.

Xwa_iY ② $\underset{1.3.2}{\longrightarrow} Xw\beta\alpha^iY$ ②.

Tube 2.

No applications of rules.

Step 2.

Tube 1.

No applications of rules.

Tube 2.

Xwa_iY ⓗ $\underset{2.3.1}{\longrightarrow} wa_iY \uparrow$.

$Xwa_iY \textcircled{h} \xrightarrow[2.1.1]{} X_\alpha \alpha wa_iY \uparrow$.
$Xw\beta\alpha^iY \textcircled{h} \xrightarrow[2.3.1]{} w\beta\alpha^iY \textcircled{h} \xrightarrow[2.1.2]{} w\beta\alpha^{i-1}Y_\alpha \textcircled{1}$.
$Xw\beta\alpha^iY \textcircled{h} \xrightarrow[2.1.1]{} X_\alpha \alpha w\beta\alpha^iY \textcircled{h} \xrightarrow[2.1.2]{} X_\alpha \alpha B\beta\alpha^{i-1}Y_\alpha \textcircled{1}$.
The last two rules can be applied in reverse order, but this gives the same result.
Step 3.
Tube 1.
$w\beta\alpha^{i-1}Y_\alpha \textcircled{h} \xrightarrow[1.1.1]{} w\beta\alpha^{i-1}Y'_\alpha \textcircled{2}$.
$X_\alpha \alpha w\beta\alpha^{i-1}Y_\alpha \textcircled{h} \xrightarrow[1.1.1]{} X_\alpha \alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{h}$.
Tube 2.
No applications of rules.
Step 4.
Tube 1.
$X_\alpha \alpha w\beta\alpha^{i-1}Y_\alpha \textcircled{h} \xrightarrow[1.2.3]{} X'_\alpha \alpha w\beta\alpha^{i-1}Y_\alpha \textcircled{h}$
$X_\alpha \alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{h} \xrightarrow[1.2.3]{} X'_\alpha \alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{2}$
Tube 2.
$w\beta\alpha^{i-1}Y'_\alpha \textcircled{h} \xrightarrow[2.1.3]{} w\beta\alpha^{i-1}Y' \textcircled{1}$
Step 5.
Tube 1.
$w\beta\alpha^{i-1}Y' \textcircled{2}$.
$X'_\alpha \alpha w\beta\alpha^{i-1}Y_\alpha \textcircled{h} \xrightarrow[1.1.1]{} X'_\alpha \alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{2}$.
Tube 2.
$X'_\alpha \alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{h} \xrightarrow[2.2.1]{} X'\alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{h}$.

The word $w\beta\alpha^{i-1}Y'$ is not changed because we are during an odd step and the rule 1.2.4 is not activated. Therefore, it will travel from one component to another, by consequent, it will not participate to the production of the result.

Step 6.
Tube 1.
No applications of rules.
Tube 2.
$X'_\alpha \alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{h} \xrightarrow[2.1.3]{} X'_\alpha \alpha w\beta\alpha^{i-1}Y' \textcircled{h}$.
$X'\alpha w\beta\alpha^{i-1}Y'_\alpha \textcircled{h} \xrightarrow[2.1.3]{} X'\alpha w\beta\alpha^{i-1}Y' \textcircled{1}$.
Step 7.
Tube 1.
$X'\alpha w\beta\alpha^{i-1}Y' \textcircled{h} \xrightarrow[1.1.2]{} X\alpha w\beta\alpha^{i-1}Y' \textcircled{2}$.
Tube 2.
$X'_\alpha \alpha w\beta\alpha^{i-1}Y' \textcircled{h} \xrightarrow[2.2.1]{} X'\alpha w\beta\alpha^{i-1}Y' \textcircled{1}$.
Step 8.
Tube 1.
$X'\alpha w\beta\alpha^{i-1}Y' \textcircled{h} \xrightarrow[1.2.4]{} X'\alpha w\beta\alpha^{i-1}Y \textcircled{h}$.
Tube 2.

$X\alpha w\beta\alpha^{i-1}Y'\,\textcircled{h} \xrightarrow[2.1.1]{} X_\alpha\alpha\alpha w\beta\alpha^{i-1}Y'\,\textcircled{1}$.

If $i = 1$, then we can also apply the rule 1.3.4, but this application will be discussed later at the step $8i$. Similarly, we will discuss later the evolution of words of form $X_\alpha wY'$.

Step 9.

Tube 1.

$X_\alpha\alpha\alpha w\beta\alpha^{i-1}Y'\,\textcircled{2}$.

$X'\alpha w\beta\alpha^{i-1}Y\,\textcircled{h} \xrightarrow[1.1.2]{} X\alpha w\beta\alpha^{i-1}Y\,\textcircled{2}$.

Tube 2.

No applications of rules.

So, we rotated the word $Xw\beta\alpha^{i-1}\alpha Y$.

We continue in this way until we obtain the word $X\alpha^i w\beta Y$ at the step $8i+1$. In this case we have the following computation which we shall detail starting from the previous step, $8i$, in order to show the application of the rule 1.3.4.

Step 8i.

Tube 1.

$X'\alpha^i w\beta Y\,\textcircled{h} \xrightarrow[1.3.4]{} X'\alpha^i wY_\beta\,\textcircled{h}$.

Tube 2.

No applications of rules.

Step 8i+1.

Tube 1.

$X'\alpha^i wY_\beta\,\textcircled{h} \xrightarrow[1.1.2]{} X\alpha^i wY_\beta\,\textcircled{2}$.

$X'\alpha^i w\beta Y \xrightarrow[1.1.2]{} X\alpha^i w\beta Y \xrightarrow[1.3.4]{} X\alpha^i wY_\beta \uparrow$.

Tube 2.

No applications of rules.

Step 8i+2.

Tube 1.

$X\alpha^i wY_\beta\,\textcircled{h} \xrightarrow[1.2.2]{} X\alpha^i wY'_\beta\,\textcircled{2}$.

$X\alpha^i wY'_\beta\,\textcircled{2} \xrightarrow[1.2.1]{} X'_\beta\beta\alpha^i wY'_\beta\,\textcircled{2}$.

Both previous applications may be done in reverse order, but this leads to the same result.

Tube 2.

No applications of rules.

Step 8i+3.

Tube 1.

No applications of rules.

Tube 2.

$X'_\beta\beta\alpha^i wY'_\beta\,\textcircled{h} \xrightarrow[2.2.2]{} X'\beta\alpha^i wY'_\beta\,\textcircled{h}$.

Step 8i+4.

Tube 1.

No applications of rules.

Tube 2.

$X\beta\alpha^i wY'_\beta \text{ⓗ} \underset{2.1.1}{\longrightarrow} X_\alpha\beta\alpha^i wY'_\beta \text{ⓗ} \underset{2.1.4}{\longrightarrow} X_\alpha\alpha\beta\alpha^i wY' \text{①}$.
$X'_\beta\beta\alpha^i wY'_\beta \text{ⓗ} \underset{2.1.4}{\longrightarrow} X'_\beta\beta\alpha^i wY' \text{ⓗ}$.
$X'\beta\alpha^i wY'_\beta \text{ⓗ} \underset{2.1.4}{\longrightarrow} X'\beta\alpha^i wY' \text{①}$.

After this moment the evolution is similar to the evolution after the step 6 and after several steps we obtain $X\beta\alpha^i wY$ in the first tube. After that we can apply the rule 1.3.3: $X\beta\alpha^i wY \underset{1.3.3}{\longrightarrow} Xa_i wY$, by consequent, we rotated Xwa_iY.

We note that in spite of the soundness of the computation above, we cannot be sure that the system function correctly. Indeed, the computation of the system depends on the parity of the step number and if a molecule would appear during a step with another parity, then it would cause an erroneous computation. This is why we show below that we cannot obtain incorrect molecules during a step which is different from the one showed above.

In order to do this we analyse molecules bracketed by X and Y with indices which can pass from the second tube to the first one. This gives us 4 possibilities:
a) Molecules of type $X_\alpha wY'$.
b) Molecules of type $X'wY_\alpha$.
c) Molecules of type $X_\alpha wY_\alpha$.
d) Molecules of type $X'wY'$.

The first possibility is realisable only during an even step because we should have been used one of rules 2.1.1, 2.1.3 or 2.1.4 which are from the first group. In this case the molecule $X_\alpha wY'$ appears in the first tube during an odd step and it cannot enter any rule because all rules that it matches are not activated during this step. Consequently, this molecule is sent unchanged to the second tube. Similarly, in that tube there is no rule that can be matched by this molecule. Therefore, this molecule will change the tube at each step, as it was described above. In this way we implemented a garbage collector which is not fixed and whose contents moves to the other component after each step.

We shall show now that (b) is not possible. First we observe that in order to obtain $X'wY_\alpha$ we should have been used rules 2.2.2 and 2.1.2, by consequent, we should had $X'_\beta wY$ in the second tube. This is possible only if this word was sent before from the first tube. In order to have the word $X'_\beta wY$ in the first tube we should had the word XwY in the first tube during an even step. It is easy to see that this is not possible. If we have the word XwY during an odd step, then it is immediately sent to the second tube. Similarly, if we would had $X'wY'$ and if we would apply the rule 1.1.2, then the resulting word XwY' would be sent immediately to the second tube.

Now we analyse the remaining possibilities. First we note that in the second tube we can add at most one α to the beginning of the word and to erase at most one α at the end. Similarly, in order to have the case (c) or (d) we need to apply both rules once and this leaves unchanged the number of α in a word.

If we have the case (c), then only words $X'_\alpha wY'_\alpha$ and $X_\alpha wY'_\alpha$ are sent to the second tube. The first word represents a correct evolution and it is transformed in $X'wY'$ which is the case (d). The second word may become $X_\alpha wY'$ which represents the case (a) which is already examined.

If we have the case (d), then we obtain $X'wY'$ in the first tube.

Now if we are at an odd step, then we apply the rule 1.1.2 and eventually the rule 1.3.3 obtaining XwY' which is sent to the second tube where it can evolve to $X_\alpha \alpha wY'$ which is the case (a).

If we are at an even step, then we apply the rule 1.2.4 obtaining $X'wY$. After that during an odd step we apply the rule 1.1.2 obtaining XwY which represent a correct evolution. We can also apply rules 1.3.4, 1.2.2 or 1.2.1 obtaining $X'wY_\beta$, XwY_β, XwY'_β or $X'_\beta wY'_\beta$. The fourth word represents a correct evolution. The first two words may evolve to the third word only. The third word is sent to the second tube where it is transformed in $X_\alpha wY'$ which is the case (a).

Simulation of productions and the result

It is easy to see that we do a correct simulation of productions of the grammar by the rule 1.3.1 and that we obtain a correct word of the grammar by using rules 2.3.1 and 2.3.2.

□

8.2 Conclusions

In this chapter we terminate the study of the method of directing molecules. During last three chapters we have shown different examples which use this method that permits to simplify a lot of proofs. The key of this method, the regulated creation based on elimination, may be realised in different ways, even if we do not have physical elimination. The systems studied in this chapter have an interesting elimination which is dynamic. Also, the "trash" containing unwanted molecules travel from one component to another. This process needs a good synchronisation of different parts of the computation and it may be a good starting point for an improvement of the method of directing molecules.

So, we stop here and in the next chapter we consider other systems which have a different inspiration and structure, but which have strong links with systems that we already studied.

Chapter 9

Splicing P systems

In previous chapters we were talking about computational models inspired by the way that living beings manipulate their DNA. Consequently, our point of view was placed at the molecular level. In this chapter we change the view and we place ourself at the cellular level by considering membrane systems, or P systems, which are a model of computation inspired by different intra- and inter-cellular processes. The cell is considered as a set of compartments nested one in another and which contain objects and evolution rules. The base model does not specify neither the nature of these objects, nor the nature of rules. Numerous variants specify these two parameters by obtaining a lot of different models of computing. We are interested in models having strings as objects and splicing rules as rules. Such systems are called splicing membrane systems, or splicing P systems. It is known that two membranes suffice to generate all recursively enumerable languages. We are particulary interested in systems having only one membrane. We found that the original definition of such such systems given by Gh. Păun is not complete and we propose several variants in order to complete it. We show that the choice of the variant is important and that the computational power of obtained systems directly depends on it. We also show similarities between splicing P systems having one membrane and TVDH systems of degree 1. These similarities permit to simulate certain types of such TVDH systems by splicing P systems with one membrane.

We also study non-extended splicing P systems, *i.e.* which have no terminal alphabet. We give the frontier between the decidability and undecidability for these systems by showing that two membranes suffice in order to generate all recursively enumerable languages, while one membrane limits the power of the corresponding systems by the family of regular languages.

In the remaining of the chapter we deal with another variant of splicing P systems: splicing P systems with immediate communication. In such systems the result of the application of splicing rules is sent to all adjacent membranes. These systems have a strong connection with TVDH systems and we show that two membranes suffice in order to generate all recursively enumerable languages. In the same time we show that with one membrane we can generate only the family of finite languages, hence we showed again in this case the frontier between the decidability and the undecidability.

The method of directing molecules may also be used in the case of splicing P systems. Even if we do not give any proof by using this method, it is still possible to apply it and we shall indicate it at the appropriate moment.

9.1 Splicing membrane systems

In this section we give some notions about membrane systems and we introduce splicing P systems.

A *membrane structure* of a P system is the hierarchical arrangement of membranes which are nested into the *skin* membrane which separates the system from the *environment*. A membrane which does not contain any other membrane is called *elementary*. Each membrane defines a *region*. The region of an elementary membrane is the space enclosed by that membrane, while the region on a non-elementary membrane is the space between that membrane and the membranes included in it. The Fig. 9.1 illustrates these notions. We label membranes by positive integer numbers. We note that there is a one-to-one correspondence between membranes and the regions that they define, this is why we will not make in the future the difference between membranes, their regions and their numbers.

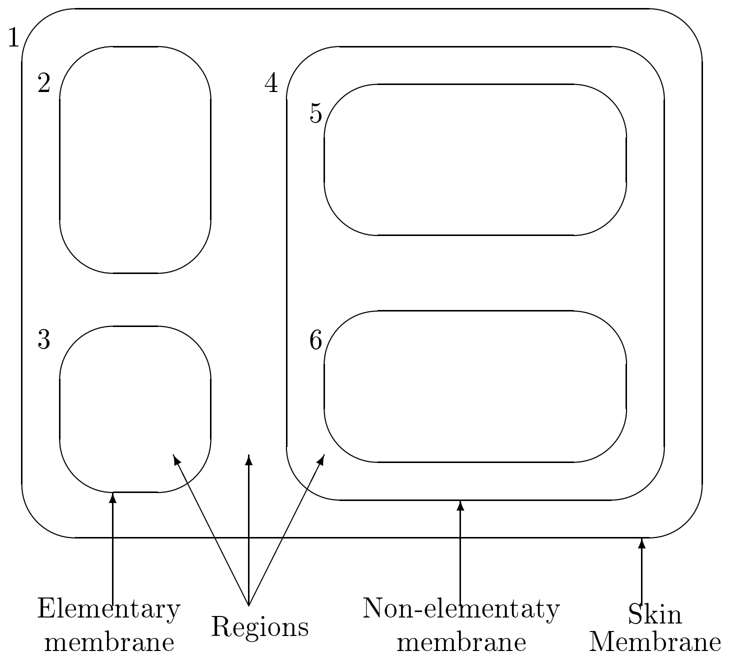


Figure 9.1: A membrane structure

The spacial arrangement of membranes may be represented by a Venn diagram, see Fig. 9.1, or by a tree whose nodes represent membranes and whose edges are defined by the relation "be contained in", see Fig. 9.2. The same structure can be defined by a string of matching parenthesis where the corresponding parenthesis have the same label. For example, the structure represented in Fig. 9.1 may be described by the following word: $[_1\, [_2\,]_2\, [_3\,]_3\, [_4\, [_5\,]_5\, [_6\,]_6\,]_4\,]_1$.

$$\begin{tikzcd}
& 1 \arrow[dl] \arrow[d] \arrow[dr] & & \\
2 & 3 & 4 \arrow[dl] \arrow[dr] & \\
& 5 & & 6
\end{tikzcd}$$

Figure 9.2: The tree representing the membrane structure from Fig. 9.1

Definition 9.1.1. A *splicing P system* of degree $m \geq 1$ is the following $2m+3$-tuple:

$$\Pi = (V, T, \mu, A_1, \dots, A_m, R_1, \dots, R_m),$$

where V is an alphabet, $T \subseteq V$ is the terminal alphabet, μ is a membranes structure containing m membranes, $A_i \subseteq V^*$ are languages associated to regions $1, \dots, m$ of μ, R_i are finite sets of evolution rules of form $(r; tar_1, tar_2)$, where r is a splicing rule: $r = u_1\#u_2\$u_3\#u_4$ and $tar_1, tar_2 \in \{here, in, out\}$ are target indicators.

We can also write $\dfrac{u_1 \mid u_2}{u_3 \mid u_4}\begin{matrix}\scriptstyle (tar_1)\\ \scriptstyle (tar_2)\end{matrix}$ instead of $(u_1\#u_2\$u_3\#u_4; tar_1, tar_2)$.

We say that the system Π has *global* rules if $R_1 = \dots = R_m$.

A *configuration* of Π is the m-tuple $(N_1, \dots, N_m)$, where $N_i \subseteq V^*$. We define the transition between two configurations $(N_1, \dots, N_m) \Rightarrow (N'_1, \dots, N'_m)$ as follows. In order to pass from one configuration to another we apply splicing rules of each region of μ, in parallel, to all possible words which are in corresponding regions and after that we distribute the result of each splicing depending on 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 the words w and z are sent to the regions indicated by tar_1, respectively tar_2. We write this as follows $(x, y) \vdash_r (w, z)(tar_1, tar_2)$. If $tar_j = here$, then the molecule remains in the membrane i; if $tar_j = out$, then the molecule is sent to a region situated immediately above the membrane i, in this case the molecule can also exit the system; if $tar_j = in$, the molecule is sent to a region situated immediately below the membrane i and that is chosen non-deterministically.

Since the words are present in an arbitrary number of copies, after the application of rule r in membrane i, the molecules x and y are still present in the same region. However, there are some particular cases which will be discussed later.

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

If we have only one membrane, then we omit the parentheses and we write $M_1 \Rightarrow M_2 \Rightarrow \dots \Rightarrow M_k$ $(M_1 = A_1)$ for a sequence of transitions.

We can encounter the following problem: how we should proceed if we have a word w in a membrane and we produce the same word w by a certain rule r and this word shall be sent outside the membrane. There are two possible solutions: the first one is to keep the old w in the membrane and to send a new copy of w outside. The second solution is to send w outside and to eliminate it from the membrane.

A similar problem arises in the case when there are two rules which produce the same word w one having the target indicator *here* and the other one having the target indicator *in* or *out*. Also, both situations described above may happen in the same time.

In order to solve the problems above we shall consider all possible cases and we obtain several solutions which we present below.

Let w be a molecule which is produced by a rule r in the membrane i and suppose that w is sent outside of the membrane. Then we have the following possibilities:

1. The molecule w is sent outside of the membrane and no copy of it remains in the same membrane.

2. The molecule w is sent outside of the membrane, but a copy of it remains in the same membrane providing that one of following conditions is satisfied.

   (a) The molecule w is already present in the same membrane.

   (b) There is another rule r' which produces w with the target indicator *here*.

   (c) Both conditions 2a and 2b are satisfied.

   (d) One of conditions 2a or 2b is satisfied.

So, in the variant 1 we send all occurrences of the word w outside of the membrane. In variants 2a and 2d, if w is produced in a membrane, then it remains there forever. The variants 2b and 2c are situated somewhere between two extremes above. The variants 2a and 2d were considered by Gh. Păun in [43], but without distinguishing them.

We say that a splicing P system is of *type x* if we consider the variant x in order to solve problems above.

We denote by $ELSP_m(spl, in)$ the family of languages generated by splicing P systems of type 2d having the degree at most m and by $ELSP_m(spl, in_w)$ the family of languages generated by splicing P systems of type 2a having the degree at most m. We also denote by $ELSP_m(spl, in \uparrow)$ the family of languages generated by splicing P systems of type 1 having the degree at most m.

The problems above may be avoided if we consider systems which have more than one membrane. More exactly, it is possible to construct a splicing P system having two membranes and which generates all recursively enumerable languages without encounter one of situations above, see [36, 9, 43]. On the other hand, we show that in the case of only one membrane the choice of the definition is very important and that obtained systems have a different computational power. So, in the remaining of this section we shall consider splicing P systems which have only one membrane.

9.1.1 Decidability results

We show below the limits of the family $ELSP_1(spl, in)$.

Let us consider splicing P systems of type 2d. Let $\Pi = (V, T, [_1]_1, M_1, R_1)$ be a such system. In this subsection we consider also 1-splicing instead of 2-splicing. This consideration does not restrict the generated language, because we can replace every rule of form $(u_1\#u_2\$u_3\#u_4; tar_1, tar_2)$ by two rules $(u_1\#u_2\$u_3\#u_4; tar_1)$ and $(u_3\#u_4\$u_1\#u_2; tar_2)$, see also Proposition 3.1.2 at the page 28.

In this case we can decompose R_1: $R_1 = R_1^h \cup R_1^o$, where R_1^h contains all rules having a target indicator *here* and R_1^o contains all rules having a target indicator *out*. We also denote by $\sigma_{h_o} = (V \cup T, R_1^o)$ and $\sigma_{h_h} = (V \cup T, R_1^h)$ the corresponding H schemes and by $H_h = (h_h, M_1)$ the corresponding H system.

Lemma 9.1.1. *$M_{k+1} = \sigma_{h_h}^k(M_1)$, where $M_1 \overset{k}{\Rightarrow} M_{k+1}$.*

Proof. We shall prove this statement by induction. The base of induction is true because both sets are equal to M_1. It is easy to see that the induction step is also true because in order to obtain M_{k+1} we add to M_k molecules obtained by the applying rules having a target indicator *here* on M_k, *i.e.* $M_{k+1} = M_k \cup \sigma_{h_h}(M_k)$. □

Lemma 9.1.2. *$L(\Pi) = \sigma_{h_o}(L(H_h)) \cap T^*$.*

Proof. Let $w \in L(\Pi)$. Then $w \in T^*$, $w = x_1u_1u_4y_2$ and there is a number $k \in \mathbb{N}$ and the words $x, y \in M_{k+1}$ such that $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$. There is also a rule $r = (u_1\#u_2\$u_3\#u_4; out) \in R_1^o$ such that the word w was produced by the following application: $(x, y) \vdash_r w$. From the previous lemma we obtain $x, y \in \sigma_{h_h}^k(M_1) \subseteq L(H_h)$. By consequent, $w \in \sigma_{h_o}(L(H_h)) \cap T^*$.

Conversely, if we have $w \in \sigma_{h_o}(L(H_h)) \cap T^*$, then there is a number $k \in \mathbb{N}$, the words $x, y \in \sigma_{h_h}^k(M_1)$ and a rule $r \in R_1^o$ such that $(x, y) \vdash_r w$. In this case $x, y \in M_{k+1}$, see Lemma 9.1.2, and w is sent outside of the system by r. Also $w \in T^*$. This implies $w \in L(\Pi)$. □

Since the family of regular languages is closed with respect to splicing and with respect to intersection, we obtain the following result.

Corollary 9.1.3. *$ELSP_1(spl, in) \subseteq REG$.*

The converse inclusion is also true.

Lemma 9.1.4. *$REG \subseteq ELSP_1(spl, in)$.*

Proof. It is easy to see that we can simulate an extended H system S by a splicing P system of type 2d, Π, which has only one membrane. This simulation can be done as follows: for each rule r of S we consider two rules $(r; here, here)$ and $(r; out, out)$ in Π. In this case the first rule permits to simulate the rule r of the system S, while the second rule permits to send outside of the membrane the result of the computation. $\square$

Therefore, we showed:

Theorem 9.1.5. *$ELSP_1(spl, in) = REG$.*

9.1.2 Undecidability results

In this subsection we consider the family $ELSP_m(spl, in \uparrow)$.

We shall prove the following theorem.

Theorem 9.1.6. *$ELSP_1(spl, in \uparrow) = RE$.*

We define:

$$Out(r) = \left\{ \frac{u_1 \mid u_2}{u_1 \mid u_2} \begin{matrix} (out) \\ (out) \end{matrix}, \frac{v_1 \mid v_2}{v_1 \mid v_2} \begin{matrix} (out) \\ (out) \end{matrix} : r = \frac{u_1 \mid u_2}{v_1 \mid v_2} \right\}.$$

Now we show how it is possible to simulate a TVDH system of degree 1 by a splicing P system of type 1 having one membrane. The main idea is to simulate the application of a rule r of a TVDH system of degree 1 by the following rules:

1. $(r; here, here)$.
2. $Out(r)$.

The rule (1) permits to simulate the application of r, while the rules (2) eliminate initial molecules.

In order to prove the theorem we prove first the following lemma.

Lemma 9.1.7. *Let $D = (V, T, A, R)$ be a TVDH system of degree 1 which has the following properties:*

- *$L_k \cap L_{k+1} = \emptyset$,* i.e. *molecules produced at the step k and $k+1$ are different.*
- *In order to obtain the resulting word we use once a rule from the set $R_R \subseteq R$ and the second result of this splicing may be eliminated without altering the generated language.*

Then, there is a splicing P system $\Pi \in ELSP_1(spl, in \uparrow)$ which simulates D and $L(\Pi) = L(D)$.

Proof.
We consider the following splicing P system $\Pi = (V, T, [_1]_1, M_1, R')$, where $M_1 = A$ et $R' = R_1 \cup R_2 \cup R_3$ with

$R_1 = \{(r, here, here), Out(r) : r \in R \setminus R_R\}$,

$R_2 = \{(r, out, out), Out(r) : r \in R_R\}$,

$R_3 = \left\{ \begin{array}{c|c} w & \varepsilon \\ \hline w & \varepsilon \end{array} \begin{array}{c} (out) \\ (out) \end{array} : w \in A \right\}$.

We note that, by construction of Π, if M_k contains a molecule which cannot match any rule of Π, then this molecule will remain forever in the membrane and it will not participate in any further computation, so it will never participate to the production of the result. We shall consider the equivalence class defined by the relation "match a rule", *i.e.* we do not consider any more molecules which cannot match any rule of the membrane. So, at each step all molecules of the system match a rule of Π and, because of the form of rules, participate in a splicing.

We claim that $L(\Pi) = L(D)$.

In order to prove this affirmation we shall prove the following assertions:

1. $L_k \setminus T^* = M_k$.

2. $L_{k+1} \cap T^* = \sigma_{h_o}(M_k) \cap T^*$, where $\sigma_{h_o} = (V, R_o)$ and R_o contains all rules of R' having a target indicator *out*.

Proof of 1. We shall show that $L_k \setminus T^* = M_k$, *i.e.* the set of non-terminal molecules of L_k coincide with M_k.

We shall prove this assertion by induction. The base of the induction is true as both sets are equal to A. Let us suppose now that $L_k \setminus T^* = M_k$ and let us prove that $L_{k+1} \setminus T^* = M_{k+1}$.

Let $w \in L_{k+1} \setminus T^*$. Then, there are words x and y in L_k and a rule r in $R \setminus R_R$ such that w is produced by the following application: $(x, y) \vdash_r (w, z)$. We remark that the second property of D implies that the rule r belongs to the set $R \setminus R_R$, because otherwise w would be composed only from the letters of the terminal alphabet T. So, the words w and z are not terminal and they are different of x and y because of the first property of D. By induction hypothesis x and y belong to M_k. As well, by construction, there is a rule $(r; here, here)$ in Π that we can apply to x and y producing w and z: $(x, y) \vdash_r (w, z)(here, here)$. We note that, because of the first property of D, the words w and z are different of any x and y which may enter a rule of D, *i.e.* we cannot produce molecules w and z with a target indicator *out*. Therefore, the only molecules which may be sent outside are the molecules of M_k, except the axioms which are sent outside by the rules of R_2. So, we obtain $w, z \in M_{k+1}$.

Conversely, let $w \in M_{k+1}$. Then there are words x and y in M_k complementary with respect to the rule $(r, here, here) \in R'$ such that w is produced by the following application: $(x, y) \vdash_r (w, z)(here, here)$. We remark that if we consider a rule having target indicators *out*, then the result of the application of that rule will not belong to M_{k+1}. Hence, both results w and z remain in the membrane. In the same time, the words x and y are sent outside by rules of $Out(r)$. Since x and y are in L_k by the induction hypothesis and there is, by construction, a rule $r \in R$, we have the following application in D: $(x, y) \vdash_r (w, z)$, *i.e.* $w, z \in L_{k+1}$.

So, we showed that $L_{k+1} \setminus T^* = M_{k+1}$.

Proof of 2. We shall show that $L_{k+1} \cap T^* = \sigma_{h_o}(M_k) \cap T^*$, *i.e.* that the terminal words of L_{k+1} are the terminal words sent outside of the membrane at the step $k+1$.

Let $w \in L_{k+1} \cap T^*$. Then, there are words x and y in L_k and a rule r in R_R such that w is obtained by the following splicing: $(x, y) \vdash_r (w, z)$. We remark that the second property of D implies that the rule r belongs to R_R, because otherwise w would not be composed from letters of the terminal alphabet T. The same property of D implies that $x, y \in L_k \setminus T^*$. By consequent, $x, y \in M_k$. Also, by construction, there is a rule $(r, out, out) \in R'$ and we can splice x and y in Π by r by sending the resulting words w and z outside of the membrane.

Conversely, if a terminal word $w \in T^*$ is sent outside of the membrane by the rule (r, out, out), where $r \in R_R$, then this w can be obtained by using the rule r in D. From the other hand, if a word w is sent outside of the membrane by a rule (r, out, out), where $r \notin R_R$, then this word will contain non-terminal letters because of the second property of D. $\square$

Now in order to prove Theorem 9.1.6 it suffices to observe that the TVDH system constructed in Theorem 6.3.1 satisfy the conditions of the previous lemma.

We could also use the method of directing molecules in order to prove Theorem 9.1.6. We can take the flow-chart of the computation shown in the Fig. 4.1, see page 46, and introduce necessary directing molecules. For example, following the notations of Theorem 6.3.1, see Fig. 6.2 at page 76, we consider the directing molecule Z_2Y_i which shall appear with period 8. For this it is enough to add the words Z_2Y_i, Z_2Z, Z_2^kZ, $1 \leq k \leq 7$ to the axioms, and to add the rules $\dfrac{Z_2^{k-1} \mid Y_i}{Z_2^k \mid Z}\begin{smallmatrix}(here)\\(here)\end{smallmatrix}$, $\dfrac{Z_2^{7} \mid Y_i}{Z_2 \mid Z}\begin{smallmatrix}(here)\\(here)\end{smallmatrix}$, $\dfrac{Z_2^{k}Y_i \mid \varepsilon}{Z_2^kY_i \mid \varepsilon}\begin{smallmatrix}(out)\\(out)\end{smallmatrix}$ et $\dfrac{Z_2Y_i \mid \varepsilon}{Z_2Y_i \mid \varepsilon}\begin{smallmatrix}(out)\\(out)\end{smallmatrix}$, $1 \leq k \leq 7$ to the rules of the membrane.

9.1.3 Systems of type 2b, 2c and 2a

In this subsection we show that splicing P systems of types 2b and 2c generate all recursively enumerable languages.

Theorem 9.1.8. *Let $D = (V, T, A, R)$ be a TVDH system of degree 1. Then, there is a splicing P system of type 2b and of degree 1, Π, which simulates D and $L(\Pi) = L(D)$.*

Proof. Consider $\Pi = (V, T, [_1]_1, A, R')$, where

$$R' = \{(r, here, here); (r, out, out); Out(r) : r \in R\} \cup \left\{ \frac{w \mid \varepsilon}{w \mid \varepsilon}\begin{smallmatrix}(out)\\(out)\end{smallmatrix} : w \in A \right\}.$$

It is easy to see that Π simulates D. Let $x, y \in L_k$ which are complementary with respect to the rule $r \in R$. Then we have the following application in D: $(x, y) \vdash_r (w, z)$. In Π, we have $(x, y) \vdash_r (w, z)$ and the words w and z are sent outside of the membrane, but their copies are kept inside as well. The words x and y are sent outside by the rule $Out(r)$ and they do not remain any more in the membrane. Therefore, $L(D) \subseteq L(\Pi)$.

It is clear that the converse inclusion is also true. If we use a sequence of rules $(r_1, here, here), \ldots, (r_n, out, out)$ in order to obtain a molecule w in Π, then the similar sequence $r_1, \ldots, r_n$ permits to obtain the same word w in D.

□

As $VDH_1 = RE$ we obtain:

Corollary 9.1.9. *Splicing P systems of type 2b and of degree 1 generate all recursively enumerable languages.*

For systems of type 2c we have a similar result.

Theorem 9.1.10. *Splicing P systems of type 2c and of degree 1 generate all recursively enumerable languages.*

More exactly, this theorem is a corollary of Theorem 9.1.6. If we take the proof of that theorem we observe that the situation when a copy of w remains in the membrane never happens.

On the other hand, we could not describe the computational power of splicing P systems of type 2a having only one membrane, but we suppose that they do not have a big computational power.

Conjecture 9.1.1. $ELSP_1(spl, in_w) = REG$.

We see that this case is very similar to the one discussed in subsection 9.1.1. Indeed, it is easy to see that

$M_{k+1} = M_k \cup \{\sigma_{h_h}(M_k) \setminus \sigma_{h_o}(M_k)\}$,

following notations of that section.

Similarly, it is easy to see that if a molecule w is produced in the membrane before being sent outside, then it will remain there forever. Otherwise, it will never be present in the membrane. This point is the main argument of our conjecture.

9.2 Non-extended splicing P systems

In this section we consider non-extended splicing P systems which differ from the systems introduced before by the fact that they do not have a terminal alphabet. Therefore, all words sent outside of the system form the result of the computation.

We denote by $LSP_m(spl, in)$ the family of languages generated by non-extended splicing P systems of degree at most m. We do not indicate the type of the corresponding system because, as it it shown in the next theorem, one membrane does not suffice in order to obtain a big computational power and with two membranes we can avoid the problems which forced us to introduce different definitions.

Theorem 9.2.1. *$LSP_1(spl, in) \subset REG$.*

This theorem is a corollary of Lemma 9.1.2 and of Theorem 3.1.3, see also Tab. 2.1.

Theorem 9.2.2. *Let $G = (N, T, S, P)$ be a type-0 grammar. Then, there is a non-extended splicing P system of degree 2, Π, having global rules which simulates G and $L(\Pi) = L(G)$.*

Proof. We construct $\Pi = (V, [_1[_2\ ]_2]_1, M_1, M_2, R_1 \cup R_2)$ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(a_n = B)$ and $B \notin \{N \cup T\}$. We assume that $T = \{a_s, \ldots, a_{n-1}\}$.

In what follows we assume that:
$1 \leq \mathbf{i} \leq n$, $0 \leq \mathbf{j} \leq n-1$, $\mathbf{a} \in N \cup T \cup \{B\}$, $s \leq \mathbf{p} \leq n-1$, $s-1 \leq \mathbf{q} \leq n-2$.

The alphabet V is defined by:
$V = N \cup T \cup \{B\} \cup \{X, Y, X_\mathbf{i}, Y_\mathbf{i}, X'_\mathbf{j}, Y'_\mathbf{j}, X_0, Y_0, X', Y', Z_1, Z_2, V, W, V_\mathbf{i}, W_\mathbf{i}, V'_\mathbf{j}, W'_\mathbf{j}, V_{s-1}, W_{s-1}, V', W', V_E, W_E, V'_E, W'_E, V''_E, E, Z_V, Z_W\}$.

The axioms are defined by:
$M_1 = \{XSBY\} \cup \{Z_1 v Y_\mathbf{j} : \exists u \to v a_\mathbf{j} \in P\} \cup \{Z_1 Y_\mathbf{i}, Z_1 Y_0, Z_1 Y'_\mathbf{j}, X' Z_1, X Z_1, X_V B Z_1, Z_1 Y'_W, X'_V Z_1, Z_1 W_p, Z_1 W'_q, Z_1 W_{s-1}, V' Z_1, V Z_1, V_E Z_1, Z_1 W'_E, V'_E Z_1, V''_E Z_1, Z_W, Z_V\}$.

$M_2 = \{X_\mathbf{i} a_\mathbf{i} Z_2, X'_\mathbf{j} Z_2, X_\mathbf{j} Z_2, X_0 Z_2, Z_2 Y', Z_2 Y, Z_2 Y_W, Z_2 W, V Z_2, V_\mathbf{p} a_\mathbf{p} Z_2, V'_\mathbf{q} Z_2, V_\mathbf{q} Z_2, V_{s-1} Z_2, Z_2 W', Z_2 W, Z_2 W_E, Z_2 W''_E, E Z_2\}$.

The rules of the system are defined as follows.

Rules of R_1:

$$1.1 : \begin{array}{c|c} \varepsilon & uY \\ \hline Z & vY \end{array} \begin{array}{l} (in) \\ (in) \end{array}, \quad \text{si } \exists u \to v \in P;$$

$$1.2 : \begin{array}{c|c} \varepsilon & a_\mathbf{i} Y \\ \hline Z & Y_\mathbf{i} \end{array} \begin{array}{l} (in) \\ (here) \end{array}; \quad 1.3 : \begin{array}{c|c} \mathbf{a} & Y_\mathbf{i} \\ \hline Z_1 & Y'_{\mathbf{i}-1} \end{array} \begin{array}{l} (in) \\ (here) \end{array}; \quad 1.4 : \begin{array}{c|c} \mathbf{a} & Y'_\mathbf{j} \\ \hline Z_1 & Y_\mathbf{j} \end{array} \begin{array}{l} (in) \\ (here) \end{array};$$

$$1.5 : \begin{array}{c|c} X_0 & \mathbf{a} \\ \hline X' & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array}; \quad 1.6 : \begin{array}{c|c} X' & \mathbf{a} \\ \hline X & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array};$$

$$1.7 : \begin{array}{c|c} X' & \mathbf{a} \\ \hline X_V B & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array}; \quad 1.8 : \begin{array}{c|c} \mathbf{a} & Y_W \\ \hline Z_1 & Y'_W \end{array} \begin{array}{l} (here) \\ (in) \end{array}; \quad 1.9 : \begin{array}{c|c} X_V & B \\ \hline X'_V & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array};$$

$$1.10 : \begin{array}{c|c} X'_V & B \\ \hline X''_V & Z_1 \end{array} \begin{array}{l} (in) \\ (here) \end{array}; \quad 1.11 : \begin{array}{c|c} \mathbf{a} & Y''_W \\ \hline Z_1 & W \end{array} \begin{array}{l} (in) \\ (here) \end{array};$$

$$1.12: \begin{array}{c|c} \varepsilon & a_{\mathbf{p}}W \\ \hline Z_1 & W_{\mathbf{p}} \end{array} \begin{array}{l} (in) \\ (here) \end{array}; \quad 1.13: \begin{array}{c|c} \mathbf{a} & W_{\mathbf{p}} \\ \hline Z_1 & W'_{\mathbf{p}-1} \end{array} \begin{array}{l} (in) \\ (here) \end{array}; \quad 1.14: \begin{array}{c|c} \mathbf{a} & W'_{\mathbf{q}} \\ \hline Z_1 & W_{\mathbf{q}} \end{array} \begin{array}{l} (in) \\ (here) \end{array};$$

$$1.15: \begin{array}{c|c} V_{s-1} & \mathbf{a} \\ \hline V' & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array}; \quad 1.16: \begin{array}{c|c} V' & \mathbf{a} \\ \hline V & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array};$$

$$1.17: \begin{array}{c|c} V' & \mathbf{a} \\ \hline V_E & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array}; \quad 1.18: \begin{array}{c|c} \mathbf{a} & W_E \\ \hline Z_1 & W'_E \end{array} \begin{array}{l} (here) \\ (in) \end{array}; \quad 1.19: \begin{array}{c|c} V_E & \mathbf{a} \\ \hline V'_E & Z_1 \end{array} \begin{array}{l} (here) \\ (in) \end{array};$$

$$1.20: \begin{array}{c|c} V'_E & \mathbf{a} \\ \hline V''_E & Z_1 \end{array} \begin{array}{l} (in) \\ (here) \end{array}; \quad 1.21: \begin{array}{c|c} \mathbf{a} & W''_E \\ \hline Z_W & \varepsilon \end{array} \begin{array}{l} (in) \\ (in) \end{array};$$

$$1.22: \begin{array}{c|c} E & \mathbf{a} \\ \hline \varepsilon & Z_V \end{array} \begin{array}{l} (in) \\ (out) \end{array};$$

Rules of R_2:

$$2.1: \begin{array}{c|c} X & \mathbf{a} \\ \hline X_{\mathbf{i}}a_{\mathbf{i}} & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array}; \quad 2.2: \begin{array}{c|c} X_{\mathbf{i}} & \mathbf{ab} \\ \hline X'_{\mathbf{i}-1} & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array}; \quad 2.3: \begin{array}{c|c} X'_{\mathbf{j}} & \mathbf{a} \\ \hline X_{\mathbf{j}} & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array};$$

$$2.4: \begin{array}{c|c} \mathbf{a} & Y_0 \\ \hline Z_2 & Y' \end{array} \begin{array}{l} (out) \\ (here) \end{array}; \quad 2.5: \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z_2 & Y \end{array} \begin{array}{l} (out) \\ (here) \end{array};$$

$$2.6: \begin{array}{c|c} \mathbf{a} & BY' \\ \hline Z_2 & Y_W \end{array} \begin{array}{l} (out) \\ (here) \end{array}; \quad 2.7: \begin{array}{c|c} \mathbf{a} & Y'_W \\ \hline Z_2 & Y''_W \end{array} \begin{array}{l} (out) \\ (here) \end{array}; \quad 2.8: \begin{array}{c|c} X''_V & B \\ \hline V & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array};$$

$$2.9: \begin{array}{c|c} V & \mathbf{a} \\ \hline V_{\mathbf{p}}a_{\mathbf{p}} & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array}; \quad 2.10: \begin{array}{c|c} V_{\mathbf{p}} & \mathbf{ab} \\ \hline V'_{\mathbf{p}-1} & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array}; \quad 2.11: \begin{array}{c|c} V'_{\mathbf{q}} & \mathbf{a} \\ \hline V_{\mathbf{q}} & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array};$$

$$2.12: \begin{array}{c|c} \mathbf{a} & W_{s-1} \\ \hline Z_2 & W' \end{array} \begin{array}{l} (out) \\ (here) \end{array}; \quad 2.13: \begin{array}{c|c} \mathbf{a} & W' \\ \hline Z_2 & W \end{array} \begin{array}{l} (out) \\ (here) \end{array};$$

$$2.14: \begin{array}{c|c} \mathbf{a} & BW' \\ \hline Z_2 & W_E \end{array} \begin{array}{l} (out) \\ (here) \end{array}; \quad 2.15: \begin{array}{c|c} \mathbf{a} & W'_E \\ \hline Z_2 & W''_E \end{array} \begin{array}{l} (out) \\ (here) \end{array}; \quad 2.16: \begin{array}{c|c} V''_E & \mathbf{a} \\ \hline E & Z_2 \end{array} \begin{array}{l} (here) \\ (out) \end{array};$$

We claim that $L(\Pi) = L(G)$.

We shall prove this assertion in the following way. First we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(\Pi)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. By consequent our assertion will be proved.

Our simulation is based on the "rotate-and-simulate" method, see also Section 4.1.2 and Theorem 4.3.1. We perform the rotation and the simulation of rules of the grammar by rules 1.1 to 1.6 and 2.1 to 2.5. When we have B at the end of the word, we substitute the brackets X and Y by V and W by rules 1.7 to 1.11 and 2.6 to 2.8. After that we perform the rotation step but we rotate only terminal letters of the grammar. The rules 1.12 to 1.16 and 2.9 to 2.13 permit to do this. When we have again the symbol B at the end of the word, we know that the considered string is terminal and that it is in a right form. At this moment we eliminate the parentheses V and W by rules 2.14 to 2.16 and 1.17 to 1.22. We note that, by construction, these rules permit to send outside of the system only terminal words.

We also note that the rules are symmetrical with respect to parentheses X, Y and V, W: the rules 1.2 to 1.11 and 2.1 to 2.8 are identical to rules 1.12 to 1.21 and 2.9 to 2.16 if we make the following transformation: $X \to V$, $Y \to W$, $X_V \to V_E$,

$Y_W \to W_E$ and $0 \to s-1$. This symmetry is due to the fact that we perform two times similar steps: rotation and fixation of the result. The difference is only in the fact that the second time we consider only terminal letters for the rotation.

We remark that this repetition of the rotation was proposed by Gh. Păun in [43], but the obtained system had four membranes.

Notations

We note that bottom site of each rule represents a molecule which is already present in the corresponding membrane. Moreover, one of the results of the splicing will contain Z and thus will not contribute to the production of the result. So, we will omit these molecules and we will write:

$Xwa_iY \underset{1.2}{\longrightarrow} XwY_i(tar_1)$ instead of

$(Xw|a_iY, Z|Y_i) \vdash_{1.2} (XwY_i, Za_iY)\ (tar_1; tar_2)$.

Like in the previous section the rules are constructed in a special way such that if a molecule cannot enter any rule, then it will never enter any rule and it will never contribute to the result. Similarly, we write $w\uparrow$ if the molecule w is in such a case.

We also note that, by construction, the rules from R_1 can be used in the first membrane only, as they contain at leat one target indicator *in*, and the rules from R_2 can be used only in the second membrane, because at least one of the targets is *out*.

Rotation

We start with Xwa_kY in the first membrane.

$Xwa_kY \underset{1.2}{\longrightarrow} XwY_k(in)$,

$XwY_k \underset{2.1}{\longrightarrow} \mathbf{X_ja_jwY_k}(out)$.

We have 4 cases:

a) We have molecules of form $X_0a_kwY_i, i>0$ in the first membrane. Then:

$X_0a_kwY_i \underset{1.5}{\longrightarrow} X'a_kwY_i(in)\uparrow$.

b) We have molecules of form $X_ia_kwY_0, i>0$ in the first membrane. Again they cannot evolve any more:

$X_ia_kwY_0\uparrow$.

c) We have molecules of form $X_ja_kwY_i,\ i,j>0$ in the first membrane. Then we have the following computation:

$X_ja_kwY_i \underset{1.3}{\longrightarrow} X_ja_kwY'_{i-1}(in)$,

$X_ja_kwY'_{i-1} \underset{2.2}{\longrightarrow} X'_{j-1}a_kwY'_{i-1}(out)$,

$X'_{j-1}a_kwY'_{i-1} \underset{1.4}{\longrightarrow} X'_{j-1}a_kwY_{i-1}(in)$,

$X'_{j-1}a_kwY_{i-1} \underset{2.3}{\longrightarrow} \mathbf{X_{j-1}a_kwY_{i-1}}(out)$ or $X'_{j-1}a_kwY_0 \underset{2.4}{\longrightarrow} X'_{j-1}a_kwY'(out)\uparrow$.

So we decreased simultaneously by 1 the indices of X and Y.

d) We have molecules of form $X_0a_kwY_0$ in the first membrane. This gives us:

$X_0a_kwY_0 \underset{1.5}{\longrightarrow} X'a_kwY_0(in)$,

$X'a_kwY_0 \underset{2.4}{\longrightarrow} X'a_kwY'(out)$,
(*) $X'a_kwY' \underset{1.6}{\longrightarrow} Xa_kwY'(in)$,
(**) $Xa_kwY' \underset{2.5}{\longrightarrow} \mathbf{Xa_kwY}(out)$ or $Xa_kwY' \underset{2.1}{\longrightarrow} X_ia_ia_kwY'(out) \uparrow$.

So we rotated Xwa_kY. It is possible to apply the rules 1.7 and 2.6 in cases labeled by (*) and (**), but these applications will be discussed later.

Simulation of productions of the grammar

If we have the word $XwuY$ in the first membrane and if there is a production $u \to v \in P$, then we can apply the rule 1.1 which produces $XwvY$, *i.e.* we simulated the corresponding production of the grammar.

Checking for termination

We obtain $X'wY'$ in the first membrane (case (*)). Now we can use the following rules:

$X'wY' \underset{1.7}{\longrightarrow} X_VBwY'(in)$,
$X_VBwY' \underset{2.6}{\longrightarrow} X_VBw'Y_W(out)$ if $w = w'B$,
$X_VBw'Y_W \underset{1.8}{\longrightarrow} X_VBw'Y'_W(here)$ or $X_VBw'Y_W \underset{1.9}{\longrightarrow} X'_VBw'Y_W(in) \uparrow$,
$X_VBw'Y'_W \underset{1.9}{\longrightarrow} X'_VBw'Y'_W(in)$,
$X'_VBw'Y'_W \underset{2.7}{\longrightarrow} X'_VBw'Y''_W(out)$,
$X'_VBw'Y''_W \underset{1.10}{\longrightarrow} X''_VBw'Y''_W(here)$ or $X'_VBw'Y''_W \underset{1.11}{\longrightarrow} X'_VBw'W(in) \uparrow$,
$X''_VBw'Y''_W \underset{1.11}{\longrightarrow} X''_VBw'W(in)$,
$X''_VBw'W \underset{2.8}{\longrightarrow} \mathbf{VBw'W}(out)$.

The evolution of $VBw'W$ is examined later.

Other possible evolutions:

1. (case (**)) $XwY' \underset{2.6}{\longrightarrow} Xw'Y_W(out)$ $(w = w'B)$,
$Xw'Y_W \underset{1.8}{\longrightarrow} Xw'Y'_W(here) \uparrow$.
2. $X_VBwY' \underset{2.5}{\longrightarrow} X_VBwY(out)$,
$X_VBw'a_iY \underset{1.2}{\longrightarrow} X_VBw'Y_i(in) \uparrow$ or $X_VBwY \underset{1.9}{\longrightarrow} X'_VwY(in) \uparrow$.

Rotation of terminals

Now we perform a rotation of terminal letters of the grammar only, *i.e.* we check if $w' \in T^*$. The computation is very similar to the general rotation, see above, and if we substitute X by V, Y by W and 0 by $s-1$, then we obtain exactly the same evolution.

We have Vwa_kW in the first membrane.

$Vwa_kW \underset{1.12}{\longrightarrow} VwW_k(in)$,
$VwW_k \underset{2.9}{\longrightarrow} \mathbf{V_ja_jwW_k}(out)$.

We have now 4 cases:
a) We have molecules of form $V_{s-1}a_kwW_i, i > s-1$ in the first membrane. Then:
$V_{s-1}a_kwW_i \underset{1.15}{\longrightarrow} V'a_kwW_i(in) \uparrow$.
b) We have molecules of form $V_ia_kwW_{s-1}, i > s-1$ in the first membrane. Again they cannot evolve any more:
$V_ia_kwW_{s-1} \uparrow$.
c) We have molecules of form $V_ja_kwW_i,\ i,j > s-1$ in the first membrane. Then we have the following computation:
$V_ja_kwW_i \underset{1.13}{\longrightarrow} V_ja_kwW'_{i-1}(in)$,
$V_ja_kwW'_{i-1} \underset{2.10}{\longrightarrow} V'_{j-1}a_kwW'_{i-1}(out)$,
$V'_{j-1}a_kwW'_{i-1} \underset{1.14}{\longrightarrow} V'_{j-1}a_kwW_{i-1}(in)$,
$V'_{j-1}a_kwW_{i-1} \underset{2.11}{\longrightarrow} \mathbf{V_{j-1}a_kwW_{i-1}}(out)$ or $V'_{j-1}a_kwW_0 \underset{2.12}{\longrightarrow} V'_{j-1}a_kwW'(out) \uparrow$.
So we decreased simultaneously by 1 the indices of V and W.
d) We have molecules of form $V_{s-1}a_kwW_{s-1}$ in the first membrane. This gives us:
$V_{s-1}a_kwW_{s-1} \underset{1.15}{\longrightarrow} V'a_kwW_{s-1}(in)$,
$V'a_kwW_{s-1} \underset{2.12}{\longrightarrow} V'a_kwW'(out)$,
(+) $V'a_kwW' \underset{1.16}{\longrightarrow} Va_kwW'(in)$,
(++) $Va_kwW' \underset{2.13}{\longrightarrow} \mathbf{Va_kwW}(out)$ or $Va_kwW' \underset{2.10}{\longrightarrow} V_ia_ia_kwW'(out) \uparrow$.
So we rotated Vwa_kW. It is possible to apply the rules 1.17 and 2.15 in cases labeled by (+) and (++), but these applications will be discussed later.

Obtention of the result

We obtain $V'wW'$ in the first membrane (case (+)). Now we can use the following rules:
$V'wW' \underset{1.17}{\longrightarrow} V_EwW'(in)$,
$V_EwY' \underset{2.14}{\longrightarrow} X_Vw'W_E(out)$ if $w = w'B$,
$V_Ew'W_E \underset{1.18}{\longrightarrow} V_Ew'W'_E(here)$ or $V_Ew'W_E \underset{1.19}{\longrightarrow} V'_Ew'W_E(in) \uparrow$,
$V_Ew'W'_E \underset{1.19}{\longrightarrow} V'_Ew'W'_E(in)$,
$V'_Ew'W'_E \underset{2.15}{\longrightarrow} V'_Ew'W''_E(out)$,
$V'_Ew'W''_E \underset{1.20}{\longrightarrow} V''_Ew'W''_E(here)$ or $V'_Ew'W''_E \underset{1.21}{\longrightarrow} V'_Ew'(in) \uparrow$,
$V''_Ew'W''_E \underset{1.21}{\longrightarrow} V''_Ew'(in)$,
$V''_Ew' \underset{2.16}{\longrightarrow} Ew'(out)$,
$Ew' \underset{1.22}{\longrightarrow} \mathbf{w'}(out)$.
In this case the word w' which is terminal is sent outside of the system and it will belong to the result of the computation.
Other possible evolutions:
1. (case (++)) $VwW' \underset{2.14}{\longrightarrow} Vw'W_E(out)$ $(w = w'B)$,
$Vw'W_E \underset{1.18}{\longrightarrow} Vw'W'_E(here) \uparrow$.

2. $V_E w W' \underset{2.13}{\longrightarrow} V_E w W(out)$,
$V_E w' a_p W \underset{1.12}{\longrightarrow} V_E w' W_p(in) \uparrow$ ou $V_E w W \underset{1.19}{\longrightarrow} V'_E w W(in) \uparrow$.

It is easy to see that by following the computation above we generate all words of $L(G)$ and, as we considered all possible cases, it is clear that the system does not produce other words. □

9.3 Splicing P systems with immediate communication

In this section we consider splicing P systems with immediate communication which differ from the systems introduced in Section 9.1 by the fact that the result of each splicing shall move from the membrane in which it was produced. More exactly, we can see an splicing P system with immediate communication as a splicing P system whose rules do not contain the target indicator *here* and in the same time each splicing rule appears with the four possible combinations of target indicators *in* and *out*.

We denote by $ELSP_m(spl, move)$ the family of languages generated by splicing P systems with immediate communication of degree at most m. Like in the previous section we do not indicate the type of the considered system, because we can avoid the problems which forced us to introduce different definitions.

If we have only one membrane, then at each step we send outside of the system the same molecules which are the result of splicing of axioms. By consequent, we have the following result:

Theorem 9.3.1. $ELSP_1(spl, move) = FIN$.

Two membranes are already enough in order to have a big computational power.

Theorem 9.3.2. *Let $G = (N, T, S, P)$ be a type-0 grammar. Then, there is a splicing P system with immediate communication of degree 2, Π, which simulates G and $L(\Pi) = L(G)$.*

Proof. We construct $\Pi = (V, T, [_1[_2\,]_2]_1, M_1, M_2, R_1, R_2))$ as follows.

Let $N \cup T \cup \{B\} = \{a_1, a_2, \ldots, a_n\}$ $(a_n = B)$ and $B \notin \{N \cup T\}$.

In what follows we assume that:
$1 \le \mathbf{i} \le n$, $0 \le \mathbf{j} \le n-1$, $1 \le \mathbf{k} \le n$, $\mathbf{a} \in N \cup T \cup \{B\}$.

The alphabet V is defined by:
$V = N \cup T \cup \{B\} \cup \{X, Y, X_\mathbf{i}, Y_\mathbf{i}, X_0, Y_0, X'_\mathbf{j}, Y'_\mathbf{j}, X', Y', Z, Z_X, Z_Y\}$.

The terminal alphabet T is the same as for the formal grammar G.

The axioms are defined by:

$M_1 = \{XSBY\} \cup \{ZvY_\mathbf{j} \mid u \to va_\mathbf{j} \in P\} \cup \{ZY_\mathbf{i}, ZY_0, ZY'_\mathbf{j}, X'Z, XZ, Z_X\}$.

$M_2 = \{X_\mathbf{i} a_\mathbf{i} Z, X'_\mathbf{j} Z, X_\mathbf{j} Z, ZY', ZY, Z_Y\}$.

The rules of the system are defined as follows.

Rules of R_1:

$$1.1 : \begin{array}{c|c} \varepsilon & uY \\ \hline Z & vY_{\mathbf{j}} \end{array}, \quad \text{if } \exists u \to va_{\mathbf{j}} \in P; \qquad 1.1' : \begin{array}{c|c} \varepsilon & a_{\mathbf{i}}uY \\ \hline Z & Y_{\mathbf{i}} \end{array}, \quad \text{if } \exists u \to \varepsilon \in P;$$

$$1.2 : \begin{array}{c|c} \varepsilon & a_{\mathbf{i}}Y \\ \hline Z & Y_{\mathbf{i}} \end{array}; \qquad 1.3 : \begin{array}{c|c} \mathbf{a} & Y_{\mathbf{k}} \\ \hline Z & Y'_{\mathbf{k-1}} \end{array}; \qquad 1.4 : \begin{array}{c|c} \mathbf{a} & Y'_{\mathbf{j}} \\ \hline Z & Y_{\mathbf{j}} \end{array};$$

$$1.5 : \begin{array}{c|c} X_0 & \mathbf{a} \\ \hline X' & Z \end{array}; \qquad 1.6 : \begin{array}{c|c} X' & \mathbf{a} \\ \hline X & Z \end{array}; \qquad 1.7 : \begin{array}{c|c} X' & \mathbf{a} \\ \hline \varepsilon & Z_X \end{array}.$$

Rules of R_2:

$$2.1 : \begin{array}{c|c} X & \mathbf{a} \\ \hline X_{\mathbf{i}}a_{\mathbf{i}} & Z \end{array}; \qquad 2.2 : \begin{array}{c|c} X_{\mathbf{k}} & \mathbf{a} \\ \hline X'_{\mathbf{k-1}} & Z \end{array}; \qquad 2.3 : \begin{array}{c|c} X'_{\mathbf{j}} & \mathbf{a} \\ \hline X_{\mathbf{j}} & Z \end{array}; \qquad 2.4 : \begin{array}{c|c} \mathbf{a} & Y_0 \\ \hline Z & Y' \end{array};$$

$$2.5 : \begin{array}{c|c} \mathbf{a} & Y' \\ \hline Z & Y \end{array}; \qquad 2.6 : \begin{array}{c|c} \mathbf{a} & BY_0 \\ \hline Z_Y & \varepsilon \end{array}.$$

We claim that $L(\Pi) = L(G)$.

We shall prove this assertion in the following way. First we show how we can simulate the derivations of the formal grammar G. In this way we prove that $L(G) \subseteq L(\Pi)$. In the same time we consider all other possible evolutions and we show that they do not lead to a terminal string. By consequent our assertion will be proved.

Our simulation is based on the "rotate-and-simulate" method, see also Section 4.1.2 and Theorem 4.3.1. We start with the word Xwa_iY in the first membrane. After the application of rule 1.2 the word XwY_i is sent to the second membrane. After that, the rule 2.1 is applied and the first membrane receives words $X_ja_jwY_i$ ($1 \leq j \leq n$). After that point the system works in a cycle where indices i and j decrease simultaneously. Words for which $j \neq i$ are eliminated and only words of type $X_0a_iwY_0$ remain. After that we obtain the word Xa_iwY, so we rotated the word Xwa_iY. If $a_iw = a_iw'B$ and $a_iw' \in T^*$, then this word which belongs to the result is also produced and send outside of the system.

We will emphasise certain similarities between the constructed system and TVDH systems of degree 2, in particular, the system D_G presented in Theorem 4.3.1.

- The membranes can be seen as components.
- Only the result of the computation participate in the computation in the next component, respectively next membrane.
- If a word cannot enter any rule, then it is eliminated. In the case of the system Π this word will never participate to a splicing, so it will never contribute to the result.

In order to obtain the third point we constructed the rules in a special way: one can see that lower site of each rule represents a molecule which is already present in the membrane. Therefore, if a molecule cannot enter any rule then it will never enter a rule, hence it will never contribute to the result. We write $w\uparrow$ if the molecule w is in a such case.

However there are differences between the system that we just constructed and the system D_G:

- If a molecule can enter a rule, then this molecule "lives" forever in the corresponding membrane, so it will participate at each step in a splicing.

Since we simulate a grammar, at each step we have several molecules which encode different branches of the simulation and which evolve in parallel. This is why our system is constructed in a way that permits to handle this parallelism. By consequent, the differences above are not important for our proof.

So, the proof is very similar to the proof of Theorem 4.3.1 and the computation of Π follows closely the computation of D_G.

Using the same arguments as the ones used during the proof of Theorem 9.2.2, we shall write:

$Xwa_iY \underset{1.2}{\longrightarrow} XwY_i$ instead of

$(Xw|a_iY, Z|Y_i) \vdash_{1.2} (XwY_i, Za_iY)$.

We also note that the resulting molecule is immediately sent outside the membrane in which it was produced. Therefore, at each step this molecule changes the membrane. By consequent, it is not necessary to indicate the current membrane, because the indication of rule numbers suffices to solve any ambiguities.

Rotation

We start with Xwa_kY in the first membrane.

$Xwa_kY \underset{1.2}{\longrightarrow} XwY_k$,

$XwY_k \underset{2.1}{\longrightarrow} \mathbf{X_ja_jwY_k}$.

We have now 4 cases:

a) We have molecules of form $X_0a_kwY_i, i>0$ in the first membrane. Then:

$X_0a_kwY_i \underset{1.5}{\longrightarrow} X'a_kwY_i\uparrow$.

b) We have molecules of form $X_ia_kwY_0, i>0$ in the first membrane. Again they cannot evolve any more:

$X_ia_kwY_0\uparrow$.

c) We have molecules of form $X_ja_kwY_i,\ i,j>0$ in the first membrane. Then we have the following computation:

$X_ja_kwY_i \underset{1.3}{\longrightarrow} X_ja_kwY'_{i-1}$,

$X_ja_kwY'_{i-1} \underset{2.2}{\longrightarrow} X'_{j-1}a_kwY'_{i-1}$,

$X'_{j-1}a_kwY'_{i-1} \underset{1.4}{\longrightarrow} X'_{j-1}a_kwY_{i-1}$,

$X'_{j-1}a_kwY_{i-1} \underset{2.3}{\longrightarrow} \mathbf{X_{j-1}a_kwY_{i-1}}$, or

$X'_{j-1}a_kwY_0 \underset{2.4}{\longrightarrow} X'_{j-1}a_kwY'\uparrow$ $(X'_{j-1}a_kwY_0 \underset{2.6}{\longrightarrow} X'_{j-1}a_kw\uparrow)$.

So we decreased simultaneously by 1 the indices of X and Y.

d) We have molecules of form $X_0a_kwY_0$ in the first membrane. This gives us:

$X_0a_kwY_0 \underset{1.5}{\longrightarrow} X'a_kwY_0$,
(*) $X'a_kwY_0 \underset{2.4}{\longrightarrow} X'a_kwY'$,
(**) $X'a_kwY' \underset{1.6}{\longrightarrow} Xa_kwY'$,
$Xa_kwY' \underset{2.5}{\longrightarrow} \mathbf{Xa_kwY}$ or $Xa_kwY' \underset{2.1}{\longrightarrow} X_ia_ia_kwY' \uparrow$.

So we rotated Xwa_kY. It is possible to apply the rules 2.6 and 1.7 in cases labeled by (*) and (**), but these applications will be discussed later.

Simulation of productions of the grammar

If we have the word $XwuY$ in the first membrane and if there is a production $u \to v$ in P, then we can apply the rule 1.1 or 1.1′ in order to simulate the corresponding production of the grammar and after that we continue as it is described above.

Obtention of the result

We obtain $X'a_kwY_0$ in the second membrane (case (*)). Now we can use the following rules:
$X'a_kw'BY_0 \underset{2.6}{\longrightarrow} X'a_kw'$ $(w = w'B)$,
$X'a_kw' \underset{1.7}{\longrightarrow} a_kw'$.
In this case if $a_kw' \in T^*$, then it will belong to the result.
We can also apply the rule 1.6:
$X'a_kw' \underset{1.6}{\longrightarrow} Xa_kw'$,
$Xa_kw' \underset{2.1}{\longrightarrow} X_ia_ia_kw' \uparrow$.

Other possible evolutions:

We have the word $X'a_kwY'$ in the first membrane (case (**)).
$X'a_kwY' \underset{1.7}{\longrightarrow} a_kwY'$,
$a_kwY' \underset{2.5}{\longrightarrow} a_kwY$,
$a_kw'a_iY \underset{1.2}{\longrightarrow} a_kw'Y_i \uparrow$.

It is easy to see that by following the computation above we generate all words of $L(G)$ and, as we considered all possible cases, it is clear that the system does not produce other words. Moreover, the computation follows closely the computation of systems presented in Theorems 4.3.1 and 9.2.2. □

9.4 Conclusions

In this section we gather certain common features of systems constructed in this chapter.

We note that all systems from this chapter eliminate undesirable molecules in two ways. The first possibility consists in sending these molecules outside of the system, while the second one keeps them into membranes in a way that do not permit their further utilisation. In order to achieve this, the rules are made in a special way and if a molecule cannot enter any rule, then it will never participate in a splicing, by consequent, it will never contribute to the result. These eliminations which we call external and, respectively, internal play an important role during the computation.

We also remark that the systems presented in this chapter have strong connections with TVDH systems. Indeed, the computation in some of the systems from this chapter, modulo internal elimination, follows closely the computation of the corresponding TVDH systems.

Finally, the method of directing molecules is applicable for the systems which permit a strong external elimination, more exactly the splicing P systems of type 1, 2b and 2c.

Conclusions

We have studied in this work H systems and their extensions, or, more generally, different systems based on splicing. We have seen that this operation is very powerful and only small additions are necessary to reach the computational power of a Turing machine. This is not surprising, because the splicing operation has elements depending on the context, and the context dependence is almost the universality in the theory of formal languages.

We have seen that, without addition of other elements, the splicing operation permits to generate only regular languages. We studied for the first time the relation between 1-splicing and 2-splicing and showed that the family of 2-splicing languages is strictly included in the family of 1-splicing languages. We also found non-trivial examples of regular languages which cannot be splicing languages.

Then we studied TVDH systems which are simple and powerful at the same time. We have shown that these systems have a big computational power with two components and even with one component only. We also showed with the help of a computer program, which we developed before, that certain articles on TVDH systems have errors, and we showed how it is possible to correct them.

The main point of this thesis, the *method of directing molecules*, permitted us to uniform and simplify proofs of several results. TVDH systems, ETVDH systems, test tube systems with alternating filters, modified test tube systems and even splicing P systems constituted the field for application of this method introduced by us. More generally, we can say that splicing operation combined with elimination of molecules in the sense described at the end of Chapter 6 is sufficient to obtain the computational power of a Turing machine. Thus we recapitulate some reflections on splicing made by different authors.

Similarly, the splicing operation in combination with membranes gives a powerful tool and it is not surprising that we easily obtain a big computational power.

This work leaves several open problems. First, we made a small contribution to the resolution of the problem of characterisation of splicing languages. As it it shown in our work, now we need to consider now two cases: 1-splicing languages and 2-splicing languages.

We also showed that TVDH systems are a good candidate for a "Turing machine" in the area of molecular computing. As we could see, several systems have similarities with TVDH systems, or may be even reduced to them. We think that a study in this direction may give interesting results. This is why we also presented in Chapter 4 an universal TVDH system which has a very compact description. We shall continue investigations in this direction in order to find, in a similar way to what it was done for Turing machines, systems which will be smaller and smaller. We think that the research of minimal systems in combination with the simple structure and functioning of TVDH systems may bring interesting results.

Another research direction consist in study and further development of the method of directing molecules. Since it is a very generic method, we think that it may have a lot of applications for different systems based on splicing, and also for other types of n-ary operations, with $n > 2$. For example, as we closed the chapter of splicing P systems by showing final results, we hope to apply the method of directing molecules over other types of membrane systems, even those not based on splicing.

We also approached the resolution of the problem of the computational power of test tube systems with two tubes. The variants that we proposed are closer and closer to the original definition. This is why we hope that the work that we have done will help to solve this open problem. These variants have also interesting properties like fixed and mobile garbage collectors, and their study is not finished yet.

In conclusion, we want to add that the study of biologically inspired systems is very exciting and can bring colossal gains. Maybe it is the time to learn new things and, by reformulating Wordsworth, [57], “let the nature be our teacher”.

Bibliography

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, **266**, (1994), 1021–1024.

[2] P. Bonizzoni, C. D. Felice, G. Mauri, and R. Zizza. Regular languages generated by reflexive finite splicing systems. In Z. Esik and Z. Fulop, editors, *Proceedings of Developments in Language Theory 2003, DLT03, Szeged, Hungary, July 7-11, 2003.* 2003, volume 2710 of *Lecture Notes in Computer Science*, 134–145.

[3] P. Bonizzoni, C. D. Felice, G. Mauri, and R. Zizza. The structure of reflexive finite splicing languages via Schützenberger constants, 2004. Submitted.

[4] J. Cocke and M. Minsky. Universality of tag systems with p=2. *Journal of the ACM*, **11**(1), (1964), 15–20.

[5] E. Csuhaj-Varjú, L. Kari, and G. Păun. Test tube distributed systems based on splicing. *Computers and AI*, **15**(2–3), (1996), 211–232.

[6] K. Culik II and T. Harju. Splicing semigroups of dominoes and DNA. *Discrete Applied Mathematics*, **31**, (1991), 261–277.

[7] F. Freund, R. Freund, M. Margenstern, M. Oswald, Y. Rogozhin, and S. Verlan. P systems with cutting/recombination rules assigned to membranes. In A. Alhazov, C. Martín-Vide, and G. Păun, editors, *Preproceedings of the Workshop on Membrane Computing. Tarragona, July 17-22, 2003.* 2003, 241–251.

[8] R. Freund and F. Freund. Test tube systems: when two tubes are enough. In G. Rozenberg and W. Thomas, editors, *Preproceedings of DLT99, Developments in Language Theory.* World Scientific Publishing Company, 2000, 275–286.

[9] P. Frisco. On two variants of splicing super-cell systems. *Romanian Journal of Information Science and Technology*, **4**(1-2), (2001), 89–100.

[10] P. Frisco and C. Zandron. On variants of communicating distributed H systems. *Fundamenta Informaticae*, **48**(1), (2001), 9–20.

[11] T. E. Goode Laun. *Constants and Splicing Systems.* Ph.D. thesis, Binghamton University, 1999.

[12] 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.

[13] T. Head. Splicing languages generated with one sided context. In G. Păun, editor, *Computing with Bio-Molecules. Theory and Experiments.* Springer Verlag, Berlin, Heidelberg, New York, 1998, 158–181.

[14] T. Head, G. Păun, and D. Pixton. Language theory and molecular genetics. generative mechanisms suggested by DNA recombination. In *Handbook of Formal Languages, 3 volumes* [47], 295–360.

[15] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 2nd edition, 2001.

[16] L. Kari. DNA computing: arrival of biological mathematics. *The Mathematical Intelligencer*, **19**(2), (1997), 9–22. Earlier version under the title DNA computers, tomorrow's reality. Bulletin of the European Association for Theoretical Computer Science, (59):256–266, June 1996 `http://www.csd.uwo.ca/ lila/amsn.ps`.

[17] S. M. Kim. An algorithm for identifying spliced languages. In *Proceedings of Cocoon97.* 1997, volume 1276 of *Lecture Notes in Computer Science*, 403–411.

[18] S. Kleene. *Introduction to Metamathematics.* Van Nostrand Comp. Inc., New-York, 1952.

[19] R. J. Lipton. Using DNA to solve NP-complete problems. *Science*, **268**, (1995), 542–545.

[20] V. Manca. A proof of regularity for finite splicing. In N. Jonoska, G. Paun, and G. Rozenberg, editors, *Aspects of Molecular Computing. Essays Dedicated to Tom Head on the Occasion of His 70th Birthday*, Springer Verlag, Berlin, Heidelberg, New York, volume 2950 of *LNCS.* 2004, 309–317.

[21] M. Margenstern and Y. Rogozhin. Generating all recursively enumerable languages with a time-varying distributed H system of degree 2. Technical report, Institut Universitaire de Technologie de Metz, 1999. Publications du G.I.F.M.

[22] M. Margenstern and Y. Rogozhin. A universal time-varying distributed H-system of degree 2. In L. Kari, H. Rubin, and D. H. Wood, editors, *Proceedings of the 4th DIMACS meeting on DNA based computers.* Elsevier, 1999, volume 52, (1–3), 73–80.

[23] M. Margenstern and Y. Rogozhin. About time-varying distributed H systems. In A. Condon and G. Rozenberg, editors, *DNA Computing: 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands,*

June 13-17, 2000, Revised Papers. Springer Verlag, Berlin, Heidelberg, New York, 2000, volume 2054 of *Lecture Notes in Computer Science*, 53–62.

[24] M. Margenstern and Y. Rogozhin. Time-varying distributed H systems of degree 2 generate all recursively enumerable languages. In Martin-Vide and Mitrana [31], 399–407.

[25] M. Margenstern and Y. Rogozhin. Extended time-varying distributed H systems - universality result. In *Proceedings of The 5th World Multi-Conference on Systemics, Cybernetics and Informatics, Industrial Systems, SCI 2001, Orlando, Florida USA, July 22-25, 2001.* 2001, volume IX.

[26] M. Margenstern and Y. Rogozhin. Time-varying distributed H systems of degree 1 generate all recursively enumerable languages. In M. Ito, G. Păun, and S. Yu, editors, *Words, Semigroups, and Transductions*, World Scientific, Singapore. 2001, 329–340. Festschrift in Honor of Gabriel Thierrin.

[27] M. Margenstern and Y. Rogozhin. A universal time-varying distributed H system of degree 1. In N. Jonoska and N. C. Seeman, editors, *DNA Computing: 7th International Workshop on DNA-Based Computers, DNA7, Tampa, FL, USA, June 10-13, 2001. Revised Papers.* Springer Verlag, Berlin, Heidelberg, New York, 2002, volume 2340, 371–380.

[28] M. Margenstern, Y. Rogozhin, and S. Verlan. Time-varying distributed H systems of degree 2 can carry out parallel computations. 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.* Hokkaido University, Springer Verlag, Berlin, Heidelberg, New York, 2002, volume 2568 of *Lecture Notes in Computer Science*, 326–336.

[29] M. Margenstern, Y. Rogozhin, and S. Verlan. Time-varying distributed H systems with parallel computations: the problem is solved. In *Preliminary Proceedings of DNA Computing, 9th international Workshop on DNA-Based Computers, DNA 2003, Madison, Wisconsin, USA, 1-4 June 2003.* 2003, 47–51.

[30] C. Martin-Víde, G. Păun, and A. Rodríguez-Patón. P systems with immediate communication. *Romanian Journal of Information Science and Technology*, **4**(1-2), (2001), 171–182.

[31] C. Martin-Vide and V. Mitrana, editors. *Where Do Mathematics, Computer Science and Biology Meet.* Kluwer Academic, Dortrecht, 2000.

[32] M. Minsky. *Computations: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffts, NJ, 1967.

[33] D. Pixton. Regularity of splicing languages. *Discrete Applied Mathematics*, **69**(1–2), (1996), 101–124.

[34] L. Priese, Y. Rogozhin, and M. Margenstern. Finite H-systems with 3 test tubes are not predictable. In R. Altman, A. Dunker, L. Hunter, and T. Klein, editors, *Proceedings of Pacific Symposium on Biocomputing, 3, Kapalua, Maui, January 1998, Hawaii, USA*. World Scientific Publishing Company, 1998, 547–558.

[35] A. Păun. On time-varying H systems. *Bulletin of EATCS*, **67**, (1999), 157–164.

[36] A. Păun and M. Păun. On the membrane computing based on splicing. In Martin-Vide and Mitrana [31], 409–422.

[37] G. Păun. On the splicing operation. *Discrete Applied Mathematics*, **70**(1), (1996), 57–79.

[38] G. Păun. Regular extended H systems are computationally universal. *JALC*, **1**(1), (1996), 27–36.

[39] G. Păun. DNA computing: distributed splicing systems. In J. Mycielsky, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science. A Selection of Essays in Honor of A. Ehrenfeucht.* Springer Verlag, Berlin, Heidelberg, New York, 1997, volume 1261 of *Lecture Notes in Computer Science*, 351–370.

[40] G. Păun. DNA computing based on splicing: universality results. In M. Margenstern, editor, *Proceedings of the Second Internetional Colloquium on Universal Machines and Computations, Metz, France.* IUT de Metz, 1998, volume I, 67–91.

[41] G. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **1**(61), (2000), 108–143. Also TUCS Report No. 208, 1998.

[42] G. Păun. DNA computing based on splicing: universality results. *Theoretical Computer Science*, **231**(2), (2000), 275–296. Journal version of [40].

[43] G. Păun. *Membrane Computing. An Introduction.* Springer Verlag, Berlin, Heidelberg, New York, 2002.

[44] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms.* Springer Verlag, Berlin, Heidelberg, New York, 1998.

[45] G. Păun and T. Yokomori. Membrane computing based on splicing. In E. Winfree and D. K. Gifford, editors, *DNA Based Computers V.* American Mathematical Society, 1999, volume 54 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 217–232.

[46] Y. Rogozhin. Small universal turing machines. *Theoretical Computer Science*, **168**(2), (1996), 215–240.

[47] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages, 3 volumes.* Springer Verlag, Berlin, Heidelberg, New York, 1997.

[48] M. P. Schutzenberger. Sur certaines opérations de fermeture dans les langages rationnels. *Symposia Mathematica*, **15**, (1975), 245–253.

[49] TVDHsim: Time-varying distributed H systems simulator. `http://lita.sciences.univ-metz.fr/~verlan/`.

[50] S. Verlan. On extended time-varying distributed H systems. In *Preproceedings at 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands, June 13–17, 2000.* 2000, 281. Poster.

[51] S. Verlan. *Calculs Moléculaires: les Systèmes Distribués à Changement de Phase.* Master's thesis, Université de Metz, 2001.

[52] S. Verlan. On enhanced time-varying distributed H systems. *Computer Science Journal of Moldova*, **10**(3), (2002), 263–279. Kishinev.

[53] S. Verlan. About splicing P systems with immediate communication and non-extended splicing P systems. In A. Alhazov, C. Martín-Vide, and G. Păun, editors, *Preproceedings of the Workshop on Membrane Computing. Tarragona, July 17-22, 2003.* 2003, 461–473.

[54] S. Verlan. A frontier result on enhanced time-varying distributed H systems with parallel computations. In *Preproceedings of DCFS'03, Descriptional Complexity of Formal Systems, Budapest, Hungary, July 12-14, 2003.* 2003, 221–232.

[55] S. Verlan. Communicating distributed H systems with alternating filters. In N. Jonoska, G. Paun, and G. Rozenberg, editors, *Aspects of Molecular Computing. Essays Dedicated to Tom Head on the Occasion of His 70th Birthday*, Springer Verlag, Berlin, Heidelberg, New York, volume 2950 of *LNCS*. 2004, 367–384.

[56] S. Verlan and R. Zizza. 1-splicing vs. 2-splicing: separating results. In *Proceedings of WORDS'03, Turku, Finland, September 10-13, 2003.* 2003, 320–331.

[57] W. Wordsworth. The tables turned. In *Wordworth's Poems*, Ed. P. Wayne, Dent, London, volume 1996. 1965.

[58] C. Zandron. The P systems web page. `http://psystems.disco.unimib.it/`.

[59] C. Zandron, C. Ferretti, and G. Mauri. A reduced distributed splicing system for RE languages. In G. Păun and A. Salomaa, editors, *New trends in Formal Language. Control, cooperatio, and conbinatorics.* Springer Verlag, Berlin, Heidelberg, New York, 1997, volume 1218 of *Lecture Notes in Computer Science*, 346–366.

[60] C. Zandron, C. Ferretti, and G. Mauri. Nine test tubes generate any RE language. *Theoretical Computer Science*, **231**(2), (2000), 171–180.

[61] R. Zizza. *On the power of classes of splicing systems.* Ph.D. thesis, University of Milan, 2002.